Practical Structured Learning Techniques for Natural Language Processing

by

Harold Charles Daumé III

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

August 2006

"Arrest this man, he talks in maths..."

Radiohead, *Karma Police*

Dedication

For Kathy, who keeps me sane and happy. . .

Acknowledgments

My thesis work has benefited tremendously from the influence, advice and support of many colleagues, friends and family. I am eternally grateful to my adviser, Daniel Marcu, for continuous help and support throughout my graduate career. Daniel's grounding kept me focused, but I am equally indebted to his support while I found my own path. Many thanks also to the other members of my committee, especially Stefan Schaal (to whom most blame goes for my interest in machine learning) and Andrew McCallum (discussions with whom have greatly improved this work). The other members of my thesis committee—Ed Hovy, Kevin Knight and Gareth James—have provided consistently useful feedback. Many thanks also to John Langford for pushing me to also consider the theoretical implications of this work. Many of the theoretical results in this thesis are due to interactions with John, especially the central convergence theorem in Chapter 3.

My path to NLP was a circuitous one. Many thanks to Chris Quirk for pointing me to LTI back at CMU when I didn't even know NLP was a field. The entire LTI crowd was incredibly supportive as I was getting to know the field, especially Eric Nyberg, Teruko Mitamura, Lori Levin and Alon Lavie who guided my first year's travels through this field. My LTI experiences were made enjoyable by interactions with many other faculty and students, especially Kathrin Probst, Alicia Tribble, Rosie Jones and Ben Han.

Many thanks to Alex Fraser, my fantastic officemate, for both providing a sounding board for ideas and teaching me how to use the ISI espresso machine. I'm also greatly indebted to Mike Collins, Fernando Pereira and Ben Taskar: their encouragement and enthusiasm has been priceless. Discussions with others, inside and outside ISI, have greatly influenced this work. At the risk of forgetting someone, I have greatly enjoyed my interactions with: Drew Bagnell, Jeff Bilmes, John Blitzer, Eric Brill, Mike Collins, Kevin Duh, Mike Fleischman, Matti Kääriäinen, Sham Kakade, Philipp Koehn, Chin-Yew Lin, David McAllester, Ryan McDonald, Dragos Munteanu, Ani Nenkova, Franz Och, Bo Pang, Patrick Pantel, Deepak Ravichandran, Radu Soricut, Charles Sutton, Yee-Whye Teh, Liang Zhou.

Finally, I thank my family and friends both for their support during this thesis as well as before it. My parents always encouraged me intellectually and provided for me a fantastic education. My friendships have salvaged my sanity on several occasions. A special thanks to Jason Cheng, Rob Pierry, Charlie Sharp, Dane Tice and Mark Yohalem for non-academic support. My love and gratitude goes out to Kathy, who now knows much more about natural language processing and machine learning than she probably ever wanted to. Her support, through the good times and the bad, was a necessary nutrient for this thesis to properly develop.

Contents

List Of Tables

List Of Figures

Abstract

Natural language processing is replete with problems whose outputs are highly complex and structured. The current state-of-the-art in machine learning is not yet sufficiently general to be applied to general problems in NLP. In this thesis, I present SEARN (for "search-learn"), an approach to learning for structured outputs that *is* applicable to the wide variety of problems encountered in natural language (and, hopefully, to problems in other domains, such as vision and biology). To demonstrate SEARN's general applicability, I present applications in such diverse areas as automatic document summarization and entity detection and tracking. In these applications, SEARN is empirically shown to achieve state-of-the-art performance.

SEARN is based on an integration of *learning* and *search.* This contrasts with standard approaches that define a model, learn parameters for that model, and then use the model and the learned parameters to produce new outputs. In most NLP problems, the "produce new outputs" step includes an intractable computation. One must therefore employ a heuristic search function for the production step. Instead of shying away from search, SEARN attacks it head on and considers structured prediction to be *defined* by a search problem. The corresponding learning problem is then made natural: learn parameters so that search succeeds.

The two application domains I study most closely in this thesis are entity detection and tracking (EDT) and automatic document summarization. EDT is the problem of finding all references to people, places and organizations in a document and identifying their relationships. Summarization is the task of producing a short summary for either a single document or for a collection of documents. These problems exhibit complex structure that cannot be captured and exploited using previously proposed structured prediction algorithms. By applying Searn to these problems, I am able to learn models that benefit from complex, non-local features of both the input *and* the output. Such features would not be available to structured prediction algorithm that require model tractability. These improvements lead to state-of-the-art performance on standardized data sets with low computational overhead.

Searn operates by transforming structured prediction problems into a collection of classification problems, to which any standard binary classifier may be applied (for instance, a support vector machine or decision tree). In fact, Searn represents a family of structured prediction algorithms depending on the classifier and search space used. From a theoretical perspective, Searn satisfies a strong fundamental performance guarantee: given a good classification algorithm, Searn yields a good structured prediction algorithm. Such theoretical results are possible for other structured prediction only when the underlying model is tractable. For Searn, I am able to state strong results that are independent of the size or tractability of the search space. This provides theoretical justification for integrating search with learning.

Chapter 1

Introduction

I present an efficient, theoretically justified learning algorithm for *structured prediction* that achieves state-of-the-art performance in a wide range of natural language processing problems. Structured prediction is a generalized task that encompasses many problems in natural language processing, as well as many problems from computational biology, computational vision and other areas. The key issue in structured prediction that differentiates it from more canonical machine learning tasks (such as classification or regression) is that the objects being predicted have internal structure. Adequately representing this internal structure is key to obtaining good solutions to real-world problems, and an algorithm that can function under any notion of structure is to be preferred to one with restricted applicability.

1.1 Structure in Language

Many tasks in natural language processing can be formulated as mappings from inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. For example, in machine translation, $\mathcal{X}$ might be the set of all French sentences and $\mathcal{Y}$ might be the set of all English sentences. In this setting, one can view machine translation as the task of developing a mapping from $\mathcal{X}$ to $\mathcal{Y}$ that obeys some properties (adequacy of the translation to the original and fluency of the translation). Other common NLP tasks also fit naturally into this framework. In automatic document summarization, $x \in \mathcal{X}$ is a document (or document collection) and $y \in \mathcal{Y}$ is a summary. In information extraction, $x \in \mathcal{X}$ is a document and $y \in \mathcal{Y}$ is the relevant "information" contained in x. In sequence labeling and parsing, x is a sentence and y is the corresponding annotation.

For each of these problems, specialized solutions have been developed. Beginning with the influential work in machine translation by Brown et al. (1993), we have witnessed a burgeoning of statistical approaches to natural language problems. We have high performance models for machine translation (Och, 2003), parsing (Collins, 2003; Charniak and Johnson, 2005), information extraction (Bikel, Schwartz, and Weischedel, 1999; Florian et al., 2004; Wellner et al., 2004), summarization (Knight and Marcu, 2002; Barzilay, 2003; Zajic, Dorr, and Schwartz, 2004), part of speech tagging (Brill, 1995) and syntactic chunking (Punyakanok and Roth, 2001; Zhang, Damerau, and Johnson, 2002; Sutton, Rohanimanesh, and McCallum, 2004; Sutton, Sindelar, and McCallum, 2005), to name a

$\underline{\text{JERUSALEM}}^{\text{NAM}}_{\text{GPE}-1}$ – The $\underline{\text{commander}}^{\text{NOM}}_{\text{PER}-2}$ of $\underline{\text{Israeli}}^{\text{PRE}}_{\text{GPE}-3}$ $\underline{\text{troops}}^{\text{NOM}}_{\text{PER}-4}$ in the $\underline{\text{West Bank}}^{\text{NAM}}_{\text{LOC}-5}$ said there was a simple goal to the $\underline{\text{helicopter}}^{\text{PRE}}_{\text{VEH}-6}$ assassination on Thursday of a gun-wielding local $\underline{\text{Palestinian}}^{\text{PRE}}_{\text{GPE}-7}$ $\underline{\text{leader}}^{\text{NOM}}_{\text{PER}-8}$. " $\underline{\text{I}}^{\text{PRO}}_{\text{PER}-2}$ hope it will reduce the violence and bring back reason to this $\underline{\text{area}}^{\text{NOM}}_{\text{LOC}-9}$ " , Maj $\underline{\text{Gen}}^{\text{PRE}}_{\text{PER}-2}$ $\underline{\text{Yitzhak Eitan}}^{\text{NAM}}_{\text{PER}-2}$ told $\underline{\text{reporters}}^{\text{NOM}}_{\text{PER}-10}$ at a briefing hours after three $\underline{\text{missiles}}^{\text{NOM}}_{\text{WEA}-11}$ fired from an $\underline{\text{Apache}}^{\text{PRE}}_{\text{VEH}-6}$ $\underline{\text{helicopter}}^{\text{NOM}}_{\text{VEH}-6}$ killed $\underline{\text{Hussein Obaiyat}}^{\text{NAM}}_{\text{PER}-8}$, along with two middle-aged $\underline{\text{women}}^{\text{NOM}}_{\text{PER}-12}$ standing near $\underline{\text{his}}^{\text{PRO}}_{\text{PER}-8}$ $\underline{\text{van}}^{\text{NOM}}_{\text{VEH}-13}$ in $\underline{\text{Beit Sahur}}^{\text{NAM}}_{\text{GPE}-14}$, near $\underline{\text{Bethlehem}}^{\text{NAM}}_{\text{GPE}-15}$. Instead , it has touched off one of the bloodiest and most intense weekends of fighting yet in the six-week-old conflict , with gunfire crackling through the $\underline{\text{West Bank}}^{\text{NAM}}_{\text{LOC}-5}$ and $\underline{\text{Gaza Strip}}^{\text{NAM}}_{\text{LOC}-16}$. Five $\underline{\text{Palestinians}}^{\text{NOM}}_{\text{PER}-17}$ and an $\underline{\text{Israeli}}^{\text{PRE}}_{\text{GPE}-3}$ $\underline{\text{soldier}}^{\text{NOM}}_{\text{PER}-18}$ were shot dead on Friday .

Figure 1.1: An example paragraph extract from a document from our training data with entities identified.

few. With a handful of exceptions (primarily the work stemming from the use of conditional random fields), the majority of these techniques have required the development of specialized algorithms for performing the parameter learning. One goal of this thesis is to provide a generic learning technique that can be applied to a large variety of problems, allowing the researcher to focus effort on other aspects of natural language problems.

1.2 Example Problem: Entity Detection and Tracking

For the purposes of clear exposition, I will use the *entity detection and tracking* (EDT) problem as a running example throughout the thesis. (Additionally, of all the tasks I attack in this thesis, EDT is the most significant.)

The entity detection and tracking problem focuses on discovering the set of entities discussed in a document and identifying the textual span of the document (the *mentions*) that refer to these entities. As part of the detection phase, a system must also identify, for each entity, its corresponding *entity type* (person, place, organization, etc.) and, for each mention of an entity, its *mention type* (name, nominal, pronoun, etc.).

In Figure 1.2, I show one paragraph from the data set I use, wherein entities have been identified, types have been disambiguated and coreference chains have been marked. In this paragraph, I underline every entity mention. Each mention is followed by a superscript that identifies the *mention type* and a subscript that identifies both the *entity type* and *coreference chain* of that mention. For instance, the word "commander" is a nominal reference to a person, identified as entity number 2. At the beginning of the second sentence, the word "I" is a pronominal mention also referring to entity 2 (and hence is the same entity). A few of the coreference chains that appear in this extract are: {JERUSALEM}, {commander, I, Gen, Yitzhak Eitan}, {Israeli, Israeli} and {troops}.

Entity detection and tracking is interesting from three separate angles. From a linguistics perspective, identifying coreference is a challenging problem. An analysis of what sources of knowledge are required to adequately solve this problem would greatly increase our state of knowledge. From a computer science perspective, it is computationally challenging. Even just the coreference task—identifying the entity chains given the mentions—turns out to be FNP-hard[1] under any reasonable model. This can be shown by reduction to graph partitioning (McCallum and Wellner, 2004). Developing efficient algorithms for solving this problem is of utmost importance to building a system that can function in the real-world. Finally, from a machine learning perspective, this task is interesting because it exhibits significantly complex structure. A machine learning technique that could solve EDT directly would need to be able to make much more complex decisions than simple "yes/no" answers.

Like all natural language processing problems, the primary difficulty in the EDT task is ambiguity and the multiple diverse sources of information required to resolve this ambiguity. Consider, for instance, the example paragraph shown in Figure 1.2. Identifying that the "I" in the second sentence is the same person as the "commander" in the first sentence is an extremely challenging inference to make. In fact, it is possible that the two mentions actually refer to two different entities who happen to agree in what they say. Identifying that the "Gen" entity is the same as "Yitzhak Eitan" requires some knowledge of syntax, as does linking this entity with the pronoun "I." On the other hand, identifying that the "Apache" referred to in the second sentence is coreferent with "helicopter" form the first sentence requires external knowledge that an Apache is a type of helicopter. Identifying that "his" in the second sentence is coreferent with "Hussein Obaiyat" and not "Yitzhak Eitan" requires further syntactic knowledge.

From a machine learning perspective, the EDT problem is hard because of the necessity for tying decisions together. That is, the decision at the end of the example in Figure 1.2 that stipulates that "West Bank" is a named location is wholly tied to the decision at the beginning of the example that the same string is also a named location. Learning under the influence of such mutually reinforced decisions is challenging. A significant contribution of this thesis is a technique for dealing with this difficulty.

1.3 The Role of Search

Natural language processing problems like those discussed in Section 1.6—and structured prediction problems more generally—all include a search component. This component is inherantly tied to the fact that structured prediction involves producing something more complex than a single scalar response. To find the best (or approximate best) output, some variety of search is necessary.

In real-world NLP applications, search comes in many flavors. In very rare cases, one can apply dynamic programming-based exact search techniques. This occurs most frequently in sequence labeling problems or in natural language parsing. However, in order to make the problems amenable to dynamic programming (and hence efficient), restrictions must be placed on the models and feature spaces. In particular, the "Markov assumption" must be used in sequence labeling tasks: this states that the features used to predict the label for the word at position i can only refer to the k most recent other labels (for typical $k \in \{0, 1, 2\}$). In the case of parsing, a similar assumption is used: that the grammar is context free. Although these assumptions patently violate what we know about language, they are necessary for maintaining a polynomial time search algorithm.

[1]See Appendix A.3 for a discussion of the computational complexity classes relevant to this thesis.

Unfortunately, being polynomial time is often not sufficient in practice. For instance, *lexicalized* context free parsing is $\mathcal{O}(N^6)$, where N is the length of the sentence (Manning and Schutze, 2000). Even worse, *synchronous* context free parsing, as used in syntactic machine translation, is $\mathcal{O}(N^{12})$, where N is the length of the input sentence (Huang and Chiang, 2005). Even simple sequence labeling is $\mathcal{O}(NK^2)$, where N is the length of the sentence and K is the number of possible labels. When K is very large (on the order of hundreds), such as for phoneme recognition, K^2 is very costly (Pal, Sutton, and McCallum, 2006). In other applications, there simply is no polynomial time solution under even very simplified models; see (Germann et al., 2003) for an example in machine translation.

The effectively intractable (intractable or high-order polynomial) nature of these important problems has led to the use of approximate search algorithms. These include greedy search (Germann et al., 2003), beam search (Och, Zens, and Ney, 2003; Pal, Sutton, and McCallum, 2006), approximate A* search (Klein and Manning, 2003b), lazy pruning, hill-climbing search, and others (Russell and Norvig, 1995). None of these algorithms is guaranteed to find the best possible output. In practice, this is a significant problem. Each requires domain-specific tweaking of search parameters to balance efficiency against search errors. Performing this tweaking well is often incredibly difficult.

1.4 Learning in Search

The canonical way of looking at structured prediction problems is as follows. First, one constructs a *model*. This model effectively tells us: for a given input, what are all the possible outputs. For instance, in machine translation, a phrase-based model tells us that the set of possible translations for a given Arabic input sentence is the set of all English sentences that can be derived through a sequence of phrase translation and reordering steps. In sequence labeling, the model tells us all the possible output sequences for a given input string (typically this is just the set of all sequences over an alphabet of tags of equal length to the input sentence).

Once one has a model, one attaches *features* to that model. The goal of the features is to identify characteristics of input/output pairs that are indicative of whether the output is "good" or not. For translation, these features might look like phrase translation probabilities. For sequence labeling, the features are often lexicaled pairs, such as "assign label 'determiner' to the word 'the'." The features come with corresponding parameters, and the goal of learning is to adjust the parameters so that, for a given input, out of all possible outputs considered by the model, the "correct one" has a high score. The corresponding *search* problem is to *find* the output with the highest score.

The approach advocated in this thesis falls under the heading of *learning in search*. The key premise of this paradigm is that *given* that one will be applying search to find the best output, one should adjust the learning algorithm to account for this. This idea has been previously explored by Boyan and Moore (1996), Collins and Roark (2004) and me (Daumé III and Marcu, 2005c). However, the algorithm described in this thesis takes this idea one step further. Instead of *accounting for* search in the process of learning, I treat the structured prediction problem as being *defined* by a search process. The result is that the role played originally by the model is now played by the specification of a search algorithm, and the learning involved is only to learn how to search.

The specific algorithm I describe, Searn, works on the following basic principle. Each decision made during search is treated as a (large) classification problem. The goal is to learn a classifier that will make each search decision optimally. The primary difficulty is that in order to define "optimally" we must take into account what this same classifier did in the past search steps *and* what it will do in future search steps. I propose a relatively straightforward iterative algorithm for optimizing in this chicken-and-egg situation.

1.5 Contributions

The primary contribution in this thesis is the development of an algorithm called Searn (for "search-learn") for solving structured prediction problems under *any* model, *any* feature functions and *any* loss. Unlike previous approaches to the structured prediction problem (see Section 2.2), Searn makes no assumptions of conditional independence and is computationally efficient in a *superset* of those problems to which competing generic algorithms may be applied.

I formally show that Searn possesses many desirable properties (see Chapter 3). Most importantly, I show that the difference in performance between the model that Searn learns and the best possible model is small (under certain conditions). This result holds independent of the model structure or the feature functions and is a significant improvement over techniques whose performance depends strongly on the locality of features in the output. More generally, I show that any problem that can be solved efficiently by competing techniques can also be solved efficiently by Searn. Finally, I show that Searn is easily extended to *hidden variable* problems, both in the unsupervised and semi-supervised settings, as well as learning under weak feedback (see Chapter 7).

In addition to having attractive theoretical properties, I show that Searn performs very well in a set of diverse real-world problems. These problems include the standard sequence labeling tasks considered by most other structured prediction techniques as well as the more complex *joint* sequence labeling task (see Chapter 4). However, the true test of Searn is in problems with more complex structure. I apply Searn to a complex information extraction problem—entity detection and tracking—and obtain a state-of-the-art model (see Chapter 5). Finally, I apply Searn in the development of a novel model for automatic summarization (see Chapter 6) that easily surpasses the limitations of any other current structured prediction technique.

In addition to the main contributions described above, the development of Searn has led to several other results. The most significant secondary result is that, to my knowledge, Searn is the first algorithm to show a strong connection between structured prediction and reinforcement learning. This connection alone opens up the possibility for many avenues of future research (some of which are discussed in Chapter 7). Additionally, this thesis opens up the possibility to ask new interesting questions about the connection between computational complexity, search and learning (also discussed in Chapter 7). Finally, I will make available many of the applications developed in this thesis to the general public to allow others to benefit from this work.

1.6 An Overview of This Thesis

This thesis is presented in three parts. The first part, comprising the next two chapters, focuses on structured prediction as a machine learning problem. This part concludes with a description of my structured prediction algorithm, SEARN. The second part of the thesis, comprising Chapter 4 though Chapter 6, discusses the application of SEARN to three problems in NLP (one problem per chapter). The third and final part of the thesis concludes and presents preliminary results on extensions to the SEARN algorithm in more complex settings.

The breakdown of this thesis makes it inappropriate to discuss "prior work" in a single chapter. Instead, I have adopted the following strategy. Chapter 2 will discuss background information on machine learning and structured prediction. It will not discuss any prior work on any of the applications I consider. Subsequent chapters will include their own prior work sections at the end. This organization allows easy referencing between my work and that of others. It also enables more discussion of the pros and cons of my approach in comparison to prior work.

The chapters in this thesis are organized as follows:

Part I: Machine Learning

Chapter 2 introduces relevant background from machine learning. The chapter introduces the relevant statistical learning theory necessary to understand the remainder of the thesis as well as the notion of *loss-driven* learning. This chapter also formally defines the notion of a *learning reduction* that I make heavy use of in the development of my own algorithm, SEARN. It concludes with a discussion of prior work on the structured prediction task.

Chapter 3 introduces my algorithm, SEARN, for solving structured prediction problems. This chapter also contains the bulk of the theoretical results pertaining to SEARN and describes the connections between structured prediction and reinforcement learning. Chapter 3 concludes with a comparison of SEARN to prior work in structured prediction.

Part II: Applications

Chapter 4 begins a sequence of three chapters on experimental results with SEARN. This chapter focuses on the simplest problem: sequence labeling. I describe how to apply SEARN to this problem and present results on three data sets: syntactic chunking, named entity recognition in Spanish and handwriting recognition. I then present the results of applying SEARN to a *joint* sequence labeling task: simultaneous part of speech tagging and syntactic chunking.

Chapter 5 describes the application of SEARN to the entity detection and tracking problem introduced in Section 1.2. In this chapter, I discuss both the algorithmic and search issues involved in the EDT task as well as the task of developing useful features for this problem. I report the effects of various knowledge sources on the EDT problem: lexical, syntactic, semantic and knowledge-based, and find that knowledge-based features prove incredibly useful for this problem.

Chapter 6 applies SEARN to a set of summarization models. These models truly stretch the applicability of generic structured prediction techniques and show that it is possible to optimize a structured prediction model against a weaker variety of loss function than I consider in the other experimental setups.

Part III: Future Work

Chapter 7 describes two extensions to SEARN. The first is a methodology for applying SEARN to hidden variable models, such as those commonly used in machine translation. The second is a technique for improving SEARN-learned models on the basis of *weak* user feedback. I present proof-of-concept experimental results in word alignment and summarization. I then conclude the thesis by summarizing the important contributions and looking forward to future research, both theoretical and practical.

Chapter 2

Machine Learning

One goal of this thesis is to develop a learning framework that is able to learn to predict complex, structured outputs with highly interdependent features, as typified by the entity detection and tracking problem. This chapter presents the background material necessary to understand my contributions in this area.

There are three primary sections in this chapter. In Section 2.1, I introduce background information in non-structured statistical learning. This second focuses on three popular algorithms for binary classification: the perceptron, logistic regression and the support vector machine. In Section 2.2, I introduce the current state-of-the-art structured prediction techniques. These techniques can be seen as *extensions* of the previously described binary classification algorithms to the structured prediction domain. Finally, in Section 2.3, I describe the technique of learning *reductions*. Reductions are a technique for transforming a hard learning problem into an easier learning problem and form the theoretical basis of my algorithm for solving structured prediction problems.

2.1 Binary Classification

Supervised learning aims to learn a function f that maps an input $x \in \mathcal{X}$ to an output $y \in \mathcal{Y}$. The standard supervised learning setting typically focuses on binary classification ($\mathcal{Y} = \{-1, +1\}$), multiclass classification ($\mathcal{Y} = \{1, \dots, K\}$ for a small K), or regression ($\mathcal{Y} = \mathbb{R}$). For an example of binary classification, we might want to predict whether or not it will be sunny tomorrow on the basis of past weather data. Such a decision will be made on the basis of a *feature function*, denoted $\Phi : \mathcal{X} \to \mathcal{F}$, where $\mathcal{F}$ is the "feature space." In our example, $\Phi(x)$ might encode information such as temperature, atmospheric pressure and time of year. Typically, $\mathcal{F} = \mathbb{R}^D$, the D-dimension real vector space.

The general *hypothesis class* we consider is that of linear classifiers (i.e., biased hyperplanes)[1]. That is, we parameterize our binary classification function by a weight vector $\boldsymbol{w} \in \mathbb{R}^D$ and a scalar $b \in \mathbb{R}$. The classification function is given in Eq (2.1).

[1] The restriction to linear classifiers may seem overly restrictive (for instance, linear classifiers cannot correctly solve the "XOR problem"). However, by employing *kernels*, one can convert most of the algorithms I describe in this chapter into non-linear classifiers. The use of kernels is a bit outside the scope of this thesis, so I do not discuss them further. See (Burges, 1998; Daumé III, 2004a; Christianini and Shawe-Taylor, 2000) for further discussion.

$$f(x; \boldsymbol{w}, b) = \boldsymbol{w}^\top \Phi(x) + b = \sum_d w_d \Phi(x)_d + b \tag{2.1}$$

The classification decision is according to the sign of f. That is, if $f(x) > 0$ then we decide the class is $+1$ and if $f(x) < 0$ then we decide the class is -1.

Once we have restricted ourselves to the linear hypothesis class, the learning problem becomes that of finding "good" values of $\boldsymbol{w}$ and b. These values are learned on the basis of a finite data sample $\langle x_n, y_n \rangle_{1:N}$ of training examples. Exactly how we define "good" determines the algorithm we choose to use. Nevertheless, all three algorithms we discuss have the same basic flavor for how they define "good." Each involves two components:

1. Fitting the data. The algorithms attempt to find parameters that correctly classify the training data, or at least make few mistakes. Moreover, the algorithms disprefer weight vectors that *over-classify* the negative examples: $yf(x) = -2$ is worse than $yf(x) = -1$ for an incorrectly classified example.

2. Not *over-fitting* the training data. Often by having some very large components in the weight vector, our learned function is able to trivially predict the training data, but does not generalize to new data. By requiring that the weight vector is small (or sparse), we aid generalization ability.

2.1.1 Perceptron

The perceptron algorithm (Rosenblatt, 1958) learns a weight vector $\boldsymbol{w}$ and bias b in an online fashion. That is, it processes the training set one example at a time. At each step, it ensures that the current parameters correctly classify the training example. If so, it proceeds to the next example. If not, it moves the weight vector and bias closer to the current example. The algorithm repeatedly loops over the training data until either no further updates are made or a maximum iteration count has been reached.

It can be shown that, if possible, the perceptron algorithm will eventually converge to a setting of parameter values that correctly classifies the entire data set. Unfortunately, this often leads to poor generalization. Improved generalization ability is available by using *weight averaging.* Weight averaging is accomplished by modifying the standard perceptron algorithm so that the final weights returned are the *average* of all weight vectors encountered during the algorithm. In can be shown that weight averaging leads to a more stable solution with better expected generalization (Freund and Shapire, 1999; Gentile, 2001).

Averaging can be naïvely accomplished by maintaining two sets of parameters: the current parameters and the averaged parameters. At each step of the algorithm (after processing a single example), the current parameters are added to the averaged parameters. Once the algorithm completes, the averaged parameters are divided by the number of steps and returned as the final parameters.

Unfortunately, this naïve algorithm is terribly inefficient. First, we would like to avoid adding the entire weight vector to the averaged vector in each iteration. We would only like to make the addition when an update is made. Moreover, the vectors $\Phi(x)$ are often sparse. This makes the update to the true weight vector efficient, but the sum of the

Algorithm $\textsc{AveragedPerceptron}(x_{1:N}, y_{1:N}, I)$

1: $\boldsymbol{w}_0 \leftarrow \langle 0, \dots, 0 \rangle$, $b_0 \leftarrow 0$
2: $\boldsymbol{w}_a \leftarrow \langle 0, \dots, 0 \rangle$, $b_a \leftarrow 0$
3: $c \leftarrow 1$
4: **for** $i = 1 \dots I$ **do**
5:     **for** $n = 1 \dots N$ **do**
6:         **if** $y_n \big[{\boldsymbol{w}_0}^\top \Phi(x_n) + b_0 \big] \leq 0$ **then**
7:             $\boldsymbol{w}_0 \leftarrow \boldsymbol{w}_0 + y_n \Phi(x_n)$, $b_0 \leftarrow b_0 + y_n$
8:             $\boldsymbol{w}_a \leftarrow \boldsymbol{w}_a + c y_n \Phi(x_n)$, $b_a \leftarrow b_a + c y_n$
9:         **end if**
10:         $c \leftarrow c + 1$
11:     **end for**
12: **end for**
13: **return** $(w_0 - w_a/c, b_0 - b_a/c)$

Figure 2.1: The averaged perceptron learning algorithm.

weights and the averaged weights inefficient. It turns out we can get around both of these problems very straightforwardly.

An efficient implementation of the averaged perceptron training algorithm is shown in Figure 2.1. In step (1), the running weight vector and bias are initialized to zero. In step (2), the averaged weight vector and bias are initialized to zero. In step (3), the averaging count is initialized to 1. The algorithm then runs for I iterations. In each iteration, the algorithm processes each example. Step (6) checks to see if the algorithm currently classifies example $\langle x_n, y_n \rangle$ incorrectly. The example is classified incorrectly exactly when y_n and the current prediction ${\boldsymbol{w}_0}^\top \Phi(x_n) + b_0$ have a different sign: when their product is negative.

If the current example $\langle x_n, y_n \rangle$ is misclassified by the current parameters (w_0, b_0), then in step (7), the algorithm moves w_0 closer to $y_n \Phi(x_n)$ and b_0 closer to y_n. In step (8), the averaged weights are updated in the same way, but where the averaging count c is used as a multiplicative factor. Finally, in step (10), regardless of whether an error was made or not, c is incremented.

After the algorithm has finished, the final parameters are returned. The non-averaged version would simply return w_0 and b_0. To accomplish averaging, the algorithm instead returns $(w_0 - w_a/c)$ and $(b_0 - b_a/c)$. It is straightforward to show that this accomplishes weight averaging as desired.

2.1.2 Logistic Regression

Logistic regression is a second popular binary classification method. It is identical to binary *maximum entropy* classification in practice, though the derivation of the two formulations differs. Logistic regression assumes that the conditional probability of the class y is proportional to $\exp f(x)$. This is given in Eq (2.2).

$$
\begin{aligned}
p\left(y \mid x; \boldsymbol{w}, b\right) &= \frac{1}{Z_{x;\boldsymbol{w},b}} \exp\left[y\big(\boldsymbol{w}^\top \Phi(x) + b\big)\right] \qquad (2.2)\\
&= \frac{1}{1 + \exp\left[-2y\big(\boldsymbol{w}^\top \Phi(x) + b\big)\right]}
\end{aligned}
$$

Like the perceptron, the classification decision is based on the sign of $f(x)$.

To train a logistic regression classifier, one attempts to find parameters $\boldsymbol{w}$ and b that *maximize* the likelihood (probability) of the training data. Thus, logistic regression is a maximum likelihood classifier. This accomplishes our goal of performing well on the training data, but does not explicitly seek small weights. To accomplish the latter, a *prior* is placed over the weights. This is typically taken to be a zero-mean, spherical Gaussian with variance σ^2 (Chen and Rosenfeld, 1999), though alternative priors have been employed (Goodman, 2004). This transforms logistic regression from a maximum likelihood method to a maximum a posteriori method, where the posterior distribution over weights given the training data is given in Eq (2.3).

$$
\begin{aligned}
p\left(\boldsymbol{w}, b \mid \langle x_n, y_n\rangle_{1:N}; \sigma^2\right) &\propto p\left(\boldsymbol{w} \mid \sigma^2\right) \prod_{n=1}^{N} p\left(y_n \mid x_n; \boldsymbol{w}, b\right) \qquad (2.3)\\
&\propto \exp\left[-\frac{1}{\sigma^2}\,||\boldsymbol{w}||^2\right] \prod_{n=1}^{N} \frac{1}{1 + \exp\left[-2y_n\big(\boldsymbol{w}^\top \Phi(x_n) + b\big)\right]}
\end{aligned}
$$

Originally, this maximization problem was solved using iterative scaling methods (Berger, 1997). Unfortunately, these techniques are quite inefficient in practice. Recently, gradient-based techniques such as conjugate gradient (Press et al., 2002) and limited-memory BFGS (Nash and Nocedal, 1991; Averick and Moré, 1994) have enjoyed great success (Minka, 2001; Malouf, 2002; Minka, 2003; Daumé III, 2004b). Both of these techniques rely on the ability to compute the gradient of Eq (2.3) with respect to $\boldsymbol{w}$ and b. This is easier if, instead of maximizing the *posterior*, we instead maximize the *log posterior*. The log posterior is given in Eq (2.4) and its gradient in given in Eq (2.5), where C is independent of $\boldsymbol{w}$ and b.

$$
\log p\left(\boldsymbol{w}, b\right) = -\frac{1}{\sigma^2}\,||\boldsymbol{w}||^2 - \sum_{n=1}^{N} \log\left[1 + \exp\left[-2y_n\big(\boldsymbol{w}^\top \Phi(x_n) + b\big)\right]\right] + C \qquad (2.4)
$$

$$
\frac{\partial}{\partial \boldsymbol{w}} \log p\left(\boldsymbol{w}, b\right) = -\frac{1}{2\sigma^2}\boldsymbol{w} + 2\sum_{n=1}^{N} y_n \Phi(x_n) \left[\frac{1}{1 + \exp\left[-2y_n \boldsymbol{w}^\top \Phi(x_n)\right]}\right] \qquad (2.5)
$$

In the binary classification case, one can explicitly compute the second order information required to directly apply a conjugate gradient method. For multiclass classification, this is not possible, and an approximate Hessian method such as limited memory BFGS must be employed. See (Minka, 2003) for more information about the derivation of these results and (Daumé III, 2004b) for a description of an efficient implementation.

2.1.3 Support Vector Machines

Support vector machines provide an alternative formulation of the learning problem in terms of a formal optimization problem (Boser, Guyon, and Vapnik, 1992). SVMs are based on the *large margin* framework. This framework states that if we have to choose between two settings of parameters, we should choose the one that maximizes the distance between the corresponding hyperplane and the nearest data point on either side. Such large margin solutions are intuitively appealing because they are robust against small changes in the data. Theoretically, it can be shown that maintaining a large margin will lead to good generalization (Vapnik, 1979; Vapnik, 1995). Furthermore, it is straightforward to show that the parameters have a large margin if and only if $||w||$ is small (independent of b).

For a moment we restrict ourselves to the simplified problem of separable training data (with a margin of 1)[2]. That is, there exists setting of the parameters so that we can perfectly classify the training data with a large margin. This leads to the simplest formulation of the SVM, given in Eq (2.6).

$$\begin{aligned} \text{minimize}_{\boldsymbol{w},b} \quad & \frac{1}{2}\,||\boldsymbol{w}||^2 && \qquad (2.6)\\ \text{subject to} \quad & y_n\big[\boldsymbol{w}^\top \Phi(x_n) + b\big] \geq 1 && \forall n \end{aligned}$$

The SVM optimization problem states that we wish to find a weight vector $\boldsymbol{w}$ and bias b with minimum norm. The constraints state that, for each data point $\langle x_n, y_n \rangle$ the given parameters *over-classify* this example. That is, the example would be correctly classified if the product in the constraints were always greater than zero, but here we require the stronger condition that it be greater than one.

In many cases this optimization problem will be infeasible: there will not exist a parameter setting that obeys the constraints. Moreover, even for separable data, we often do not wish to force the algorithm to actually achieve perfect classification performance on the training data (for instance, if there are any errors on the data). This leads to the *soft-margin* formulation of the SVM. The idea in the soft-margin SVM is that we no longer require all examples to be over-classified with a margin of one. However, for every example that does not obey this constraint, we measure how far we would have to "push" that example in order to achieve the desired hard-margin constraint. This measurement is known as the "slack" of the corresponding example. This leads to the formulation shown in Eq (2.7).

$$\text{minimize}_{\boldsymbol{w},b} \quad \frac{1}{2}\,||\boldsymbol{w}||^2 + C\sum_{n=1}^{N} \xi_n \qquad (2.7)$$

[2]The margin is simply the smallest value of $y_n f(x_n)$ across the entire data set.

$$\text{subject to} \quad y_n\big[\boldsymbol{w}^\top \Phi(x_n) + b\big] \geq 1 - \xi_n \qquad \forall n$$
$$\xi_n \geq 0$$

In the soft-margin formulation, our objective function includes two components. The first (small norm) forces the SVM to find a solution that is likely to generalize well. The second (small sum of slack variables ξ) forces the SVM to classify most of the training data correctly. The hyper-parameter $C \geq 0$ controls the trade-off between fitting the training data and finding a small weight vector. As C tends toward infinity, the soft-margin SVM approaches the hard-margin SVM and all the training data must be correctly classified. As C tends toward zero, the SVM cares less and less about correctly classifying the training data and simply seeks a small weight vector.

In the constraints of the soft-margin SVM formulation, we now require that each example be over-classified by $1 - \xi_n$ rather than 1. If parameters can be found that classifies each example with a margin of 1, then the ξ_ns can be made to all be zero. However, for inseparable data, these slack variables account for the training error. While there are as many constraints as data points, it can be shown by the Karush-Kuhn-Tucker conditions (Bertsekas, Nedic, and Daglar, 2003) that at the optimal weight vector, only very few of these are "active." That is, at the optimal values of $\boldsymbol{w}$ and b, $y_n\big[\boldsymbol{w}^\top \Phi(x_n)+b\big]$ is strictly greater than one and are hence inactive for many n. The examples n that are active are called the *support vectors* because those are the only examples that have any affect on the classification decision. In particular, $\boldsymbol{w}$ can be written as a linear combination of the support vectors, ignoring the rest of the training data.

There are many algorithms for solving the SVM problem. The most straightforward is to treat it directly as a quadratic programming problem (Bertsekas, Nedic, and Daglar, 2003) and apply a generic optimization package, such as CPLEX (CPLEX Optimization, 1994). However, the very special form of the optimization problem (namely the sparsity of the constraints) has lead to the development of specialized algorithms, such as sequential minimal optimization (Platt, 1999). More recently, however, it has been recognized that simple gradient-based techniques can lead to highly efficient solutions to the SVM problem (Wen, Edelman, and Gorsich, 2003; Ratliff, Bagnell, and Zinkevich, 2006).

2.1.4 Generalization Bounds

One of the most fundamental theoretical questions about classification problems is the question of generalization: how well will we do on "test data." This question is usually answered in the form "with high probability, the error we observe on unseen test data will be at most the error we incur on the training data plus a regularization term." The regularization term typically makes use of quantities such as the number of training examples, the number of features, and the "size" (or complexity) of the weight vector.

In order to prove statements of this form, one needs to make assumptions about the relationship between the training data and the test data. In particular, we have to assume that the training data is representative of the test data. This is formalized as saying that there is a fixed, but unknown, probability distribution $\mathcal{D}$ and the training data and test data are both sampled from $\mathcal{D}$. This is the *identicality* assumption. The second assumption is to assure us that our training data is representative of the entire distribution $\mathcal{D}$.

We assume that the training data is drawn *independently* from $\mathcal{D}$. Formally, *if* we knew $\mathcal{D}$, then, conditional on $\mathcal{D}$, the points in the training data would be independent. When the training data obeys these properties, we say that it is *independently and identically distributed* from $\mathcal{D}$ (or, "i.i.d." from $\mathcal{D}$). The i.i.d. assumption underlies the majority of the theoretical work on generalization bounds.

For concreteness, consider the support vector machine. Denote by $\mathcal{L}^{\text{emp}}(D, f)$ the average empirical loss (Eq (2.8)) over the training data D for the classifier f. Denote by $\mathcal{L}^{\text{exp}}(\mathcal{D}, f)$ the expected loss (Eq (2.9)) of the classifier f over data drawn i.i.d. from a distribution $\mathcal{D}$.

$$\mathcal{L}^{\text{emp}}(D, f) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}(y_n \neq f(x_n)) \tag{2.8}$$

$$\mathcal{L}^{\text{exp}}(\mathcal{D}, f) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \big[\mathbf{1}(y \neq f(x)) \big] \tag{2.9}$$

A (comparatively) simple generalization bound for the SVM takes the form of Theorem 2.1. Note that, depending on stronger assumptions, stronger bounds are available (Bartlett and Shawe-Taylor, 1999; Zhang, 2002; McAllester, 2003; McAllester, 2004; Langford, 2005). This one was chosen because it is comparatively easier to state.

Theorem 2.1 (SVM Generalization; (Langford and Shawe-Taylor, 2002)). *For all averaging classifiers c with normalized weights $\boldsymbol{w}$, for all error rates $\epsilon > 0$ and all margins $\gamma > 0$, Eq (2.10) holds with probability greater than $1 - \delta$ over training sets S of size m drawn i.i.d. from a distribution $\mathcal{D}$.*

$$KL\left(\hat{e}_\gamma(c) + \epsilon \;||\; e(c) - \epsilon\right) \leq \frac{1}{m} \left[2 \ln \frac{m+1}{\delta} - \ln \bar{F} \left(\frac{\bar{F}^{-1}(e)}{\gamma} \right) \right] \tag{2.10}$$

where $\bar{F}(x)$ is the tail probability of a zero-mean, unit variance Gaussian, $e_\gamma(c)$ is the expected margin-error rate for the classifier c with respect to a margin γ and $e(c)$ is the error of the classifier c (i.e., $e(c) = e_0(c)$).

This theorem works as follows. We are comparing the *empirical* error (on the lhs of the KL) of the classifier to the *true* error (on the rhs of the KL), modulo a fixed error rate ϵ. We desire the divergence between these error distributions to be small because this would imply that our estimated empirical error is close to what we expect to see on test data. The theorem states that this divergence is bounded by a term that scales roughly as $1/m$, where m is the number of training points, and roughly as $\ln \bar{F}(1/\gamma)$, where γ is the margin. In particular, as m increases, the bound becomes tighter. Also, as γ increases, the bound becomes tighter. Thus, to achieve good generalization, one wants a lot of data and a large margin.

The important things to note about Theorem 2.1 are the following. First, it assumes that the training data is i.i.d. (this is the standard assumption). Second, the bound improves as the weight vector shrinks (i.e., as the margin increases). Third, the bound improves as the number of training examples grows. This provides some theoretical justification for the SVM formulation.

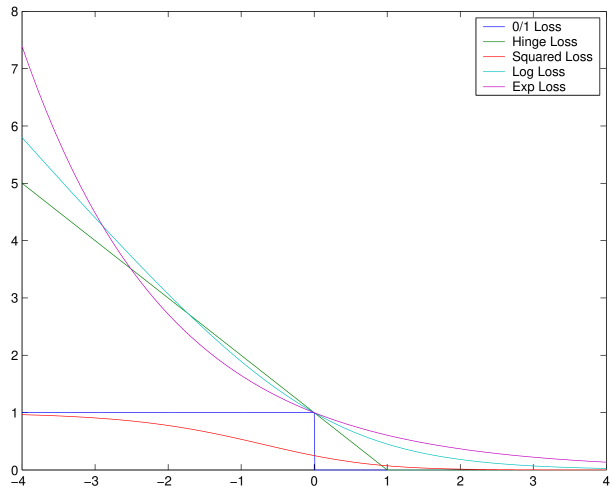


Figure 2.2: Plot of several convex approximations to the zero-one loss function.

2.1.5 Summary of Learners

The learners described in this section—the Perceptron, maximum entropy models and support vector machines—are effective solutions to the binary classification problem. In general, support vector machines tend to outperform the perceptron and maximum entropy models empirically. However, they do so at a non-trivial computational cost. The perceptron is highly efficient and often reaches a reasonable solution even after only one pass through the training data. Maximum entropy models, while slightly slower, still operate at a speed of roughly $\mathcal{O}(N)$. SVMs, contrastively, often scale at least as $\mathcal{O}(N^2)$ if not $\mathcal{O}(N^3)$. For large data sets this can render them intractable.

Despite these differences, these three models are not so dissimilar. In fact, when optimized using sub-gradient methods (Zinkevich, 2003; Ratliff, Bagnell, and Zinkevich, 2006), SVMs are *exactly* the result of adding regularization and margins to the perceptron (Collobert and Bengio, 2004). In particular, the "update" term in the perceptron happens not only when a mistake is made, but when an example is not over-classified. Furthermore, weights are shrunk at every iteration toward zero according to the regularization parameter C. On the other hand, the perceptron can also be seen as a stochastic approximation to the gradient for maximum entropy models when the log normalizing constant is approximated with a max rather than a sum (Collins, 2002).

These similarities can be seen more clearly by examining the exact loss function optimized by the three learners. In Figure 2.2, I have plotted these (and other) loss functions. In this graph, I plot the prediction $yf(x)$ along the x-axis and the loss along the y-axis. The most basic loss, 0/1 loss, is the desired loss. It is a step-function that is zero when $yf(x) > 0$ and one otherwise. This is the loss function that the perceptron optimizes. In general, however, it is a difficult function to optimize: neither is it convex nor differentiable. The other functions we consider are convex upper bounds on the 0/1 loss. For instance, the log loss, which is optimized by maximum entropy models, touches the 0/1 loss at the corner and slowly falls to asymptote at the axis as $yf(x) \to \infty$.

The hinge loss (also called the margin loss), which is optimized by the SVM, is a ramp function that has slope -1 when $yf(x) < 1$ and is zero otherwise. Two other loss functions—squared loss and exponential loss—are also shown; these are used in other learning algorithms such as neural networks (Bishop, 1995) and boosting (Schapire, 2003; Lebanon and Lafferty, 2002). Each of these loss functions has different advantages and disadvantages; these are too deep and off-topic to attempt to discuss in the context of this thesis. The interested reader is directed to (Bartlett, Jordan, and McAuliffe, 2005) for more in-depth discussions.

2.2 Structured Prediction

The vast majority of prediction algorithms, such as those described in the previous section, are built to solve prediction problems whose outputs are "simple." Here, "simple" is intended to include binary classification, multiclass classification and regression. (I note in passing that some of the aforementioned algorithms are more easily adapted to multiclass classification and/or regression than others.) In contrast, the problems I am interested in solving are "complex." The family of generic techniques for solving such "complex" problems are generally known as *structured prediction algorithms* or *structured learning algorithms*. To date, there are essentially four state-of-the-art structured prediction algorithms (with minor variations), each of which I briefly describe in this section. However, before describing these algorithms in detail, it is worthwhile to attempt to formalize what is meant by "simple," "complex" and "structure." It turns out that defining these concepts is remarkably difficult.

2.2.1 Defining Structured Prediction

Structured prediction is a very slippery concept. In fact, of all the primary prior work that proposes solutions to the structured prediction problem, none explicitly defines the problem (McCallum, Freitag, and Pereira, 2000; Lafferty, McCallum, and Pereira, 2001; Punyakanok and Roth, 2001; Collins, 2002; Taskar, Guestrin, and Koller, 2003; McAllester, Collins, and Pereira, 2004; Tsochantaridis et al., 2005). In all cases, the problem is explained and motivated purely by means of examples. These examples include the following problems:

- Sequence labeling: given an input sequence, produce a label sequence of equal length. Each label is drawn from a small finite set. This problem is typified in NLP by part-of-speech tagging.

- Parsing: given an input sequence, build a tree whose yield (leaves) are the elements in the sequence and whose structure obeys some grammar. This problem is typified in NLP by syntactic parsing.

- Collective classification: given a graph defined by a set of vertices and edges, produce a labeling of the vertices. This problem is typified by relation learning problems, such as labeling web pages given link information.

- Bipartite matching: given a bipartite graph, find the best possible matching. This problem is typified by (a simplified version of) word alignment in NLP and protein structure prediction in computational biology.

There are many other problems in NLP that do not receive as much attention from the machine learning community, but seem to also fall under the heading of structured prediction. These include entity detection and tracking, automatic document summarization, machine translation and question answering (among others). Generalizing over these examples leads us to a partial definition of structured prediction, which I call Condition 1, below.

Condition 1. *In a structured prediction problem, output elements $y \in \mathcal{Y}$ decompose into variable length vectors over a finite set. That is, there is a finite $M \in \mathbb{N}$ such that each $y \in \mathcal{Y}$ can be identified with at least one vector $v_y \in M^{T_y}$, where T_y is the length of the vector.*

This condition is likely to be deemed acceptable by most researchers who are active in the structured prediction community. However, there is a question as to whether it is a sufficient condition. In particular, it includes many problems that would *not* really be considered structured prediction (binary classification, multitask learning (Caruana, 1997), etc.). This leads to a second condition that hinges on the form of the loss function. It is natural to desire that the loss function does not decompose over the vector representations. After all, if it does decompose over the representation, then one can simply solve the problem by predicting each vector component independently. However, it is always possible to construct *some* vector encoding over which the loss function decomposes[3] This means that we must therefore make this conditions stronger, and require that there is no polynomially sized encoding of the vector over which the loss function decomposes.

Condition 2. *In a structured prediction problem, the loss function does* not *decompose over the vectors v_y for $y \in \mathcal{Y}$. In particular, $l(x, y, \hat{y})$ is* not *invariant under identical permutations of y and $\hat{y}$. Formally, we must make this stronger: there is* no *vector mapping $y \mapsto v_y$ such that the loss function decomposes, for which $|v_y|$ is polynomial in $|y|$.*

Condition 2 successfully excludes problems like binary classification and multitask learning from consideration as structured prediction problems. Importantly, it excludes standard classification problems and multitask learning. Interestingly, it also excludes problems such as sequence labeling under Hamming loss (discussed further in Chapter 4). Hamming loss (per-node loss) on sequence labeling problems *is* invariant over permutations. This condition also excludes collective classification under zero/one loss on the nodes. In fact, it excludes virtually any problem that one could reasonably hope to solve by using a collection of independent classifiers (Punyakanok and Roth, 2001).

[3]To do so, we encode the true vector in a very long vector by specifying the exact location of each label using products of prime numbers. Specifically, for each label k, one considers the positions $i_1, \dots, i_Z$ in which k appears in the vector. The encoded vector will contain $p_1^{i_1} p_2^{i_2} \cdots p_Z^{i_Z}$ copies of element k, where $p_1, \dots$ is an enumeration of the primes. Given this encoding it is always possible to reconstruct the original vector, yet the loss function will decompose.

The important aspect of Condition 2 is that it hinges on the notion of the loss function rather than the features. For instance, one can argue that even when sequence labeling is performed under Hamming loss, there is still important structural information. That is, we "know" that by including structural *features* (such as Markov features), we can solve most sequence labeling tasks better.[4] The difference between these two perspectives is that under Condition 2 the *loss* dictates the structure, while otherwise the *features* dictate the structure. Since when the world hands us a problem to solve, it hands us the loss but not the features (the features are part of the *solution*), it is most appropriate to define the structured prediction *problem* only in terms of the loss.

Current generic structured prediction algorithms are not built to solve problems under which Condition 2 holds. In order to facilitate discussion, I will refer to problems for which both conditions hold as "structured prediction problem" and those for which only Condition 1 holds as "decomposable structured prediction problems." I note in passing that this terminology is nonstandard.

2.2.2 Feature Spaces for Structured Prediction

Structured prediction algorithms make use of an extended notion of feature function. For structured prediction, the feature function takes as input *both* the original input $x \in \mathcal{X}$ and a hypothesized output $y \in \mathcal{Y}$. The value $\Phi(x, y)$ will again be a vector in Euclidean space, but which now depends on the output. In particular, in part of speech tagging, an element in $\Phi(x, y)$ might be the number of times the word "the" appears and is labeled as a determiner and the next word is labeled as a noun.

All structured prediction algorithms described in this Chapter are only applicable when Φ admits efficient search. In particular, after learning a weight vector $\boldsymbol{w}$, one will need to find the best output for a given input. This is the "argmax problem" defined in Eq (2.11).

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \boldsymbol{w}^\top \Phi(x, y) \tag{2.11}$$

This problem will not be tractable in the general case. However, for very specific $\mathcal{Y}$ and very specific Φ, one can employ dynamic programming algorithms or integer programming algorithms to find efficient solutions. In particular, if Φ decomposes over the vector representation of $\mathcal{Y}$ such that no feature depends on elements of y that are more than k positions away, then the Viterbi algorithm can be used to solve the argmax problem in time $\mathcal{O}(M^k)$ (where M is the number of possible labels, formally from Condition 1). This case includes standard sequence labeling problems under the Markov assumption as well as parsing problems under the context-free assumption.

2.2.3 Structured Perceptron

The structured perceptron is an extension of the standard perceptron (Section 2.1.1) to structured prediction (Collins, 2002). Importantly, it is only applicable to the problem

[4] This is actually not necessarily the case; see Section 4.2 for an extended discussion.

Algorithm AVERAGEDSTRUCTUREDPERCEPTRON($x_{1:N}, y_{1:N}, I$)

1: $\boldsymbol{w}_0 \leftarrow \langle 0, \ldots, 0 \rangle$
2: $\boldsymbol{w}_a \leftarrow \langle 0, \ldots, 0 \rangle$
3: $c \leftarrow 1$
4: **for** $i = 1 \ldots I$ **do**
5:   **for** $n = 1 \ldots N$ **do**
6:     $\hat{y}_n \leftarrow \arg\max_{y \in \mathcal{Y}} \boldsymbol{w}_0{}^\top \Phi(x_n, y_n)$
7:     **if** $y_n \neq \hat{y}_n$ **then**
8:       $\boldsymbol{w}_0 \leftarrow \boldsymbol{w}_0 + \Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n)$
9:       $\boldsymbol{w}_a \leftarrow \boldsymbol{w}_a + c\Phi(x_n, y_n) - c\Phi(x_n, \hat{y}_n)$
10:     **end if**
11:     $c \leftarrow c + 1$
12:   **end for**
13: **end for**
14: **return** $\boldsymbol{w}_0 - \boldsymbol{w}_a / c$

Figure 2.3: The averaged structured perceptron learning algorithm.

of 0/1 loss over $\mathcal{Y}$: that is, $l(x, y, \hat{y}) = \mathbf{1}(y \neq \hat{y})$. As such, it only solves decomposable structured prediction problems (0/1 loss is trivially invariant under permutations). Like all the algorithms we consider, the structured perceptron will be parameterized by a weight vector $\boldsymbol{w}$. The structured perceptron makes one significant assumption: that Eq (2.11) can be solved efficiently.

Based on the argmax assumption, the structured perceptron constructs the perceptron in nearly an identical manner as for the binary case. While looping through the training data, whenever the predicted $\hat{y}_n$ for x_n differs from y_n, we update the weights according to Eq (2.12).

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n) \tag{2.12}$$

This weight update serves to bring the vector closer to the true output and further from the incorrect output. As in the standard perceptron, this often leads to a learned model that generalizes poorly. As before, one solution to this problem is weight averaging. This behaves identically to the averaged binary perceptron and the full training algorithm is depicted in Figure 2.3.

The behavior of the structured perceptron and the standard perceptron are virtually identically. The major changes are as follow. First, there is no bias b. For structured problems, a bias is irrelevant: it will increase the score of all hypothetical outputs by the same amount. The next major difference is in step (6): the best scoring output $\hat{y}_n$ for the input x_n is computed using the $\arg\max$. After checking for an error, the weights are updated, according to Eq (2.12), in steps (8) and (9).

2.2.4 Incremental Perceptron

The *incremental perceptron* (Collins and Roark, 2004) is a variant on the structured perceptron that deals with the issue that the arg max in step 6 may not be analytically available. The idea of the incremental perceptron (which I build on significantly in Chapter 3) is to replace the arg max with a beam search algorithm. Thus, step 6 becomes "$\hat{y}_n \leftarrow \text{BeamSearch}(x_n, \boldsymbol{w}_0)$". The key observation is that it is often possible to detect in the process of executing search whether it is possible for the resulting output to ever be correct. For instance, in sequence labeling, as soon as the beam search algorithm has made an error, we can detect it without completing the search (for standard loss function and search algorithms). The incremental perceptron *aborts* the search algorithm as soon as it has detected that an error has been made. Empirical results in the parsing domain have shown that this simple modification leads to much faster convergence and superior results.

2.2.5 Maximum Entropy Markov Models

The maximum entropy Markov model (MEMM) framework, pioneered by McCallum, Freitag, and Pereira (2000) is a straightforward application of maximum entropy models (aka logistic regression models, see Section 2.1.2) to sequence labeling problems. For those familiar with the hidden Markov model framework, MEMMs can be seen as HMMs where the conditional "observation given state" probabilities are replaced with direct "state given observation" probabilities (this leads to the ability to include large numbers of overlapping, non-independent features). In particular, a first-order MEMM places the conditional distribution shown in Eq (2.13) on the nth label, y_n, given the full input x, the previous label, y_{n-1}, a feature function Φ and a weight vector $\boldsymbol{w}$.

$$
\begin{aligned}
p\left(y_n \mid x, y_{n-1}; \boldsymbol{w}\right) &= \frac{1}{Z_{x,y_{n-1};\boldsymbol{w}}} \exp\left[\boldsymbol{w}^\top \Phi(x, y_n, y_{n-1})\right] \\
Z_{x,y_{n-1};\boldsymbol{w}} &= \sum_{y' \in \mathcal{Y}^n} \exp\left[\boldsymbol{w}^\top \Phi(x, y', y_{n-1})\right]
\end{aligned}
\tag{2.13}
$$

The MEMM is trained by tracing along the true output sequences for the training data and using the *true* y_{n-1} to generate training examples. This process simply produces multiclass classification examples, equal in number to the number of labels in all of the training data. Based on this data, the weight vector $\boldsymbol{w}$ is learned exactly as in standard maximum entropy models.

At prediction time, one applies the Viterbi algorithm, as in the case of the structured perceptron, to solve the "arg max" problem. Importantly, since the true values for y_{n-1} are not known, one uses the *predicted* values of y_{n-1} for making the prediction about the nth value (albeit, in the context of Viterbi search). As I will discuss in depth in Section 3.4.5, this fact can lead to severely suboptimal results.

2.2.6 Conditional Random Fields

While successful in many practical examples, maximum entropy Markov models suffer from two severe problems: the "label-bias problem" (both Lafferty, McCallum, and Pereira (2001) and Bottou (1991) discuss the label-bias problem in depth) and a limitation to sequence labeling. Conditional random fields are an alternative extension of logistic regression (maximum entropy models) to structured outputs (Lafferty, McCallum, and Pereira, 2001). Similar to the structured perceptron, a conditional random field does not employ a loss function. It optimizes a log-loss approximation to the 0/1 loss over the entire output. In this sense, it is also a solution only to a decomposable structured prediction problems.

The actual formulation of conditional random fields is identical to that for multi-class maximum entropy models. The CRF assumes a feature function $\Phi(x, y)$ that maps input/output pairs to vectors in Euclidean space, and uses a Gibbs distribution parameterized by $\boldsymbol{w}$ to model the probability, Eq (2.14).

$$p\left(y \mid x; \boldsymbol{w}\right) = \frac{1}{Z_{x;\boldsymbol{w}}} \exp\left[\boldsymbol{w}^\top \Phi(x, y)\right] \tag{2.14}$$

$$Z_{x;\boldsymbol{w}} = \sum_{y' \in \mathcal{Y}} \exp\left[\boldsymbol{w}^\top \Phi(x, y')\right] \tag{2.15}$$

Here, $Z_{x,\boldsymbol{w}}$ (known as the "partition function") is the sum of responses of all *incorrect* outputs. Typically, this set will be too large to sum over explicitly. However, if Φ is chosen properly and if $\mathcal{Y}$ is a simple linear-chain structure, this sum can be computed using dynamic programming techniques (Lafferty, McCallum, and Pereira, 2001; Sha and Pereira, 2002). In particular, Φ must be chosen to obey the Markov property: for a Markov length of l, no feature can depend on elements of y that are more than l positions apart. The algorithm associated with the sum is nearly identical to the forward-backward algorithm for hidden Markov models (Baum and Petrie, 1966) and scales as $\mathcal{O}(NK^l)$, where N is the length of the sequence, K is the number of labels and l is the "Markov order" used by Φ.

Just as in maximum entropy models, the weights $\boldsymbol{w}$ are regularized by a Gaussian prior and the log posterior distribution over weights is as in Eq (2.16).

$$\log p\left(\boldsymbol{w} \mid D; \sigma^2\right) = -\frac{1}{\sigma^2} ||\boldsymbol{w}||^2 + \sum_{n=1}^{N} \left[\boldsymbol{w}^\top \Phi(x_n, y_n) - \log \sum_{y' \in \mathcal{Y}} \exp\left[\boldsymbol{w}^\top \Phi(x_n, y')\right]\right] \tag{2.16}$$

Finding optimal weights can be solved either using iterative scaling methods (Lafferty, McCallum, and Pereira, 2001) or more complex optimization strategies such as BFGS (Sha and Pereira, 2002; Daumé III, 2004b) or stochastic meta-descent (Schraudolph and Graepel, 2003; Vishwanathan et al., 2006). In practice, the latter two are much more efficient. In practice, in order for full CRF training to be practical, we must be able to efficiently compute both the arg max from Eq (2.11) and the log normalization constant from Eq (2.17).

$$\log Z_{x;\boldsymbol{w}} = \log \sum_{y' \in \mathcal{Y}} \exp \left[\boldsymbol{w}^\top \Phi(x, y') \right] \tag{2.17}$$

So long as we can compute these two quantities, CRFs are a reasonable choice for solving the decomposable structured prediction problem under the log-loss approximation to 0/1 loss over $\mathcal{Y}$. See (Sutton and McCallum, 2006) and (Wallach, 2004) for in-depth introductions to conditional random fields.

2.2.7 Maximum Margin Markov Networks

The Maximum Margin Markov Network (M^{3}N) formalism considers the structured prediction problem as a quadratic programming problem (Taskar, Guestrin, and Koller, 2003; Taskar et al., 2005), following the formalism for the support vector machine for binary classification. Recall from Section 2.1.3 that the SVM formulation sought a weight vector with small norm (for good generalization) and which achieved a margin of at least one on all training examples (modulo the slack variables). The M^{3}N formalism extends this to structured outputs under a given loss function l by requiring that the *difference* in score between the true output y and any incorrect output $\hat{y}$ is at least the loss $l(x, y, \hat{y})$ (modulo slack variables). That is: the M^{3}N framework scales the *margin* to be proportional to the loss. This is given formally in Eq (2.18).

$$\begin{aligned}
\text{minimize}_{\boldsymbol{w}} \quad & \frac{1}{2} ||\boldsymbol{w}||^2 + C \sum_{n=1}^{N} \sum_{\hat{y}} \xi_{n,\hat{y}} && (2.18) \\
\text{subject to} \quad & \boldsymbol{w}^\top \Phi(x_n, y_n) - \boldsymbol{w}^\top \Phi(x_n, \hat{y}) \geq l(x_n, y_n, \hat{y}) - \xi_{n,\hat{y}} && \forall n, \forall \hat{y} \in \mathcal{Y} \\
& \xi_{n,\hat{y}} \geq 0 && \forall n, \forall y' \in \mathcal{Y}
\end{aligned}$$

One immediate observation about the M^{3}N formulation is that there are too many constraints. That is, the first set of constraints is instantiated for every training instance n and for every incorrect output $\hat{y}$. Fortunately, under restrictions on $\mathcal{Y}$ and Φ, it is possible to replace this exponential number of constraints with a polynomial number. In particular, for the special case of sequence labeling under Hamming loss (a decomposable structured prediction problem), one needs only one constraint per element in an example.

In the original development of the M^{3}N formalism (Taskar, Guestrin, and Koller, 2003), this optimization problem was solved using an active set formulation similar to the SMO algorithm (Platt, 1999). Subsequently, more efficient optimization techniques have been proposed, including ones based on the exponentiated gradient method (Bartlett et al., 2004), the dual extra-gradient method (Taskar et al., 2005) and the sub-gradient method (Bagnell, Ratliff, and Zinkevich, 2006). Of these, the last two appear to be the most efficient. In order to employ these methods in practice, one must be able to compute both the arg max from Eq (2.11) as well as a so-called "loss-augmented search" problem given in Eq (2.19).

$$S(x, y) = \arg\max_{\hat{y} \in \mathcal{Y}} \boldsymbol{w}^\top \Phi(x, \hat{y}) + l(x, y, \hat{y}) \tag{2.19}$$

In order for this to be efficiently computable, the loss function is forced to decompose over the structure. This implies that M^3Ns are only (efficiently) applicable to decomposable structured prediction problems. Nevertheless, they are applicable to a strictly wider set of problems than CRFs for two reasons. First, M^3Ns do not have a requirement that the log normalization constant (Eq (2.17)) be efficiently computable. This alone allows optimization in M^3Ns for problems that would be F#P-complete for CRFs (Taskar et al., 2005). Second, M^3Ns can be applied to loss functions other than 0/1 loss over the entire sequence. However, in practice, they are essentially only applicable to a hinge-loss approximation to Hamming loss over $\mathcal{Y}$.

2.2.8 SVMs for Interdependent and Structured Outputs

The Support Vector Machines for Interdependent and Structured Outputs (SVM$^{\text{struct}}$) formalism (Tsochantaridis et al., 2005) is strikingly similar to the M^3N formalism. The difference lies in the fact that the M^3N framework scales the *margin* by the loss, while the SVM$^{\text{struct}}$ formalism scales the *slack variables* by the loss. The quadratic programming problem for the SVM$^{\text{struct}}$ is given as:

$$
\begin{aligned}
\text{minimize}_{\boldsymbol{w}} \quad & \frac{1}{2}\,||\boldsymbol{w}||^2 + C\sum_n \sum_{\hat{y}} \xi_{n,\hat{y}} && \text{(2.20)} \\
\text{subject to} \quad & \boldsymbol{w}^\top \Phi(x_n, y_n) - \boldsymbol{w}^\top \Phi(x_n, y') \geq 1 - \frac{\xi_{n,y'}}{l(x_n, y_n, y')} && \forall n, \forall y' \in \mathcal{Y} \\
& \xi_{n,y'} \geq 0 && \forall n, \forall y' \in \mathcal{Y}
\end{aligned}
$$

The objective function is the same in both cases; the only difference is found in the first constraint. Dividing the slack variable by the corresponding loss is akin to multiplying the slack variables in the objective function by the loss (in the division, we assume $0/0 = 0$). Though, to date, the SVM$^{\text{struct}}$ framework has generated less interest than the M^3N framework, the formalism seems more appropriate. It is much more intuitive to scale the training error (slack variables) by the loss, rather than to scale the margin by the loss. This advantage is also claimed by the original creators of the SVM$^{\text{struct}}$ framework, in which they suggest that their formalism is superior to the M^3N formalism because the latter will cause the system to work very hard to separate very lossful hypotheses, even if they are not at all confusable for the truth.

In addition to the difference in loss-scaling, the optimization techniques employed by the two techniques differ significantly. In particular, the decomposition of the loss function that enabled us to remove the exponentially many constraints does not work in the SVM$^{\text{struct}}$ framework. Instead, Tsochantaridis et al. (2005) advocate an iterative optimization procedure, in which constraints are added in an "as needed" basis. It can be shown that this will converge to a solution within ϵ of the optimal in a polynomial number of steps.

The primary disadvantage to the SVM$^{\text{struct}}$ framework is that it is often difficult to optimize. However, unlike the other three frameworks described thus far, the SVM$^{\text{struct}}$ does *not* assume that the loss function decomposes over the structure. However, in exchange for this generality, the loss-augmented search problem for them SVM$^{\text{struct}}$ framework becomes more difficult. In particular, while the M^{3}N loss-augmented search (Eq (2.19)) assumes decomposition in order to remain tractable, the loss-augmented search (Eq (2.21)) for the SVM$^{\text{struct}}$ framework is often never tractable.

$$S(x, y) = \arg\max_{\hat{y} \in \mathcal{Y}} \left[\boldsymbol{w}^\top \Phi(x, \hat{y}) \right] l(x, y, \hat{y}) \tag{2.21}$$

The difference between the two requirements is that in the M^{3}N case, the loss appears as an additive term, while in the SVM$^{\text{struct}}$ case, the loss appears as a multiplicative term. In practice, for many problems, this renders the search problem intractable.

2.2.9 Reranking

Reranking is an increasingly popular technique for solving complex natural language processing problems. The motivation behind reranking is the following. We have access to a method for solving a problem, but it is difficult or impossible to modify this method to include features we want or to optimize the loss function we want. Assuming that this method can produce a "n-best" list of outputs (instead of just outputting what it thinks is the *single* best output, it produces many best outputs), we can attempt to build a *second* model for picking an output from this n-best list. Since we are only ever considering a constant-sized list, we can incorporate features that would otherwise render the argmax problem intractable. Moreover, we can often optimize a reranker to a loss function closer to the one we care about (in fact, we can do so using techniques described in Section 2.3). Based on these advantages, reranking has been applied in a variety of NLP problems including parsing (Collins, 2000; Charniak and Johnson, 2005), machine translation (Och, 2003; Shen, Sarkar, and Och, 2004), question answering (Ravichandran, Hovy, and Och, 2003), semantic role labeling (Toutanova, Haghighi, and Manning, 2005), and other tasks. In fact, according to the ACL anthology[5], in 2005 there were 33 papers that include the term "reranking," compared to ten in 2003 and virtually none before 2000.

Reranking is an attractive technique because it enables one to quickly experiment with new features and new loss functions. There are, however, several drawbacks to the approach. Some of these are enumerated below:

1. Close ties to original model. In order to rerank, one must have a model whose output can be reranked. The best the reranking model can do is limited by the original model: if it cannot find the best output in an n-best list, then neither will the reranker. This is especially concerning for problems with enormous $\mathcal{Y}$, such as machine translation.

2. Segmentation of training data. One should typically not train a reranker over data that the original model was trained on. This means that one must set aside a held-out data set for training the reranker, leading to less data on which one can train the original model.

[5] `http://acl.ldc.upenn.edu`

	Loss			Features			Efficient	Easy to Implement
	0/1	**Hamming**	**Any**	**argmax and sum**	**argmax only**	**Neither**		
Structured Perceptron	√				√		√	√
Conditional Random Field	√			√				–
Max-margin Markov Network	√	√			√			
SVM for Structured Outputs	√	√			√			
Reranking	√	√	√			√	–	–

Table 2.1: Summary of structured prediction algorithms.

3. Inefficiency. At runtime, one must run two separate systems. Moreover, producing n-best lists is often significantly more complex than producing a single output (for example, in parsing (Huang and Chiang, 2005)).

4. Multiple approximations. It is, in general, advisable to avoid multiple approximations to a single learning problem. Reranking, by definition, solves what should be one problem in two separate steps.

Despite these drawbacks, reranking is a very powerful technique for exploring novel features.

2.2.10 Summary of Learners

All of the structured prediction algorithms I have described share a common property: they are extensions of standard binary classification techniques to (decomposable) structured prediction problems. Each also requires that the arg max problem (Eq (2.11)) be efficiently solvable. Each has various advantages and disadvantages, summarized below.

Structured perceptron. *Advantages:* Efficient, minimal requirements on $\mathcal{Y}$ and Φ, easy to implement. *Disadvantages:* only optimizes 0/1 loss over $\mathcal{Y}$, somewhat poor generalization.

Conditional random fields. *Advantages:* Provides probabilistic outputs, strong connections to graphical models (Pearl, 2000; Smyth, Heckerman, and Jordan, 2001), good generalization. *Disadvantages:* only optimizes log-0/1-loss over $\mathcal{Y}$, slow, partition function (Eq (2.17)) is often intractable.

Max-margin Markov Nets. *Advantages:* Can optimize both 0/1 loss over $\mathcal{Y}$ and hinge-Hamming loss, implements large-margin principle, can be tractable when CRFs are not. *Disadvantages:* very slow, limited to Hamming loss.

SVMs for Structured Outputs. *Advantages:* more loss functions applicable, implements large-margin principle, produces sparse solutions. *Disadvantages:* slow, often-intractable loss-augmented search procedure (Eq (2.21).

The important aspects of each technique are summarized in Table 2.1. This table evaluates each technique on four dimensions. First, and perhaps most importantly, is the type of loss function the algorithm can handle. This is broken down into 0/1 loss, Hamming loss and arbitrary (non-decomposable) loss. Next, the techniques are distinguished by their ability to handle complex features. In particular, all four structured prediction algorithms require that one be able to solve the argmax problem; the CRF requires that the corresponding sum also be tractable. Lastly, the algorithms are compared based on whether they are efficient and easy to implement.

As we can see from this table, none of the structured prediction techniques can handle arbitrary losses, and all require that the argmax be efficiently computable. The CRF additional requires that the sum be efficiently computable. (Though it is not shown on the table, both the M^{3}N and the SVM$^{\text{struct}}$ also require that a loss-augmented argmax be efficiently solvable.) Of the algorithms, only the structured perceptron is efficient (the others require expensive belief propagation/forward-backward computations) and easy to implement.

Also shown on this table, though not explicitly a structured prediction technique, is a row for a reranking algorithm. Reranking is popular precisely because it does enable one to (approximately) handle any loss function and use arbitrary features. However, as discussed in Section 2.2.9, there are several disadvantages to the reranking approach for solving general problems.

The four models described in this section do not form an exhaustive list of all approaches to the structured prediction problem (nor even the sequence labeling problem), though they do form a largely representative list; see also (Punyakanok and Roth, 2001; Weston et al., 2002; McAllester, Collins, and Pereira, 2004; Altun, Hofmann, and Smola, 2004; McDonald, Crammer, and Pereira, 2004) for a variety of other approaches.

2.3 Learning Reductions

Binary classification under 0/1 loss (Section 2.1) is an attractive area of study for many reasons, including simplicity and generality. However, there are many prediction problems that are not 0/1 loss binary problems. For instance, the structured prediction problems discussed in Section 2.2 are not binary classification problems. The techniques described in that section were *extensions* of standard binary classification techniques to the harder setting of structured prediction. The framework of machine learning reductions (Beygelzimer et al., 2005) gives us an alternative methodology for *relating* one prediction problem to another. (Reductions have also been called "plug in classification techniques" (PICTs); see (James and Hastie, 1998) for an example.) The idea of a reduction is to map a hard problem to a simple problem, solve the simple problem, then map the solution to the simple problem into a solution to the hard problem.

2.3.1 Reduction Theory

A reduction has three components: the sample mapping, the hypothesis mapping and a bound. The sample mapping tells us how to create data sets for the simple problem based on data sets for the hard problem. The hypothesis mapping tells us how to convert a solution to the simple problem into a solution to the hard problem. The bound tells us that if we do well on the simple problem, we are guaranteed to also do well on the hard problem.

There are two varieties of bounds that are worth consideration: error-limiting bounds and regret-limiting bounds. In the case of an error-limiting reduction, the theoretical guarantee states that a low *error* on the simple problem implies a low *error* on the hard problem. For a regret-limiting reduction, the bound states that low *regret*[6] on the simple problem implies low *regret* on the hard problem. One particularly nice thing about reductions is that the bounds compose (Beygelzimer et al., 2005). In particular, if one can reduce problem A to problem B (with bound g) and problem B to problem C (with bound h), then the composed reduction $A \circ B$ has bound $g \circ h$.

In this section, I survey several prediction problems and corresponding reductions.

2.3.2 Importance Weighted Binary Classification

The importance weighted binary classification (IWBC) problem is a simple extension to the 0/1 binary classification problem. The difference is that in IWBC, each example has a corresponding weight. These weights reflect the importance of a correct classification. Formally, an IWBC is a distribution $\mathcal{D}$ over $\mathcal{X} \times 2 \times \mathbb{R}^+$. Each sample is a triple (x, y, i), where i is the importance weight. A solution is still a binary classifier $h : \mathcal{X} \to 2$, but the goal is to minimize the expected weight loss, given in Eq (2.22).

$$L(\mathcal{D}, h) = \mathbb{E}_{(x,y,i)\sim\mathcal{D}}\big[i \ \mathbf{1}(y \neq h(x))\big] \tag{2.22}$$

The "Costing" algorithm (Zadrozny, Langford, and Abe, 2003) is designed to reduce IWBC to binary classification. Costing functions by creating C parallel binary classification data sets based on a single IWBC data set. Each of these binary data sets are generated by sampling from the IWBC data set with probability proportional to the weights. Thus, examples with high weights are likely to be in most of the binary classification sets, and examples with low weights are likely to be in few (if any). After learning C different binary classifiers, one makes an importance weighted prediction by majority vote over the binary classifiers. Costing obeys the error bound given in Theorem 2.2.

Theorem 2.2 (Costing error efficiency; (Zadrozny, Langford, and Abe, 2003)). *For all importance weighted problems $\mathcal{D}$, if the base classifiers have error rate ϵ, then Costing has loss rate at most $\epsilon \ \mathbb{E}_{(x,y,i)\sim\mathcal{D}}[i]$.*

The proof of Theorem 2.2 is a straightforward application of the definitions. Intuitively, Costing works because examples with high weights are placed in most of the

[6] The regret of a hypothesis h on a problem $\mathcal{D}$ is the difference in error between using h and using the *best possible* classifier. Formally, $R(\mathcal{D}, h) = L(\mathcal{D}, h) - min_{h^*} L(\mathcal{D}, h^*)$.

buckets and examples with low weights are placed in few buckets. This means that, on average, the classifiers will perform better on the high weight examples. The expectation in the statement of Theorem 2.2 simply shows that the performance of the Costing reduction scales with the weights. In particular, if we multiply all weights by 100, then the weighted loss (Eq (2.22)) must also increase by a factor of 100.

2.3.3 Cost-sensitive Classification

Cost-sensitive classification is the natural extension of importance weighted binary classification to a multiclass setting. For a K-class task, our problem is a distribution $\mathcal{D}$ over $\mathcal{X} \times (\mathbb{R}^+)^K$. A sample $(x, \boldsymbol{c})$ from $\mathcal{D}$ is an input x and a cost vector $\boldsymbol{c}$ of length K. $\boldsymbol{c}$ encodes the costs of predictions. We learn a hypothesis $h : \mathcal{X} \to K$ and the cost incurred for a prediction is $c_{h(x)}$. In 0/1 multiclass classification, with a single "correct" class y and $K - 1$ incorrect classes, $\boldsymbol{c}$ is structured so that $c_y = 0$ and $c_{y'} = 1$ for all other y'. The goal is to find a classifier h that minimizes the expected cost-sensitive loss, given in Eq (2.23).

$$L(\mathcal{D}, h) = \mathbb{E}_{(x,\boldsymbol{c})\sim\mathcal{D}}\big[c_{h(x)}\big] \tag{2.23}$$

There are several reductions for solving this problem. The easiest is the "Weighted All Pairs" (WAP) reduction (Beygelzimer et al., 2005). WAP reduces cost-sensitive classification to importance weighted binary classification. Given a cost-sensitive example $(x, \boldsymbol{c})$, WAP generates $\binom{K}{2}$ binary classification problems, one for each pair of classes $0 \leq i < j < K$. The binary class is the class with lower cost and the importance weight is given by $|v_j - v_i|$, with $v_i = \int_0^{c_i} \mathrm{d}t \ 1/L(t)$, where $L(t)$ is the number of classes with cost at most t. WAP obeys the error bound given in Theorem 2.3.

Theorem 2.3 (WAP error efficiency; (Beygelzimer et al., 2005)). *For all cost-sensitive problems $\mathcal{D}$, if the base importance weighted classifier has loss rate c, then WAP has loss rate at most $2c$.*

Beygelzimer et al. (2005) provide a proof of Theorem 2.3. Intuitively, the WAP reduction works for the same reason that any all-pairs algorithm works: the binary classifiers learn to separate the good classes from the bad classes. The actual weights used by WAP are somewhat unusual, but obey two properties. First, if i is the class with zero cost, then the weight of the problem when i is paired with j is simply the cost of j. This makes intuitive sense. When i has greater than zero cost, then the weight associated with separating i from j is reduced from the difference to something smaller. This means the classifiers work harder to separate the best class from all the incorrect classes than to separate the incorrect classes from each other.

2.4 Discussion and Conclusions

This chapter has focused on three areas of machine learning: binary classification, structured prediction and learning reductions. The purpose of this thesis is to present a novel algorithm for structured prediction that improves on the state of the art. In particular, in the next chapter, I describe a novel algorithm, Searn, for solving the structured prediction problem. In particular, Searn is designed to be "optimal" in the sense of the summary from Table 2.1. That is, it is be amenable to any loss function, does not require an efficient solution to the argmax problem, is efficient and is easy to implement. However, unlike reranking, it also comes with theoretical guarantees and none of the disadvantages of reranking described in Section 2.2.9. Searn is developed by casting structured prediction in the language of reductions (Section 2.3); in particular, it reduces structured prediction to cost-sensitive classification (Section 2.3.3). At that point, one can apply algorithms like weighted-all-pairs and costing to turn it into a binary classification problem. Then, any binary classifier (Section 2.1) may be applied.

Chapter 3

Search-based Structured Prediction

As discussed in Section 2.2, structured prediction tasks involve the production of complex outputs, such as label sequences, parse trees, translations, etc. I described four popular algorithms for solving the structured prediction problem: the structured perceptron (Collins, 2002), conditional random fields (Lafferty, McCallum, and Pereira, 2001), max-margin Markov networks (Taskar et al., 2005) and SVMs for structured outputs (Tsochantaridis et al., 2005). As discussed previously, these methods all make assumptions of conditional independence which are known to not hold. Moreover, they all enforce unnatural limitations on the loss: namely, that it decomposes over the structure.

In this chapter, I describe SEARN (for "search + learn"). SEARN is an algorithm for solving *general* structured prediction problems: that is, ones under which the features and loss do not necessarily decompose. While SEARN is applicable to this restricted setting (and achieves impressive empirical performance; see Chapter 4), the true contribution of this algorithm is that it is the first generic structured prediction technique that is applicable to problems with non-decomposable losses. This is particularly important in real-world natural language processing problems because nearly all relevant loss functions do not decompose over any reasonable definition of structure. For example, the following metrics do not decompose naturally: the BLEU and NIST metrics for machine translation (Papineni et al., 2002; Doddington, 2004b); the ROUGE metrics for summarization (Lin and Hovy, 2003); the ACE metric for information extraction (Doddington, 2004a); and many others. That is not to say techniques do not exist for solving these problems: simply, there is no *generic*, well-founded technique for solving them.

One way of thinking about SEARN that may be most natural to researchers with a background in NLP is to first think about what problem-specific algorithms do. For example, consider machine translation (a problem *not* tackled in this thesis, though see Section 7.4 for a discussion). After a significant amount of training of various probability models and weighting factors, the algorithm used to perform the actual translation at test time is comparatively straightforward: it is a left-to-right beam search over English outputs.[1] An English translation is produced in an incremental fashion by adding words

[1] At least, this is the case for standard phrase-based (Koehn, Och, and Marcu, 2003) and alignment-template models (Och, 1999). More recent research into syntactic machine translation typically uses extensions of parsing algorithms for producing the output (Yamada and Knight, 2002; Melamed, 2004; Chiang, 2005). For simplicity, I will focus on the phrase-based framework.

or phrases on to the end of a given translation. This search process is performed to optimize some score: a function of the learned probability models and weights.

In fact, this approach is not limited to machine translation. Many complex problem in NLP are solved in a similar fashion: summarization, information extraction, parsing, etc. The problem that plagues all of these techniques is that the arg max problem from Eq (2.11)—finding the structured output that maximizes some score over features—is either formally intractable or simply too computationally demanding (for instance, parsing is technically polynomial, but $\mathcal{O}(N^3)$ is too expensive in practice, so complex beam and pruning methods are employed (Bikel, 2004)). The *theoretical* difficulty here is that although one might have employed machine learning techniques for which some performance guarantees are available (see Section 2.1.4), once one throws an ad hoc search algorithm on top, these guarantees disappear.[2]

SEARN, viewed from the perspective on NLP algorithms, can be seen as a generalization and *simplification* of this common practice. The key idea, developed initially by the incremental perceptron (see Section 2.2.4 and Collins and Roark (2004)) and the LASO framework (Daumé III and Marcu, 2005c), is to attempt to integrate learning with search. The two previous approaches achieve this integration by modifying a standard learning procedure to be aware of an underlying search algorithm. SEARN actually *removes* search from the prediction process altogether by directly learning a classifier to make incremental decisions. The prediction phase of a model learned with SEARN does not employ search but rather runs this classifier. In addition to gained simplicity, SEARN can handle more general features and loss functions and is theoretically sound.

3.1 Contributions and Methodology

What is a principled method for interleaving learning and search? To answer this, I analyze the desirable trait: good learning implies good search. This can be analyzed by casting SEARN as a learning reduction (Beygelzimer et al., 2005) that maps structured prediction to classification (see Section 2.3). I optimize SEARN so that good performance in binary classification implies good performance on the original problem.

The precise SEARN algorithm is inspired by research in reinforcement learning. Considering structured prediction in a reinforcement learning setting, I am able to leverage previous reductions for reinforcement learning to simpler problems (Langford and Zadrozny, 2003; Langford and Zadrozny, 2005). Viewed as a reinforcement learning algorithm, SEARN operates in an environment with oracle access to an optimal policy and gradually learns its own policy using an iterative technique motivated by Conservative Policy Iteration (Kakade and Langford, 2002) forming subproblems as defined by Langford and Zadrozny (2005). Relative to these algorithms, SEARN works from an optimal

[2]There is some related evidence from research on approximate inference in graphical models that roughly shows that the same approximate algorithm should be used for both training and prediction (Wainwright, 2006). In fact, even if possible to perform prediction *exactly*, if one trains using *the same* approximate algorithm, one should *test* using an approximate algorithm. This echoes some previous results I have showing roughly the same thing, but for a simple search-based sequence labeling algorithm (Daumé III and Marcu, 2005c).

policy rather than a restart distribution (Kakade and Langford, 2002) and can achieve computational speedups (Langford and Zadrozny, 2005) in practice.

The outcome of this work is an empirically effective algorithm for solving any structured prediction problem. In fact, I have a powerful *set* of algorithms because SEARN works using any classifier (SVM, decision tree, Bayes net, etc...) as a subroutine. This simple and general algorithm turns out to have excellent state-of-the-art performance and achieves significant computational speedups over competing techniques. For instance, the complexity of training SEARN for sequence labeling scales as $\mathcal{O}(TLk)$ where T is the sequence length, L is the number of labels and k is the Markov order on the features. M^3Ns and CRFs for this problem scale exponentially in k: $\mathcal{O}(TL^k)$ in general. Finally, SEARN is simple to implement.

3.2 Generalized Problem Definition

In Section 2.2.1, I defined two flavors of the structured prediction problem, specifically with respect to whether the loss function decomposes or not. In this chapter, I will focus exclusively on the harder care, where there is no decomposition. It turns out that it is convenient to actually consider a *generalization* of the problem defined previously. Recall that, before, the structured prediction problem was given by a fixed loss function and a distribution $\mathcal{D}$ over inputs $x \in \mathcal{X}$ and correct outputs $y \in \mathcal{Y}$. This is akin to the noise-free (or "oracle") setting in binary classification (Valiant, 1994; Kearns and Vazirani, 1997). I generalize this notion to a noisy setting by letting $\mathcal{D}$ be a distribution over pairs $(x, \boldsymbol{c})$, where the input remains the same ($x \in \mathcal{X}$), but where $\boldsymbol{c}$ is a cost vector so that for *any* output $y \in \mathcal{Y}$, c_y is the loss associated with predicting y. It is clear that any problem definable in the previous setting is definable in this generalization. This notion is stated formally in Definition 3.1.

Definition 3.1 (Structured Prediction). *A* structured prediction *problem $\mathcal{D}$ is a cost-sensitive classification problem where $\mathcal{Y}$ has structure: elements $y \in \mathcal{Y}$ decompose into variable-length vectors $(y_1, y_2, \dots, y_T)$.*[3] *$\mathcal{D}$ is a distribution over inputs $x \in \mathcal{X}$ and cost vectors $\boldsymbol{c}$, where $|\boldsymbol{c}|$ is a variable in 2^T.*

As a simple example, consider a parsing problem under F_1 loss. In this case, $\mathcal{D}$ is a distribution over $(x, \boldsymbol{c})$ where x is an input sequence and for all trees y with $|x|$-many leaves, c_y is the F_1 loss of y when compared to the "true" output.

The goal of structured prediction is to find a function $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the loss given in Eq (3.1).

$$L(\mathcal{D}, h) = \mathbb{E}_{(x,\boldsymbol{c})\sim\mathcal{D}} \left\{ c_{h(x)} \right\} \tag{3.1}$$

The technique I describe is based on the view that a vector $y \in \mathcal{Y}$ can be produced by predicting each component $(y_1, \dots y_N)$ in turn, allowing for dependent predictions. This is important for coping with general loss functions. For a data set $(x_1, c_1), \dots, (x_N, c_N)$

[3] *Treating y as a vector is simply a useful encoding; we are not interested only in sequence labeling problems. See Condition 1 in Section 2.2.1.*

of structured prediction examples, I write T_n for the length of the longest search path on example n, and $T_{\max} = \max_n T_n$.

3.3 Search-based Structured Prediction

I analyze the structured prediction problem by considering what happens at *test time.* Here, a search algorithm produces a full structured output by making a sequence of *decisions* at each time step. In standard structured techniques, this process of search aims to find a structure that maximizes a scoring function. I *ignore* this aspect of search and simply treat it as an iterative process that produces an output. In this view, the goal of search-based structured prediction is to find a function h that guides us through search. More formally, given an input $x \in \mathcal{X}$ and a state s in a search space $\mathcal{S}$, we want a function $h(x, s)$ that tells us the next state to go to (or, more generally, what *action* to take). This forms the basis of a policy.

Definition 3.2 (Policy). *A* policy *h is a distribution over actions conditioned on an input x and state s.*

Under this view of structured prediction, we have transformed the structured prediction problem into a classification problem. The classifier's job is to learn to predict best actions. The remaining question is how to train such a classifier, given the fact that the search spaces are typically too large to explore exhaustively.

3.4 Training

Searn operates in an iterative fashion. At each iteration it uses a known policy to create new cost-sensitive classification examples[4]. These examples are essentially the classification decisions that a policy would need to get right in order to perform search well. These are used to learn a new classifier which gives rise to a new policy. This new policy is interpolated with the old policy and the process repeats.

3.4.1 Cost-sensitive Examples

In the training phase, Searn uses a given policy π to construct cost-sensitive multiclass classification examples from which a new classifier is learned. These classification examples are created by *running* the given policy π over the training data. This generates one path per structured training example. Searn creates a single cost-sensitive example for each state on each path. The classes associated with each example are the available actions (the set of all possible next states). The only difficulty lies in specifying the costs.

Formally, we want the cost associated with taking an action that leads to state s to be the *regret* associated with this action, given our current policy. That is, we *search* under the input x_n using π and beginning at state s to find a complete output y. Under the

[4] A k-class cost-sensitive example is given by an input X and a vector of costs $\boldsymbol{c} \in (\mathbb{R}^+)^k$. Each class i has an associated cost c_i and the goal is a function $h : X \mapsto i$ that minimizes the expected value of c_i. See Section 2.3.3.

overall structured prediction loss function, this gives us a loss of c_y. Of all the possible actions, one, a', will have the minimum expected loss. The *cost* ℓ_a^π for an action a is the difference in loss between taking action a and taking the optimal action a'; see Eq (3.2).

$$\ell_a^\pi = \mathbb{E}_{y \sim search(x_n, \pi, a)} c_y - \min_{a'} \ell_{a'}^\pi \tag{3.2}$$

The complexity of the computation associated with Eq (3.2) is problem dependent. There are (at least) three possible ways to compute it.

1. Monte-Carlo sampling: one draws many paths according to h beginning at s' and average over the costs.
2. Single Monte-Carlo sampling: draw a single path and use the corresponding cost, with tied randomization as per Pegasus (Ng and Jordan, 2000).
3. Optimal approximation: it is often possible to efficiently compute the loss associated with following an *optimal policy* from a given state; when h is sufficiently good, this may serve as a useful and fast approximation. (This is also the approach described by Langford and Zadrozny (2005).)

The quality of the learned solution depends on the quality of the approximation of the loss. Obtaining Monte-Carlo samples is likely the best solution, but in many cases the optimal approximation is sufficient. An empirical comparison of these options is performed in Section 4.5.

3.4.2 Optimal Policy

Efficient implementation of Searn requires an efficient optimal policy π^* for the *training data* (it would make no sense on the test data: our problem would be solved). The implications of this assumption are discussed in detail in Section 3.6.1, but note in passing that it is strictly weaker than the assumptions made by other structured prediction techniques. The optimal policy is a policy that, for a given state, input and output (structured prediction cost vector) always predicts the best action to take:

Definition 3.3 (Optimal Policy). *For $x, \boldsymbol{c}$ as in Def 3.1, and a node $s = \langle y_1, \dots, y_t$ in the search space, the* optimal policy $\pi^*(x, \boldsymbol{c}, y)$ *is* $\arg\min_{y_{t+1}} \min_{y_{t+2},\dots,y_T} c_{\langle y_1,\dots,y_T \rangle}$. *That is, π^* chooses the action (i.e., value for y_{t+1}) that minimizes the corresponding cost, assuming that all* future *decisions are also made optimally.*

Searn uses the optimal policy to initialize the iterative process, and attempts to migrate toward a completely *learned* policy that will generalize well.

3.4.3 Algorithm

The Searn algorithm is shown in Figure 3.1. As input, the algorithm takes a data set, an optimal policy π^* and a multiclass learner L. Searn operates iteratively, maintaining a current policy hypothesis $h^{(I)}$ at each iteration I. This hypothesis is initialized to the optimal policy (step 1).

Algorithm $\textsc{Searn}(S^{\text{SP}}, \pi^*, \text{Learn})$

1: Initialize policy $h^{(0)} \leftarrow \pi^*$
2: **for** $I = 1 \ldots$ **do**
3:     Initialize the set of cost-sensitive examples $S_I \leftarrow \emptyset$
4:     **for** $n = 1 \ldots N$ **do**
5:         Compute path under the current policy $\langle s_1, \ldots, s_{T_n} \rangle \leftarrow pth(x_n, h^{(I-1)}, \emptyset)$
6:         **for** $t = 1 \ldots T_n$ **do**
7:             Compute features $\Phi = \Phi(x_n, s_t)$ for input x_n and state s_t
8:             Initialize a cost vector $\boldsymbol{c} = \langle\rangle$
9:             **for** each possible action a **do**
10:                 Compute the cost of a: $\ell_a = \ell^{h^{(I-1)}}_{s_t \oplus a}$ (Eq (3.2))
11:                 Append ℓ to $\boldsymbol{c}$: $\boldsymbol{c} \leftarrow \boldsymbol{c} \oplus \ell_a$
12:             **end for**
13:             Add cost-sensitive example $(\Phi, \boldsymbol{c})$ to S_I
14:         **end for**
15:     **end for**
16:     Learn a classifier on S_I: $h' \leftarrow \text{Learn}(S_I)$
17:     Interpolate: $h^{(I)} \leftarrow \beta h' + (1-\beta) h^{(I-1)}$
18: **end for**
19: **return** $h^{(\text{last})}$ without π^*

Figure 3.1: Complete Searn Algorithm

The algorithm then loops for a number of iterations. In each iteration, it creates a (multi-)set of cost-sensitive examples, S_I. These are created by looping over each structured example (step 4). For each example (step 5), the *current policy* $h^{(I-1)}$ is used to produce an full output, represented as a sequence of state $s_{1:T_n}$. Each state in the sequence is used to create a single cost-sensitive example (steps 6-14).

The first task in creating a cost-sensitive example is to compute the associated feature vector, performed in step 7. This feature vector is based on the structured input x_n *and* the current state s_t (the creation of the feature vectors is discussed in more detail in Section 3.4.6). We are now faced with the task of creating the cost vector $\boldsymbol{c}$ for the cost-sensitive classification examples. This vector will contain one entry for every possible action a that can be executed from state s_t. For each action a, we compute the expected loss associated with the state $s_t \oplus a$: the state arrived at assuming we take action a (step 10). This loss is then appended to the cost vector (step 11).

Once all example have been processed, Searn has created a large set of cost-sensitive examples S_I. These are fed into any cost-sensitive classification algorithm, Learn, to produce a new classifier h' (step 16). In step 17, Searn *combines* the newly learned classifier h' with the current classifier $h^{(I-1)}$ to produce a new classifier $h^{(I)}$. This combination is performed through linear interpolation with interpolation parameter β. (The choice of β is discussed in Section 3.5.) Finally, after all iterations have been completed, Searn returns the final policy after removing π^* (step 19).

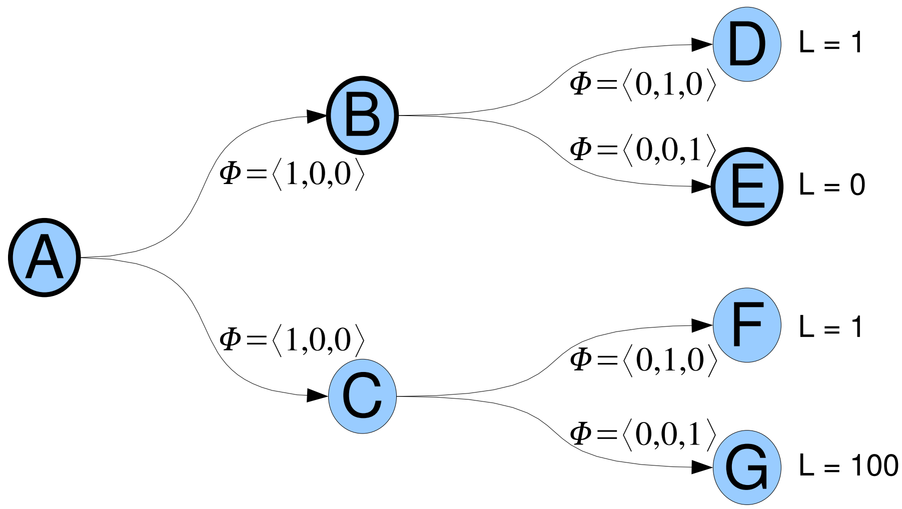


Figure 3.2: Example structured prediction problem for motivating the SEARN algorithm.

3.4.4 Simple Example

As an example to demonstrate how SEARN functions, consider the very simple search problem displayed in Figure 3.2. This can be thought of as a simple sequence labeling problem, where the sequence length is two (the "A" is given) and the correct output, shown in bold, is "A B E." This sequence achieves a loss of zero. Two other outputs ("A C F" and "A B D") achieve a loss of one, while the sequence "A C G" incurs a loss of one hundred. Along each edge is shown a feature vector corresponding to this edge. These vectors have no intuitive meaning, but serve to elucidate some benefits of SEARN. In this problem, there are three features, each of which is binary, and only one of which is active for any given edge.

Before considering what SEARN does on this problem, consider what a maximum entropy Markov model (Section 2.2.5) would do. The MEMM would use this example to construct two binary classification problems. For the "B/C" choice, this would lead to a positive example (corresponding to taking the "upper path") with feature vectors as shown in the figure. Then, a second example would be generated for the "D/E" choice. This would be a negative example with corresponding feature vectors.[5] After training a vanilla maximum entropy model on this data, one would obtain a weight vector $\boldsymbol{w} = \langle 0, 0, -1 \rangle$.

Now, consider what happens when we execute search using this policy. In the first step, we must decide between "B" and "C". Given the learned weight vector, both have value 0, so the algorithm must randomly choose between them. Suppose it chooses the upper path. Then, at the choice between "D" and "E", it will choose "E", yielding a loss

[5]In the MEGAM (`http://hal3.name/megam/`) "explicit, fval" notation, these examples would be written:

0 F1 1 # F1 1
1 F2 1 # F3 1

of zero. However, suppose it chooses the lower path on the first step. Then, at the choice between "F" and "G" it will choose "G", yielding a loss of 100. This leaves us with an expected loss of 50.5. This is far from optimal. Consider, for instance, the weight vector $\langle 0, 1, 0\rangle$. With this weight vector, the first choice is again random, but the "D/E" choice will lead to "D" and the "F/G" choice will lead to "F". This yields an expected loss of 1, significantly better than the learned weight vector.

The reason that this example fails is because we have only trained our weight vector on parts of the search space ("A" and "B") that the optimal path covers. This means that if we fall off this path at any point, we can do (almost) arbitrarily badly (this is formalized shortly in Theorem 3.4).

Now, consider executing Searn on this example. In the first step, Searn will generate an identical data set to the MEMM, on which the same weight vector will be learned. Searn will then iterate, with a current policy equal to an interpolation of the optimal policy and the learned policy given by the weight vector. In the second iteration of Searn, two things can happen: (1) the learned policy is called at the first step and it chooses "C" randomly, or (2) either the optimal policy or the learned policy is called at the first step and it chooses "B". In case (2), we will regenerate the same examples, relearn a new weight vector and re-interpolate (note that the more times this happens, the less likely it is that in the first step we call the optimal policy).

The interesting case is case (1). Here, just as before, we generate the first "B/C" choice example. However, when we follow the current policy, it chooses to go to node "C" instead of node "B". This means that instead of generating the second binary example as a choice between "D" and "E", instead we generate a second binary example as a choice between "F" and "G". Moreover, the second example is weighted much more strongly[6]. Now, when we learn a classifier off this data, we obtain a weight vector $\langle 0.01, 1, 0\rangle$, quite close to the hypothetical weight vector considered previously.

Consider the behavior of the algorithm with the newly learned weight vector. At the first step, the algorithm will select between "B" and "C" randomly. If it chooses "B", then it will choose "D" at the next step (score of 1 versus 0) and incur a loss of 1. If it chose "C" at the first step, it will choose "F" in the second step (score of 1 versus 0) and incur a loss of 1. This leads to an expected loss of 1, which is, in fact, the best one can do on this simple example.

3.4.5 Comparison to Local Classifier Techniques

There are essentially two varieties of local classification techniques applied to structured prediction problems. The first variety is typified by the work of Punyakanok and Roth (2001) and Punyakanok et al. (2005). In this variety, the structure in the problem is ignored all together, and a single classifier is trained to predict each element in the output vector *independently*. In some cases, a post-hoc search or optimization algorithm is applied on top to ensure some consistency in the output (Punyakanok, Roth, and Yih, 2005). The second variety is typified by maximum entropy Markov models (see

[6]One can simulate this in MegaM format as:
0 F1 1 # F1 1
0 $$$WEIGHT 100 F2 1 # F3 1

Section 2.2.5), though the basic idea has also been applied more generally to SVMs (Kudo and Matsumoto, 2001; Kudo and Matsumoto, 2003; Giménez and Màrquez, 2004). In this variety, the elements in the prediction vector are made sequentially, with the nth element conditional on outputs $n-k \dots n-1$ for a kth order model.

One way of contrasting Searn-based learning to more typical algorithms such as CRFs and M^{3}Ns is based on considering how they share information across a structure. The standard approach to sharing information is based on using the Viterbi algorithm (or, more generally, any exact dynamic programming algorithm) at *test time*. By applying such an search algorithm, one allows information to be shared across the entire structure, effectively "trading off" one decision for another. Searn takes an alternative approach. Instead of using a complex search algorithm at test time, it attempts to share information at *training time*. In particular, by training the classifier using a loss based on both past experience and future expectations, the training attempts to integrate this information during learning. One approach is not necessarily better than the other; they are simply different ways to accomplish the same goal.

In the purely independent classifier setting, both training and testing proceed in the obvious way. Since the classifiers make one decision completely independently of any other decision, training makes us only of the input. This makes training the classifiers incredibly straightforward, and also makes prediction easy. In fact, running Searn with $\Phi(x, y)$ independent of all but y_n for the n prediction would yield exactly this framework (note that there would be no reason to iterate Searn in this case). While this renders the independent classifiers approach attractive, it is also significantly weaker, in the sense that one cannot define complex features over the output space. This has not thus far hindered its applicability to problems like sequence labeling (Punyakanok and Roth, 2001), parsing and semantic role labeling (Punyakanok, Roth, and Yih, 2005), but does seem to be an overly strict condition. This also limits the approach to Hamming loss.

Searn is more similar to the MEMM-esque prediction setting. The key difference is that in the MEMM, the nth prediction is being made on the basis of the k previous predictions. However, these predictions are noisy, which potentially leads to the suboptimal performance described in the previous section. The essential problem is that the models have been trained assuming that they make all previous predictions correctly, but when applied in practice, they only have predictions about previous labels. It turns out that this can cause them to perform nearly arbitrarily badly. This is formalized in the following theorem, due to Matti Kääriäinen.

Theorem 3.4 (Kääriäinen (2006)). *There exists a distribution $\mathcal{D}$ over first order binary Markov problems such that training a binary classifier based on true previous predictions to an error rate of ϵ leads to a Hamming loss given in Eq* (3.3)*, where T is the length of the sequence.*

$$\frac{T}{2} - \frac{1-(1-2\epsilon)^{T+1}}{4\epsilon} + \frac{1}{2} \approx \frac{T}{2} \tag{3.3}$$

Where the approximation is true for small ϵ or large T.

The proof of this theorem is not provided, but I will give a brief intuition for the construction. Before that, notice that a Hamming loss of $T/2$ for a binary Markov problem is the same error rate as random guessing. The construction that leads to this error rate can be thought of as an XOR plus image recognition problem. The inputs are images of zeros and ones. The correct label for the nth label is the XOR of the number drawn in the nth image and the label at position $n-1$. A bit of thought can convince one that even a low error rate ϵ can lead to a high Hamming loss, essentially because once the algorithm errs, it cannot recover.[7]

One can construct similarly difficult problems for structured prediction distributions with different structure, such as larger order Markov models, models whose features can look at larger windows of the input, and multiclass cases. One might be led to believe that the result above is due to the fact that the classifier is trained on the true output, rather than its own predictions. In a sense, this is correct (and, in the same sense, this is exactly the problem Searn is attempting to solve). However, even if the model is trained in a single pass, using its previous outputs as input, one can obtain essentially the same error bound as shown in Theorem 3.4, where the algorithm will perform arbitrarily badly.

3.4.6 Feature Computations

In step 7 of the Searn algorithm (Figure 3.1), one is required to compute a feature vector Φ on the basis of the structured input x_n and a given state s_t. In theory, this step is arbitrary. However, the performance of the underlying classification algorithm (and hence the induced structured prediction algorithm) hinges on a good choice for these features.

In general, I adhere to the following recipe for creating the feature vectors. At state s_t, there will be K possible actions, $a_1, \dots, a_K$. I treat Φ as the concatenation of K subvectors, one for each choice of the next action. Then, I compute features as one normally would for the position in x_n represented by s_t and "pair" each of these with each action a_k to produce the final feature vector.

This is perhaps best understood with an example. Consider the part-of-speech tagging problem under a left-to-right greedy search (see also Chapter 4). Suppose our input is the sentence "The man ate a big sandwich with pickles ." and suppose that our current state correspond to a tagging of the first five words as "Det Noun Verb Det Adj". We wish to produce the part-of-speech tag for the word "sandwich." Suppose there are five possibilities: Det, Noun, Verb, Adj and Prep.

The first step will be to compute a standard feature vector associated with the current position in the sentence (the 6th word). This will typically include features such as the current word ("sandwich") its prefix and suffix ("san" and "ich"), and similar features computed within a window (eg., that "a" is two positions to the left and "with" is one position to the right). Additionally, we often wish to consider some structured features, such as "the previous word is tagged Adj" and "the second previous word is tagged Det." This will lead to a canonical "base" feature vector ϕ.

To compute the full feature vector Φ, I take the cross product between $\langle 1, \phi \rangle$ and the set of possible actions (the "1" is a bias term). In this case, suppose that $|\phi| = S$; then,

[7] While this construction may seem somewhat artificial, it is not unlike the common case of coreference resolution in the literature domain, where an conversation exchange occurs between two parties, with the speaker alternating and not explicitly given. Discerning who is speaking at the nth line requires that one has not erred previously.

with K actions, the length of Φ will be $K \times (S+1)$. In particular, we will take every feature f in ϕ and create K action/feature pairs "the feature f is active and the current action is a_k." Taking all of these features together gives us the full feature vector Φ.

Assuming one uses the weighted-all-pairs algorithm (Section 2.3.3) to reduce the multiclass problem to a binary classification problem, it is often possible (and beneficial) to only give the underlying classifier a subset of the features when making binary decisions. For instance, after applying weighted-all-pairs, one will be solving classification problems that look like "does action Det look better than action Verb?" For answering such questions, it is reasonable to only feed the algorithm the features associated with the Det and Verb options. Doing so both increases computation efficiency and significantly reduces the burden on the underlying classifier.

3.5 Theoretical Analysis

Searn functions by slowly moving *away* from the optimal policy *toward* a fully learned policy. As such, each iteration of Searn will *degrade* the current policy. The main convergence theorem states that the learned policy is never much worse than the starting (optimal) policy. To simplify notation, I write T for $T_{\max}$.

It is important in the analysis to refer explicitly to the error of the classifiers learned during the process of Searn. I write Searn$(\mathcal{D}, h)$ to denote the distribution over classification problems generated by running Searn with policy h on distribution $\mathcal{D}$. For a learned classifier h', I write $\ell_h^{\text{CS}}(h')$ to denote the loss of this classifier on the distribution Searn$(\mathcal{D}, h)$.

The following lemma (proof in appendix) is useful:

Lemma 3.5 (Policy Degradation). *Given a policy h with loss $L(\mathcal{D}, h)$, apply a single iteration of* Searn *to learn a classifier h' with cost-sensitive loss $\ell_h^{CS}(h')$. Create a new policy h^{new} by interpolation with parameter $\beta \in (0, T/2)$. Then, for all $\mathcal{D}$, with $c_{max} = \mathbb{E}_{(x,\boldsymbol{c})\sim\mathcal{D}} \max_i c_i$ (with $(x, \boldsymbol{c})$ as in Def (3.1)):*

$$L(\mathcal{D}, h^{new}) \leq L(\mathcal{D}, h) + T\beta\ell_h^{CS}(h') + \frac{1}{2}\beta^2 T^2 c_{max} \tag{3.4}$$

This lemma states that applying a single iteration of Searn does not cause the structured prediction loss of the learned hypothesis to degrade too much (recall that, beginning with the optimal policy, by moving away from this policy, our loss will increase at each step). In particular, up to a first order approximation, the loss increases proportional to the loss of the learned classifier.

Given this lemma one can prove the following theorem (proof in appendix):

Theorem 3.6 (Convergence). *For all $\mathcal{D}$, after C/β iterations of* Searn *beginning with a policy h_0 with loss $L(\mathcal{D}, h_0)$, and average learned losses as Eq (3.5).*

$$\ell_{avg} = \frac{1}{C/\beta} \sum_{i=1}^{C/\beta} \ell_{h^{(i-1)}}^{cs}(h^{(i)}) \tag{3.5}$$

(Each loss is with respect to the learned policy at that iteration), the loss of the final learned policy h (without the optimal policy component) is bounded by Eq (3.6)*.*

$$L(\mathcal{D}, h) \leq L(\mathcal{D}, h_0) + CT\ell_{avg} + c_{max}\left(\frac{1}{2}CT^2\beta + T\exp[-C]\right) \tag{3.6}$$

This theorem states that after C/β iterations of the Searn algorithm, the learned policy is not much worse than the quality of the optimal policy h_0. Finally, I state the following corollary that suggests a choice of the constants β and C from Theorem 3.6. The proof is by algebra.

Corollary 3.7. *For all $\mathcal{D}$, with $C = 2\ln T$ and $\beta = 1/T^3$ the loss of the learned policy is bounded by:*

$$L(\mathcal{D}, h) \leq L(\mathcal{D}, h_0) + 2T\ln T\ell_{avg} + (1 + \ln T)c_{max}/T$$

Although using $\beta = 1/T^3$ and iterating $2T^3\ln T$ times is guaranteed to leave us with a provably good policy, such choices might be too conservative in practice. In the experimental results described in future chapters, I use a development set to perform a line search minimization to find per-iteration values for β and to decide when to stop iterating. This is an acceptable approach for the following reason. The analytical choice of β is made to ensure that the probability that the newly created policy only makes *one* different choice from the previous policy for any given example is sufficiently low. The choice of β assumes the worst: the newly learned classifier will *always* disagree with the previous policy. In practice, this rarely happens. After the first iteration, the learned policy is typically quite good and only rarely differs from the optimal policy. So choosing such a small value for β is unneccesary: even with a higher value, the current classifier will not often disagree with the previous policy.

3.6 Policies

Searn functions in terms of policies, a notion borrowed from the field of reinforcement learning. This section discusses the nature of the optimal policy assumption and the connections to reinforcement learning.

3.6.1 Optimal Policy Assumption

The only assumption Searn makes is the existence of an optimal policy π^*, defined formally in Definition 3.3. For many simple problems under standard loss functions, it is straightforward to compute π^* in constant time. For instance, consider the sequence labeling problem (discussed further in Chapter 4). A standard loss function used in this task is Hamming loss: of all possible positions, how many does our model predict incorrectly. If one performs search left-to-right, labeling one element at a time (i.e., each element of the y vector corresponds exactly to one label), then π^* is trivial to compute. Given the correct label sequence, π^* simply chooses at position i the correct label at position i. However, Searn is not limited to simple Hamming loss. A more complex loss function often considered for the sequence segmentation task is F-score over (correctly labeled) segments. As discussed in Section 4.1.3, it is just as easy to compute the optimal policy for this loss function. This is not possible in many other frameworks, due to the non-additivity of F-score. This is independent of the features.

This result—that SEARN can learn under strictly more complex structures and loss functions than other techniques—is not limited to sequence labeling, as demonstrated in Theorem 3.8. In order to prove this, I need to formalize what I consider as "other techniques." I use the max-margin Markov network (M^3N) formalism (Section 2.2.7) for comparison, since this currently appears to be the most powerful generic framework. In particular, learning in M^3Ns is often tractable for problems that would be #P-hard in conditional random fields. The M^3N has several components, one of which is the ability to compute a loss-augmented minimization (Taskar et al., 2005). This requirement states that Eq (3.7) is computable for any input x, output set $\mathcal{Y}_x$, true output y and weight vector $\boldsymbol{w}$.

$$opt(\mathcal{Y}_x, y, \boldsymbol{w}) = \arg\min_{\hat{y} \in \mathcal{Y}_x} \boldsymbol{w}^\top \Phi(x, \hat{y}) + l(y, \hat{y}) \tag{3.7}$$

In Eq (3.7), $\Phi(\cdot)$ produces a vector of features, $\boldsymbol{w}$ is a weight vector and $l(y, \hat{y})$ is the loss for prediction $\hat{y}$ when the correct output is y.

Theorem 3.8. *Suppose Eq* (3.7) *is computable in time $T(x)$; then the optimal policy is computable in time $\mathcal{O}(T(x))$. Further, there exist problems for which the optimal policy is computable in constant time and for which Eq* (3.7) *may require exponential computation.*

See the appendix for a proof.

3.6.2 Search-based Optimal Policies

One advantage of the SEARN algorithm and the theory presented in Section 3.5 is that they do not actually hinge on having an optimal policy to train against. One can use SEARN to train against *any* policy. By Corollary 3.7, the loss of the learned policy simply contains a linear factor $L(\mathcal{D}, h_0)$ for the loss of the policy against which we train. If one trains against an optimal policy $L(\mathcal{D}, h_0) = 0$, but for non-optimal policies, the result still holds. Importantly one does not need to know the value of $L(\mathcal{D}, h_0)$ to use SEARN.

One artifact of this observation is that one can use *search* as a surrogate optimal policy for SEARN. That is, it may be the case that it is impossible to construct a search space in such a way that both computing the optimal policy and computing appropriate features are easy. For example, in the machine translation case, the left-to-right decoding style is natural and integrates nicely with an n-gram language model feature, but renders the computation of a BLEU-optimal policy intractable.

The solution is the following. Recall that when applying SEARN, we have an input x and a cost vector $\boldsymbol{c}$ (alternatively, we have a "true output" y and a loss function). At any step of SEARN, we need to be able to compute the best next action (note that this is the *only* requirement that needs to be fulfilled to apply SEARN). That is, given a node in the search space, and the cost vector $\boldsymbol{c}$, we need to compute the best step to take. This is *exactly* the standard search problem: given a node in a search space, we find the shortest path to a goal. By taking the first step along this shortest path, we obtain an optimal policy (assuming this shortest path is, indeed, shortest). This means that when Searn asks for the best next step, one can execute *any* standard search algorithm to compute this, for cases where the optimal policy is not available analytically.

The interesting thing to notice here is that under this perspective, we can see Searn as learning how to search. That is, there is some underlying search algorithm that is near optimal (because it knows the true output), and Searn is attempting to learn a policy to mimic this algorithm as closely as possible. From the perspective of the theory, all the bounds apply in this case as well, and the policy degradation by training on a search-based policy rather than a truly optimal policy is at most the difference in performance between the two policies.

Given this observation, we have reduced the requirement of Searn: instead of requiring an optimal policy, we simply require that one can perform efficient approximate search. This leads to the question: is this always possible. Though this is not a theorem, there is some intuition that this should be the case. For contradiction, suppose that we can *not* construct a search algorithm that does well (against which we could train). This means that *knowing* the cost vector (equivalently, knowing the *correct output*), we cannot construct a search algorithm that can find a low-loss output. If, *knowing* the correct output, we cannot find a good one, the learning problem seems hopeless. However, as always, it is up to the practitioner to structure the search space so that search and learning can be successful.

3.6.3 Beyond Greedy Search

The foregoing analysis assumes that the search runs in a purely greedy fashion. In practice, employing a more complex search technique, such as beam search, is useful. Fortunately, there is a straightforward mapping from beam search to greedy search by modifying the search space. Instead of moving a single robot in a search space, we can consider moving a beam of k robots in that space. This corresponds to a larger space whose elements are configurations of k robots. The only difference between the two is that the expected length of a search path, T, may increase.

Formally, there is a small issue with how to choose which robot's output to select to make the final prediction. The method I employ is as follows. Once a robot has created a full output, it must make one final "I'm done" decision. Once a single robot chooses the "I'm done" action, the search process ends with this robot's output. There are several advantages to doing the final step in this manner. It does not add bias by having an arbitrary selection procedure. Moreover, it enables the algorithm to learn to make the final decision quickly, if possible. In a sense, it also subsumes the reranking approach (see Section 2.2.9). This is because, in the worst case, all robots will find completed hypotheses and then the final decision is just a classification task between all possible "I'm done" action. This is very similar to a reranking problem. The advantage to running Searn in this manner is that it no longer makes sense to apply reranking as a postprocessing step to Searn: it should never be beneficial.

In general, Searn makes no assumptions about how the search process is structured. A different search process will lead to a different bias in the learning algorithm. It is up to the designer to construct a search process so that (a) a good bias is exhibited and

(b) computing the optimal policy is easy. For instance, for some combinatorial problems such as matchings or tours, it is known that left-to-right beam search tends to perform poorly. For these problems, a local hill-climbing search is likely to be more effective in the sense that it will render the underlying classification problems simpler.

From a theoretical perspective, so long as computational complexity issues are ignored, there is no reason to consider anything more than greedy search. This is because any search algorithm can feign as a greedy algorithm. When asked for a greedy step, the algorithm runs the complex search algorithm to completion and then returns the first step taken by this algorithm. While this obviates the intention behind greedy search, our theoretical results are complexity-agnostic and hence *cannot* be improved by moving to more complex search techniques.

One interesting corollary of this analysis has to do with the notion of NP-completeness. One might look at the foregoing as giving a method for solving arbitrarily complex problems in a purely greedy fashion, thus showing that FP=FNP. A closer inspection will reveal where this argument breaks down: we have only shown FP=FNP *if* the underlying binary classifier can achieve an error rate of 0. This means that (assuming FP≠FNP) one of the following must happen for computationally hard structured prediction problems. (1) The sample complexity of the underlying binary classification problems *must* become unwieldy. (2) The computational complexity of learning an optimal binary classifier *must* grow exponentially. There is a trade-off between the complexity of the search algorithm (and hence the expected length of the search path) and the underlying sample complexity. We could predict the entire structure in one step with low T but high sample complexity, or we could predict the structure in many steps with (hopefully) lower sample complexity. Balancing this trade-off is an open question.

3.6.4 Relation to Reinforcement Learning

Viewing the structured prediction problem as a search problem enables us to see parallels to reinforcement learning; see (Singh, 1993; Sutton and Barto, 1998) for introductions.

Definition 3.9 (Reinforcement Learning). *A* reinforcement learning *problem $\mathcal{R}$ is a conditional probability table $\mathcal{R}(o', r \mid (o, a, r)^*)$ on an observation set O and rewards $r \in [0, \infty)$ given any (possibly empty) history $(o, a, r)^*$ of past observations, actions (from an action set A), and rewards.*

The goal of the (finite horizon) reinforcement learning problem is as follows. Given some horizon T, find a policy $\pi : (o, a, r)^* \to a$, optimizing the expected sum of rewards: $\eta(\mathcal{R}, \pi) = \mathbb{E}_{(o,a,r)^T \sim \mathcal{R},\pi}\{\sum_{t=1}^{T} r_t\}$. Here, r_t is the tth observed reward, and the expectation is over the process which generates a history using $\mathcal{R}$ and choosing actions from π.

It is possible to map a structured prediction problem $\mathcal{D}$ to a (degenerate) reinforcement learning problem $\mathcal{R}(\mathcal{D})$ as follows. The reinforcement learning action set A is the space of indexed predictions, so $A^k = \mathcal{Y}$, and $A = \mathcal{Y}_i$. The observation o' is x initially, and the empty set otherwise. The reward r is zero, except at the final iteration when it is the negative loss for the corresponding structured output. Putting this together, one can define a reinforcement learning problem $\mathcal{R}(\mathcal{D})$ according to the following rules: When the history is empty, $o' = x$ and $r = 0$, where x is drawn from the marginal $\mathcal{D}(x)$. For all non-empty histories, $o' = \emptyset$. The reward r is zero, except when $t = k$, in which case $r = -c_{\boldsymbol{a}}$, where $\boldsymbol{c}$ is drawn from the conditional $\mathcal{D}(\boldsymbol{c} \mid x)$, and $c_{\boldsymbol{a}}$ is the $\boldsymbol{a}$th value of $\boldsymbol{c}$, thinking of $\boldsymbol{a}$ as an index.

Solving the search-based structured prediction problem is equivalent to solving the induced reinforcement learning problem. For a policy π, we define *search*(π) to be the structured prediction algorithm that behaves by searching according to π. The following theorem states that these are, in fact, equivalent problems (the proof is a straightforward application of the definitions):

Theorem 3.10. *Let $\mathcal{D}$ be an structured prediction problem and let $\mathcal{R}(\mathcal{D})$ be the induced reinforcement learning problem. Let π be a policy for $\mathcal{R}(\mathcal{D})$. Then $\eta(\mathcal{R}(\mathcal{D}), \pi) = L(\mathcal{D}, \textit{search}(\pi))$ (where L is from Eq* (3.1)*), where η denotes regret (the difference in loss between the optimal policy and the learned policy).*

It is important to notice that Searn does *not* solve the reinforcement learning problem. Searn is limited to cases where one has access to an optimal policy: this is rarely (if ever!) the case in reinforcement learning, since having an optimal policy would be all one would need. However, for the limited case of reinforcement learning where all observations are made initially and an optimal policy *is* available (which is essentially *exactly* the structured prediction problem), Searn is an appropriate algorithm.

One can think of Searn as an approach motivated by "training wheels." By starting with the optimal policy, it is like having someone show you how to ride a bike. After one "iteration," you forget a bit of the optimal policy (you "weaken" the training wheels) and are forced to use your own learned experience to compensate. Eventually, you use none of the optimal policy (you completely remove the training wheels) and ride the bike on your own.

One can imagine solving structured prediction by following the normal reinforcement learning practice of starting from a random (or uniform) policy and trying to get better, rather than following the Searn approach of starting from optimal and trying to not get much worse. My concern with doing things in the standard way is local maxima. That is, by starting from the optimal policy, we hope that any maximum we learn will be close to the global maximum. On the other hand, if one begins with a uniform policy and applies a standard reinforcement learning algorithm like conservative policy iteration (Kakade and Langford, 2002) to structured prediction setting, one obtains a loss bound that depends on T^2 (Daumé III, Langford, and Marcu, 2005). This is worse than the $T \ln T$ bound achieved by Searn (Theorem 3.6), but the comparison is somewhat void, since both are upper bounds.

3.7 Discussion and Conclusions

I have presented an algorithm, Searn, for solving structured prediction problems. Most previous work on structured prediction has assumed that the loss function and the features decompose identically over the output structure (Punyakanok and Roth, 2001; Taskar et al., 2005). When the features do not decompose, the arg max problem becomes intractable; this has been dealt with previously by augmenting the structured perceptron to acknowledge a beam-search strategy (Collins and Roark, 2004). To my knowledge, no previous work has dealt with the problem of loss functions that do not decompose (such as those commonly used in problems like machine translation, summarization and entity detection and tracking). As Searn makes no assumptions about decomposition (either on the features or the loss), it is applicable to a strictly greater number of problems than previous techniques (such as the summarization problem described in Chapter 6). Moreover, by treating predictions sequentially rather than independently (Punyakanok and Roth, 2001), Searn can incorporate useful features that encompass large spans of the output.

In addition to greater generality, Searn is computationally faster on standard problems. This means that in addition to yielding comparable performance to previous algorithms on small data sets, Searn is able to easily scale to handle all available data (see Chapter 4). Searn satisfies a strong fundamental performance guarantee: given a good classification algorithm, Searn yields a good structured prediction algorithm. In fact, Searn represents a family of structured prediction algorithms depending on the classifier and search space used. One general concern with algorithms that train on their own outputs is that the classifiers may overfit, leading to overly optimistic performance on the training data. This could lead to poor generalization. This concern is real, but does not appear to occur in practice (see Chapter 4, 5 and 6).

The efficacy of Searn hinges on the ability to compute an optimal (or near-optimal) policy. For many problems including sequence labeling and segmentation (Chapter 4) and some versions of parsing (see, for example, the parser of Sagae and Lavie (2005), which is amenable to a Searn-like analysis), the optimal policy is available in closed form. For other problems, such as the summarization problem described in the Chapter 6 and machine translation, the optimal policy may not be available. In such cases, the suggested approximation is to perform explicit search. There is a strong intuition that it should *always* be possible to perform such search under the assumption that the underlying problem should be learnable. This implies that Searn is applicable to nearly any structured prediction problem for which we have sufficient prior knowledge to design a good search space and feature function.

Chapter 4

Sequence Labeling

Sequence labeling is the task of assigning a *label* to each element in an input sequence. Sequence labeling is an attractive test bed for structured prediction algorithms because it is likely the simplest non-trivial structure. The canonical example sequence labeling problem from natural language processing is part of speech tagging. In part of speech (POS) tagging, one receives a sentence as input and is require to assign a POS to each word in the sequence. The set of possible parts of speech varies by data set, but is typically on the order of 20-40. For reasonable sentences of length 30, the number of possible outputs is is in excess of $1e48$. Despite the comparative simplicity of this task, this set is far too large to exhaustively explore without further assumptions.

Modern state-of-the-art structured prediction techniques fare very well on sequence labeling problems. However, in order to maintain tractability in search and learning, one is required to make a *Markov assumption* in the features. This is essentially a locality assumption on the *outputs.* Specifically, a k-th order Markov assumption means that no feature can reference the value of output labels whose position differs by more than k positions. It is well known that language does not obey the Markov assumption. For instance, whether "monitor" is a noun or verb at the beginning of a document is strongly correlated with how the same word would be tagged at the end of a document. Nevertheless, for many applications, it appears to be a reasonable approximation.

In this chapter, I present a wide range of results investigating the performance of SEARN on four separate sequence labeling tasks: handwriting recognition, named entity recognition (in Spanish), syntactic chunking and joint chunking and part-of-speech tagging. These results are presented for two reasons. The first reason is that previous structured prediction algorithms have reported excellent results on these problems. This allows us to compare SEARN directly to these other algorithms under identical experimental conditions. The second reason is that the simplicity of these problems allow us to compare both the various tunable parameters of SEARN and the affect of the Markov assumption in these domains.

This chapter is structured as follows. In Section 4.1 I describe the four sequence labeling tasks on which I evaluate: specifically, the data sets and the features used. In Section 4.2 I discuss the loss functions considered for the sequence labeling tasks. In Section 4.3 I describe how SEARN may be applied to these loss functions. In Section 4.4 I present experimental results comparing the performance of SEARN under different base

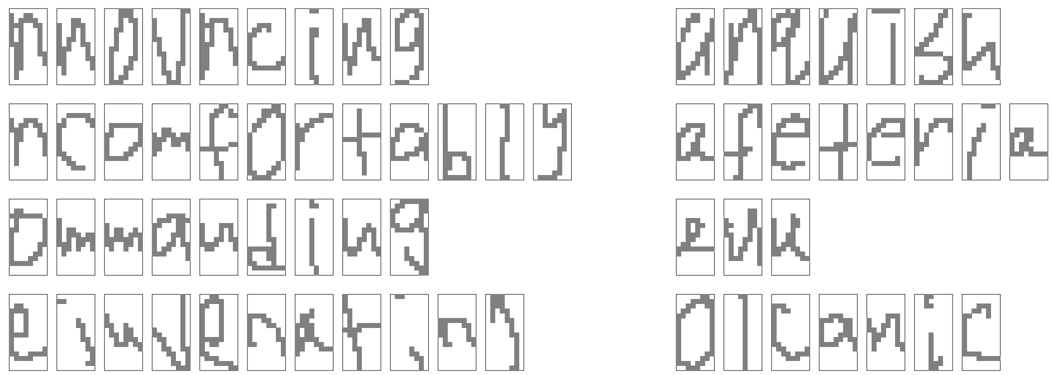


Figure 4.1: Eight example words from the handwriting recognition data set.

classifiers against the alternative structured prediction algorithms from Section 2.2. Finally, in Section 4.5 I compare the performance of Searn under different choices of the tunable parameters.

4.1 Sequence Labeling Problems

In this section, I describe the four tasks to which I apply Searn: handwriting recognition, Spanish named entity recognition, syntactic chunking and joint chunking and part-of-speech tagging.

4.1.1 Handwriting Recognition

The handwriting recognition task I consider was introduced by Kassel (1995). Later, Taskar, Guestrin, and Koller (2003) presented state-of-the-art results on this task using max-margin Markov networks. The task is an image recognition task: the input is a sequence of pre-segmented hand-drawn letters and the output is the character sequence ("a"-"z") in these images. The data set I consider is identical to that considered by Taskar, Guestrin, and Koller (2003) and includes 6600 sequences (words) collected from 150 subjects. The average word contains 8 characters. The images are 8×16 pixels in size, and rasterized into a binary representation. Two example image sequences are shown in Figure 4.1 (the first characters are removed because they are capitalized).

The standard features used in this task are as follows. For each possible output letter, there is a unique feature that counts how many times that letter appears in the output. Furthermore, for each pair of letters, there is an "edge" feature counting how many times this pair appears adjacent in the output. These edge features are the *only* "structural features" used for this task (i.e., features that span multiple output labels). Finally, for every output letter and for every pixel position, there is a feature that counts how many times that pixel position is "on" for the given output letter. In all, there are $26 + 26^2 + 26 \times (8 \times 16) = 4030$ features for this problem. This is the identical feature set

El presidente de la [Junta de Extremadura]$_{\mathsf{ORG}}$, [Juan Carlos Rodríguez Ibarra]$_{\mathsf{PER}}$, recibirá en la sede de la [Presidencia del Gobierno]$_{\mathsf{ORG}}$ extremeño a familiares de varios de los condenados por el proceso “ [Lasa-Zabala]$_{\mathsf{MISC}}$ ” , entre ellos a [Lourdes Díez Urraca]$_{\mathsf{PER}}$, esposa del ex gobernador civil de [Guipúzcoa]$_{\mathsf{LOC}}$ [Julen Elgorriaga]$_{\mathsf{PER}}$; y a [Antonio Rodríguez Galindo]$_{\mathsf{PER}}$, hermano del general [Enrique Rodríguez Galindo]$_{\mathsf{PER}}$.

Figure 4.2: Example labeled sentence from the Spanish Named Entity Recognition task.

to that used by Taskar, Guestrin, and Koller (2003). In the results shown later in this chapter, all comparison algorithms use identical feature sets.

In the experiments, I consider two variants of the data set. The first, “small,” is the problem considered by Taskar, Guestrin, and Koller (2003). In the small problem, ten fold cross-validation is performed over the data set; in each fold, roughly 600 words are used as training data and the remaining 6000 are used as test data. In addition to this setting, I also consider the “large” reverse experiment: in each fold, 6000 words are used as training data and 600 are used as test data.

4.1.2 Spanish Named Entity Recognition

The named entity recognition (NER) task is a subtask of the EDT task discussed in Chapters 1 and 5. Unlike EDT, NER is concerned only with spotting mentions of entities with no coreference issue. Moreover, in NER we only aim to spot *names* and neither pronouns (“he”) nor nominal references (“the President”). NER was the shared task for the 2002 Conference on Natural Language Learning (CoNLL). The data set consists of 8324 training sentences and 1517 test sentences; examples are shown in Figure 4.2. A 300-sentence subset of the training data set was previously used by Tsochantaridis et al. (2005) for evaluating the SVM$^{\text{struct}}$ framework in the context of sequence labeling. The small training set was likely used for computational considerations. The best reported results to date using the full data set are due to Ando and Zhang (2005). I report results on both the “small” and “large” data sets.

Named entity recognition is not naturally a sequence labeling problem: it is a *segmentation and labeling* problem: first, we must segment the input into phrases and second we must label these phrases. There are two ways to approach such problems. The first method is to map the segmentation and labeling problem down to a pure sequence labeling problem. The preferred method for performing such a mapping is through the “BIO encoding” (Ramshaw and Marcus, 1995a). In the BIO encoding, non-names are tagged as “O” (for “out”), the first word in names of type X are tagged as “B-X” (“begin X”) and all subsequent name words are tagged as “I-X” (“in X”). While such an encoded enables us to apply generic sequence labeling techniques, there are advantages to performing the segmentation and labeling simultaneously (Sarawagi and Cohen, 2004). I discuss these advantages in the context of SEARN in Section 4.3.

The structural features used for this task are roughly the same as in the handwriting recognition case. For each label, each label pair and each label triple, a feature counts the number of times this element is observed in the output. Furthermore, the standard set of input features includes the words and simple functions of the words (case markings, [Great American]$_{\mathsf{NP}}$ [said]$_{\mathsf{VP}}$ [it]$_{\mathsf{NP}}$ [increased]$_{\mathsf{VP}}$ [its loan-loss reserves]$_{\mathsf{NP}}$ [by]$_{\mathsf{PP}}$ [$ 93 million]$_{\mathsf{NP}}$ [after]$_{\mathsf{PP}}$ [reviewing]$_{\mathsf{VP}}$ [its loan portfolio]$_{\mathsf{NP}}$, [raising]$_{\mathsf{VP}}$ [its total loan and real estate reserves]$_{\mathsf{NP}}$ [to]$_{\mathsf{PP}}$ [$ 217 million]$_{\mathsf{NP}}$.

Figure 4.3: Example labeled sentence from the syntactic chunking task.

prefix and suffix up to three characters) within a window of ± 2 around the current position. These input features are paired with the current label. This feature set is fairly standard in the literature, though Ando and Zhang (2005) report significantly improved results using a much larger set of features. In the results shown later in this chapter, all comparison algorithms use identical feature sets.

4.1.3 Syntactic Chunking

The final pure sequence labeling task I consider is syntactic chunking (for English). This was the shared task of the CoNLL conference in 2000. As before, the input to the syntactic chunking task is a sentence. The desired output is a segmentation and labeling of the base syntactic units (noun phrases, verb phrases, etc.). This data set includes 8936 sentences of training data and 2012 sentences of test data. An example is shown in Figure 4.3. As in the named entity recognition task, there are two ways to approach the chunking task: via the BIO encoding and directly. (Several authors have considered the *noun-phrase chunking* task instead of the full syntactic chunking task. It is important to notice the difference, though results on these two tasks are typically very similar, indicating that the majority of the difficulty is with noun phrases. I report scores on both problems.)

I use the same set of features across all models, separated into "base features" and "meta features." The base features apply to words individually, while meta features apply to entire chunks. The standard base features used are: the chunk length, the word (original, lower cased, stemmed, and original-stem), the case pattern of the word, the first and last 1, 2 and 3 characters, and the part of speech and its first character. I additionally consider membership features for lists of names, locations, abbreviations, stop words, etc. The meta features I use are, for any base feature b, b at position i (for any sub-position of the chunk), b before/after the chunk, the entire b-sequence in the chunk, and any 2- or 3-gram tuple of bs in the chunk. I use a first order Markov assumption (chunk label only depends on the most recent previous label) and all features are placed on labels, not on transitions. In the results shown later in this chapter, some of the algorithms use a slightly different feature set. In particular, the CRF-based model uses similar, but not identical features; see (Sutton, Sindelar, and McCallum, 2005) for details.

4.1.4 Joint Chunking and Tagging

In the preceding sections, I considered the single sequence labeling task: to each element in a sequence, a single label is assigned. In this section, I consider the *joint* sequence labeling task. In this task, each element in a sequence is labeled with *multiple* tags. A canonical example of this task is joint POS tagging and syntactic chunking (Sutton, Rohanimanesh, and McCallum, 2004). An example sentence jointly labeled for these two outputs is shown in Figure 4.4 (under the BIO encoding).

Great$^{\text{NNP}}_{\text{B-NP}}$ American$^{\text{NNP}}_{\text{I-NP}}$ said$^{\text{VBD}}_{\text{B-VP}}$ it$^{\text{PRP}}_{\text{B-NP}}$ increased$^{\text{VBD}}_{\text{B-VP}}$ its$^{\text{PRP\$}}_{\text{B-NP}}$ loan-loss$^{\text{NN}}_{\text{I-NP}}$ reserves$^{\text{NNS}}_{\text{I-NP}}$ by$^{\text{IN}}_{\text{B-PP}}$ \$$^{\text{\$}}_{\text{B-NP}}$ 93$^{\text{CD}}_{\text{I-NP}}$ million$^{\text{CD}}_{\text{I-NP}}$ after$^{\text{IN}}_{\text{B-PP}}$ reviewing$^{\text{VBG}}_{\text{B-VP}}$ its$^{\text{PRP\$}}_{\text{B-NP}}$ loan$^{\text{NN}}_{\text{I-NP}}$ portfolio$^{\text{NN}}_{\text{I-NP}}$.$^{\text{.}}_{\text{O}}$

Figure 4.4: Example sentence for the joint POS tagging and syntactic chunking task.

Under a naïve implementation of joint sequence labeling, where the J label types (each with L_j classes) are collapsed to a single tag, the complexity of exact dynamic programming search with Markov order k scales as $\mathcal{O}((\prod_j L_j)^{k+1})$. For even moderately large L_j or k, this search quickly becomes intractable. In order to apply models like conditional random fields, one has to resort to complex and slow approximate inference methods, such as message passing algorithms (Sutton, Rohanimanesh, and McCallum, 2004).

Fundamentally, there is little difference between standard sequence labeling and joint sequence labeling. I use the same data set as for the standard syntactic chunking task (Section 4.1.3) and essentially the same features. The only difference in features has to do the structural features. The structural features I use include the obvious Markov features on the individual sequences: counts of singleton, doubleton and tripleton POS and chunk tags. I also use "crossing sequence" features. In particular, I use counts of pairs of POS and chunk tags at the same time period as well as pairs of POS tags at time t and chunk tags at $t-1$ and vice versa.

4.2 Loss Functions

For pure sequence labeling tasks (i.e., when segmentation is not also done), there are two standard loss functions: whole-sequence loss and Hamming loss. Whole-sequence loss gives credit only when the entire output sequence is correct: there is no notion of partially correct solutions. Hamming loss is more forgiving: it gives credit on a per label basis. For a true output y of length N and hypothesized output $\hat{y}$ (also of length N), these loss functions are given in Eq (4.1) and Eq (4.2), respectively.

$$\ell^{\text{WS}}(y,\hat{y}) \triangleq \mathbf{1}\left[\bigvee_{n=1}^{N} y_n \neq \hat{y}_n\right] \tag{4.1}$$

$$\ell^{\text{Ham}}(y,\hat{y}) \triangleq \sum_{n=1}^{N} \mathbf{1}\left[y_n \neq \hat{y}_n\right] \tag{4.2}$$

It is fairly clear that both of these loss functions decompose over the structure of $\mathcal{Y}$. That is, for any permutation π, $\ell^{\text{Ham}}(y,\hat{y}) = \ell^{\text{Ham}}(\pi \circ y, \pi \circ \hat{y})$, where we treat π as a group action over the sequence. The proof of this statement is trivial.

The most common loss function for joint segmentation and labeling problems (like the named entity recognition and syntactic chunking problems) is F_1 measure over chunks. This is the geometric mean of precision and recall over the (properly-labeled) chunk identification task, given in Eq (4.3).

$$\ell^{\mathrm{F}}(y, \hat{y}) \triangleq \frac{2\,|y \cap \hat{y}|}{|y| + |\hat{y}|} \tag{4.3}$$

In Eq (4.3), the interpretation of cardinality and intersection is in terms of chunks. That is, the cardinality of y is simply the number of chunks identified. The cardinality of the intersection is the number of chunks in common (i.e., the number of correctly identified chunks). As can be seen in Eq (4.3), one is penalized both for identifying too many chunks (penalty in the denominator) and for identifying too few (penalty in the numerator). The advantage of F_1 measure over Hamming loss seen most easily in problems where the majority of words are "not chunks"—for instance, in gene name identification (McDonald and Pereira, 2005)—Hamming loss will often prefer a system that identifies *no* chunks to one that identifies some correctly and other incorrectly. Using a weighted Hamming loss can not completely alleviate this problem, for essentially the same reasons that a weighted zero-one loss cannot optimize F_1 measure in binary classification, though one can often achieve an approximation (Lewis, 2001; Musicant, Kumar, and Ozgur, 2003).

4.3 Search and Optimal Policies

The choice of "search" algorithm in Searn essentially boils down to the choice of output vector representation, since, as defined, Searn always operates in a left-to-right manner over the output vector. In this section, we describe vector representations for the output space and corresponding optimal policies for Searn.

4.3.1 Sequence Labeling

The most natural vector encoding of the sequence labeling problem is simply as itself. In this case, the search will proceed in a greedy left-to-right manner with one word being labeled per step. This search order admits some linguistic plausibility for many natural language problems. It is also attractive because (assuming unit-time classification) it scales as $\mathcal{O}(NL)$, where N is the length of the input and L is the number of labels, independent of the number of features or the loss function. However, this vector encoding is also highly biased, in the sense that it is perhaps not optimal for some (perhaps unnatural) problems.

An alternative vector encoding is the following. We begin with a completely unlabeled sequence and, at each search step, we label a single (arbitrarily positioned) word. After sufficiently many steps have passed, we end this process. We can overwrite old labels. It is possible to define this search process as a vector in the following encoding. Let $N' \gg N$ be a "time limit." Define our vectors as sequences of length N' over the label set $N \times L$. The intuition is that choosing the label (n, l) means that the element at position n is now labeled with label l. After N' steps, we take the most recent label for each position as the final label (and an arbitrary label for any unspecified position).

This "unordered" search procedure is attractive because it does not require us to hard-code a search order. In practice, we might expect the algorithm to learn to first predict the positions it is sure about and the later move on to the less sure positions when more global information is available (nearby words have been labeled). We might hope that this will lead to a less biased algorithm. In fact, if N' is sufficiently large, this representation would potentially allow the search algorithm to mimic belief propagation (Yedidia, Freeman, and Weiss, 2003) over the sequence. We do, however, pay a cost for this added flexibility. The label space has increased by a factor of N, which means that (again, assuming unit-time classification) the algorithm now scales as $\mathcal{O}(N^2 L)$, which is reasonable only for short sequences. While this is perhaps unattractive for sequence labeling problems, this seems like an entirely reasonable approach to image segmentation problems.

4.3.2 Segmentation and Labeling

For joint segmentation and labeling tasks, such as named entity identification and syntactic chunking, there are two natural encodings: word-at-a-time and chunk-at-a-time. In word-at-a-time, one essentially follows the "BIO encoding" and tags a single word in each search step. In chunk-at-a-time, one tags single *chunks* in each search step, which can consist of multiple words (after fixing a maximum phrase length).

Under the word-at-a-time encoding, an input of length N leads to a vector of length N over $L + 1$ labels. Here L of the labels correspond to "begin" a phrase, while the $L + 1$st label corresponds to "continue the current phrase." Any vector that begins with the $L + 1$st label attains maximal loss.

Under the chunk-at-a-time encoding, an input of length N leads to a vector of length N over $M \times L + 1$ labels, where M is the maximum phrase length. The interpretation of the first $M \times L$ labels, for instance (m, l) means that the next phrase is of length m and is a phrase of type l. The "+1" label corresponds to a "complete" indicator. Any vector for which the sum of the "m" components is not exactly N attains maximum loss.

Just as there is a natural "unordered" search procedure for standard sequence labeling, there is also a natural unordered search procedure both for word-at-a-time chunking and chunk-at-a-time chunking.

Other search orders (or, more precisely, vector representations) are possible. For instance, one could perform right-to-left decoding or inside-out decoding or first decode odd positions then even. All of these will exhibit different biases, which may or may not be good for the particular problem and data set.

4.3.3 Optimal Policies

For the sequence labeling problem under Hamming loss, the optimal policy is essentially always to label the next word correctly. In the left-to-right order, this is straightforward. In the arbitrary ordering cases, after $n < N$ words have been tagged correctly, there are $N - n$ possible steps the optimal policy could take. It could either tag a currently untagged word correctly, or repair a previously incorrectly tagged word. In practice, I deterministically choose to tag the left-most untagged word first, until no words are tagged, at which point the policy corrects the tag of the left-most incorrectly tagged word first. One could alternatively randomize over these choices, but this might introduce too much noise into the system.

For sequence labeling under zero-one loss, the optimal policy is the same as for Hamming loss. Technically, once an error has been made, the optimal policy is agnostic as to future choices. This could be encoded in a randomized policy. In practice, I use the same policy as for Hamming loss.

As far as the segmentation problem, word-at-a-time and chunk-at-a-time behave very similarly with respect to the loss function and optimal policy. I will discuss word-at-a-time because its notationally more convenient, but the difference is negligible. The optimal policy can be computed by analyzing a few options in Eq (4.4)

$$\pi^*(x, y_{1:T}, \hat{y}_{1:t-1}) = \begin{cases} \text{begin } X & y_t = \text{begin } X \\ \text{in } X & y_t = \text{in } X \text{ and } \hat{y}_{t-1} \in \{\text{begin } X, \text{in } X\} \\ \text{out} & \text{otherwise} \end{cases} \tag{4.4}$$

It is fairly straightforward to show that this policy is optimal. There is, actually, another optimal policy. For instance, if y_t is "in X" but $\hat{y}_{t-1}$ is "in Y" (for $X \neq Y$), then it is equally optimal to select $\hat{y}_t$ to be "out" or "in Y". In theory, when the optimal policy does not care about a particular decision, one can randomize over the selection. However, in practice, I always default to a particular choice to reduce noise in the learning process.

For all of the policies described above, it is also straightforward to compute the optimal approximation for estimating the expected cost of an action. In the Hamming loss case, the loss is 0 if the choice is correct and 1 otherwise. For the whole-sequence loss, the loss is 0 if the choice is correct *and* all previous choices were correct and 1 otherwise. Note that under whole-sequence loss, once an error has been made, the cost function becomes ambivalent between future alternatives. The computation for F_1 loss is a bit more complicated: one needs to compute an optimal intersection size for the future and add it to the past "actual" size. This is also straightforward by analyzing the same cases as in Eq (4.4).

4.4 Empirical Comparison to Alternative Techniques

In this section, I compare the performance of Searn to the performance of alternative structured prediction techniques over the data sets described in Section 4.1. The results of this evaluation are shown in Table 4.1. In this table, I compare raw classification algorithms (perceptron, logistic regression and SVMs) to alternative structured prediction algorithms (structured perceptron, CRFs, $\text{SVM}^{\text{struct}}$s and M^3Ns) to Searn with three baseline classifiers (perceptron, logistic regression and SVMs). For all SVM algorithms and for M^3Ns, I compare both linear and quadratic kernels (cubic kernels were evaluated but did not lead to improved performance over quadratic kernels).

For all Searn-based models, I use the the following settings of the tunable parameters (see Section 4.5 for a comparison of these settings). I use the optimal approximation for the computation of the per-action costs. I use a left-to-right search order with a beam of size 10. For the chunking tasks, I use chunk-at-a-time search. I use weighted all pairs and costing to reduce from cost-sensitive classification to binary classification.

Note that some entries in Table 4.1 are missing. The vast majority of these entries are missing because the algorithm considered could not reasonably scale to the data set under consideration. These are indicated with a "$\sim$" symbol. Other entries are not

ALGORITHM	Handwriting		NER		Chunk	C+T
	Small	Large	Small	Large		
CLASSIFICATION						
Perceptron	65.56	70.05	91.11	94.37	83.12	87.88
Log Reg	68.65	72.10	93.62	96.09	85.40	90.39
SVM-Lin	75.75	82.42	93.74	97.31	86.09	93.94
SVM-Quad	82.63	82.52	85.49	85.49	∼	∼
STRUCTURED						
Str. Perc.	69.74	74.12	93.18	95.32	92.44	93.12
CRF	–	–	94.94	∼	94.77	96.48
SVM$^{\text{struct}}$	–	–	94.90	∼	–	–
M^{3}N-Lin	81.00	∼	–	–	–	–
M^{3}N-Quad	87.00	∼	–	–	–	–
SEARN						
Perceptron	70.17	76.88	95.01	97.67	94.36	96.81
Log Reg	73.81	79.28	95.90	98.17	94.47	96.95
SVM-Lin	82.12	90.58	95.91	98.11	94.44	96.98
SVM-Quad	87.55	90.91	89.31	90.01	∼	∼

Table 4.1: Empirical comparison of performance of alternative structured prediction algorithms against SEARN on sequence labeling tasks. (Top) Comparison for whole-sequence 0/1 loss; (Bottom) Comparison for individual losses: Hamming for handwriting and Chunking+Tagging and F for NER and Chunking. SEARN is always optimized for the appropriate loss.

available simply because the results I report are copied from other publications and these publications did not report all relevant scores. These are indicated with a "–" symbol.

We observe several patterns in the results from Table 4.1. The first is that structured techniques consistently outperform their classification counterparts (eg., CRFs outperform logistic regression). The single exception is on the small handwriting task: the quadratic SVM outperforms the quadratic M^{3}N.[1] Additionally, for all classifiers, adding SEARN consistently improves performance.

An obvious pattern worth noticing is that moving from the small data set to the large data set results in improved performance, regardless of learning algorithm. However, equally interesting is that simple classification techniques when applied to large data sets outperform complicated learning techniques applied to small data sets. Although this comparison is not completely fair—both algorithms should get access to the same data—if the algorithm (like the SVM$^{\text{struct}}$ or the M^{3}N) cannot scale to the large data set, then something is missing. For instance, a vanilla SVM on the large handwriting data set outperforms the M^{3}N on the small set. Similarly, a vanilla logistic regression classifier

[1] However, it should be noted that a different implementation technique was used in this comparison. The M^{3}N is based on an SMO algorithm, while the quadratic SVM is libsvm (Chang and Lin, 2001).

trained on the large NER data set outperforms the SVM^{struct} and the CRF on the small data sets.

The important observation is that, on the same data set, Searn can perform comparably or better than competing structured prediction techniques. On the small handwriting task, the two best performing systems are M^{3}Ns with quadratic kernels (87.0% accuracy) and Searn with quadratic SVMs (87.6% accuracy). On the NER task, Searn with a perceptron classifier performs comparably to SVM^{struct} and CRFs (at around 95.9% accuracy). On the Chunking+Tagging task, all varieties of Searn perform comparatively to the CRF. In fact, the only task on which Searn does *not* outperform the competition techniques is on the raw chunking task, for which the CRF obtains an F-score of 94.77 compared to 94.47 for Searn.

The final result from Table 4.1 worth noticing is that, with the exception of the handwriting recognition task, Searn using logistic regression as a base learner performs at the top of the pack. The SVM-based Searn models typically perform slightly better, but not significantly. In fact, the raw averaged perceptron with Searn performs almost as well as the logistic regression. This is a nice result because the SVM-based models tend to be expensive to train, especially in comparison to the perceptron. The fact that this pattern does not hold for the handwriting task is likely due to the fact that the data for this task is quite unlike the data for the other tasks. For the handwriting task, there are a comparatively small number of features, and are individually much less predictive of the class. It is only in combination that good classifiers can be learned.

4.5 Empirical Comparison of Tunable Parameters

In this section, I compare the performance of Searn as I adjust the tunable parameters. Unless otherwise specified, all experiments are on the small data sets. In all experiments, I use logistic regression as the base learner. This choice was made because, with the exception of the handwriting recognition task, the Searn with logistic regression was never significantly worse than any other base learner. Moreover, it is more efficient than the SVM, which also performed well. The parameters compared are:

- Computation of expected loss: by single Monte-Carlo sample, by ten samples or by the optimal approximation.
- Vector encoding: left-to-right or unordered; also, for segmentation problems, word-at-a-time versus chunk-at-a-time.
- Beam size: compare greedy search to beam search with different sized beams.
- Multiclass reduction: weighted all pairs versus unweighted all pairs.
- Number of iterations of Searn.

Not all parameters are compared on all data sets. For instance, the unordered vector encoding is only tested on the handwriting recognition task because the output sequences tend to be sufficiently short that paying the N^2 cost is possible. For all comparisons, one of the options will be taken as the "baseline" and the others will be compared by percentage

	Handwriting	**Spanish NER**	**Chunking**	**Chunk+Tag**
→ MC 1	+0.071%	+0.032%	+0.048%	−0.033%
→ MC 10	+0.092%	+0.052%	+0.069%	+0.051%

Table 4.2: Evaluation of computation of expected loss: differences between both single Monte-Carlo (MC 1) and ten Monte-Carlo (MC 10) against the optimal approximation.

	Spanish NER	**Chunking**
→ Word	−4.194%	−2.038%

Table 4.3: Evaluation of computation of vector encodings: changes in performance for using word-at-a-time rather than chunk-at-a-time encodings.

change in performance. Positive values mean improved performance by changing the baseline setting; negative values indicate decreased performance.

Table 4.2 shows the differences in system quality for models learned using alternative computations of the expected loss. The computation based on the optimal approximation is used as a baseline. The "→ MC 1" rows show the change in performance by using a single Monte-Carlo sample instead. As we can see, the performance does not change very much when moving to Monte-Carlo samples. A small degradation in performance is observed using single samples for the Chunking+Tagging problem. When 10 samples are used, all improvements are positive, but tiny. Given the computational overhead of using Monte-Carlo samples instead of the optimal approximation, for these problems, it is probably worthwhile to stay with the approximation.

Next, I compare left-to-right versus unordered vector encodings for the handwriting recognition task (small data set). By moving to an unordered vector encoding, the performance on this task rises by 0.48%. This gain comes at a significant computational cost, and is insufficient to make the logistic regression model competitive with the SVM models on this task.

In Table 4.3, I compare chunk-at-a-time search to word-at-a-time search for the segmentation problems: Spanish named entity recognition and syntactic chunking. Note that the feature space naturally expands somewhat when moving to a chunking framework. We can see that in the NER task, moving to word-at-a-time search significantly hurts performance. The change is smaller, but still negative, for the chunking task. This echos other recent results (Sarawagi and Cohen, 2004; Ciaramita and Altun, 2005).

In Table 4.4, different beam sizes are compared. The baseline is beam 10 search. The differences in performance for beam 10 versus greedy search are fairly pronounced, especially for the handwriting task. However, for the other tasks, the differences are comparatively small. The differences between beam 5, beam 10 and beam 25 are relatively negligible for all tasks. This suggests that at least for these problems, the necessity to propagate large amounts of uncertainty is low.

In Table 4.5, I compare the performance on the chunking problems when using unweighted all pairs (as opposed to weighted all pairs). We can see that there is a big performance drop for not using weighted classification for the NER task, but not so large

	Handwriting	**Spanish NER**	**Chunking**	**Chunk+Tag**
→ Greedy	−0.813%	−0.371%	−0.527%	−0.594%
→ Beam 5	−0.157%	−0.081%	−0.098%	−0.094%
→ Beam 25	+0.052%	+0.021%	+0.016%	+0.019%

Table 4.4: Evaluation of beam sizes: differences between beam search and greedy search (baseline is a beam of 10).

	Spanish NER	**Chunking**
→ Unweighted	−3.195%	−0.342%

Table 4.5: Evaluation of multiclass reduction strategies: comparing unweighted all pairs to weighted all pairs.

a drop for the chunking task. This difference can likely be attributed to the significance of the F-score for the two tasks. In NER, only very few words in a sentence are part of a chunk, but in chunking, almost every word is. This means that the weighting is much more pronounced in the NER task and therefore the results will be significantly more sensitive to the weighting for the NER problem.

Finally, in Figure 4.5, I plot the performance of the learned policy for the SEARN-based models for the four tasks as the number of iterations increases. For these graphs, I use a constant value of $\beta = 1$ for the interpolation: pure policy-iteration. The curves are somewhat different for each problem, but in general an optimum is reached in 5-15 iterations and then performance either levels off (eg., for syntactic chunking) or beings to drop (eg., for handwriting recognition). The drop in performance is likely due to overfitting. Note that these curves are the performance of the learned policy *without* the optimal policy on the test data, so these graphs do not contradict the SEARN theorem of uniform degradation of performance (Lemma 3.5).

In summary, this analysis has shown then following trends:

- For the computation of the expected loss, using the optimal approximation does not significantly worsen the results. Moreover, the optimal approximation is significantly more efficient. For these problems, it is probably the preferable approach.

- For the vector encoding, chunk-based search is significantly better than word-based search.

- For the beam size, a small beam tends to consistently outperform greedy search, but there is little bang for the buck when moving to larger and larger beam settings.

- For the multiclass reduction, using weighted-all-pairs is important, especially when the cost function being optimized is significantly different from raw accuracy.

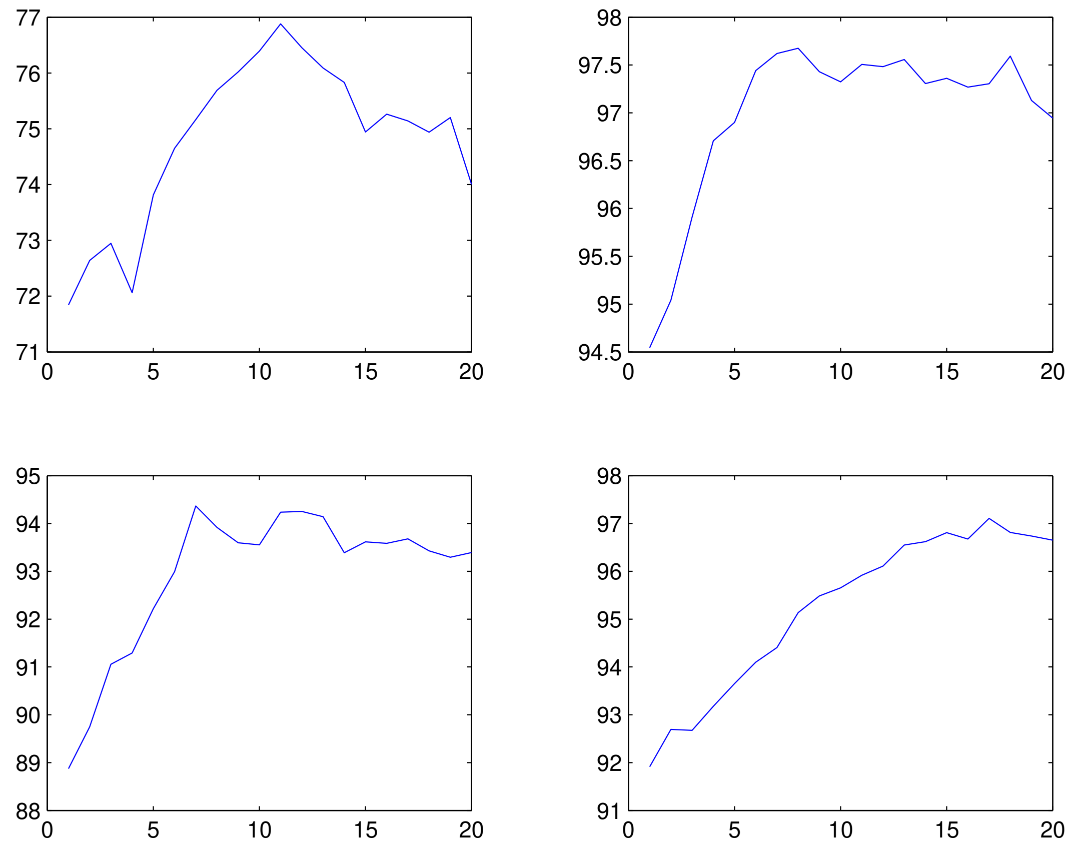


Figure 4.5: Number of iterations of Searn for each of the four sequence labeling problem. Upper-left: Handwriting recognition; Upper-right: Spanish named entity recognition; Lower-left: Syntactic chunking; Lower-right: Joint chunking/tagging.

4.6 Discussion and Conclusions

In this chapter, I have presented experimental results on simple sequence labeling tasks that showcase the efficacy of Searn (see Section 4.4). These results show that Searn is competitive with competing structured prediction techniques and scales better to large data sets. I have furthermore compared the internal settings of Searn (Section 4.5). These results have shown that the optimal approximation is an efficient, effective method for computing the losses associated with different actions. Small beams are sufficient for these problems, and chunk-at-a-time decoding leads to significant performance increases.

While these results are useful, they should be taken with a grain of salt. Sequence labeling is a very easy problem. The structure is simple and the most common loss functions decompose over the structure. The comparatively good performance of raw classifiers suggests that the importance of structure is minor. In fact, some results suggest that one need not actually consider the structure at all for some such problems (Punyakanok and Roth, 2001; Punyakanok et al., 2005).

An additional caveat is that performance is quite high in all of these problems. While this is a good phenomenon to observe, it also means that some generalizations might not extend to harder problems. For instance, it is easy to imagine that for harder problems, using larger beams *would* help significantly. Moreover, the optimal approximation is good largely because the learned classifiers tend to perform near optimally. Even if not, the dependence of the classifier's prediction on the $n+10$th word is relatively independent of its prediction on the nth word. For more problems with more complex structure, or for which performance is not so good, a larger performance improvement might be observed by using Monte-Carlo samples.

Chapter 5

Entity Detection and Tracking

This chapter focuses on the application of SEARN to the entity detection and tracking problem (introduced in Section 1.2). First, I more exactly define the problem from the perspective of the data set I use for evaluation. Second, I briefly survey prior work on this problem and discuss why SEARN is an attractive tool to apply to the EDT problem. Next, I describe the search structure I employ and the features. I then present empirical results showing state-of-the-art performance on the ACE 2004 EDT data set and conclude with a comparison of SEARN to standard models for this problem.

5.1 Problem Definition

The entity detection and tracking problem (EDT) focuses on discovering the set of entities discussed in a document and identifying the textual span of the document (the *mentions*) that refer to these entities. As part of the detection phase, a system must also identify, for each entity, its corresponding *entity type* (person, place, organization, etc.) and, for each mention of an entity, its *mention type* (name, nominal, pronoun, etc.).

In Figure 5.1 (reproduced from Figure 1.2), I show one paragraph from our data set, wherein entities have been identified, types have been disambiguated and coreference chains have been marked. In this paragraph, I underline every entity mention, each of which is followed by a superscript that identifies the *mention type* and a subscript that identifies both the *entity type* and *coreference chain* of that mention. For instance, the word "commander" is a nominal reference to a person, identified as entity number 2. At the beginning of the second sentence, the word "I" is a pronominal mention also referring to entity 2 (and hence is the same entity). A few of the coreference chains that appear in this extract are: {JERUSALEM}, {commander, I, Gen, Yitzhak Eitan}, {Israeli, Israeli} and {troops}.

The full entity detection and tracking task is to take an input document and produce the annotation of all entities and mentions, like that seen in Figure 1.2. This is typically broken into two phases: first, a *mention detection* phase underlines all mentions and assigns to them a mention type and entity type; second, a *coreference* phase links together the already-identified mentions by assigning the numbers to each mention (of course, the numbering scheme is arbitrary – any numbering that preserves identity would be just as good as another).

JERUSALEM$^{\text{NAM}}_{\text{GPE}-1}$ – The commander$^{\text{NOM}}_{\text{PER}-2}$ of Israeli$^{\text{PRE}}_{\text{GPE}-3}$ troops$^{\text{NOM}}_{\text{PER}-4}$ in the West Bank$^{\text{NAM}}_{\text{LOC}-5}$ said there was a simple goal to the helicopter$^{\text{PRE}}_{\text{VEH}-6}$ assassination on Thursday of a gun-wielding local Palestinian$^{\text{PRE}}_{\text{GPE}-7}$ leader$^{\text{NOM}}_{\text{PER}-8}$. " I$^{\text{PRO}}_{\text{PER}-2}$ hope it will reduce the violence and bring back reason to this area$^{\text{NOM}}_{\text{LOC}-9}$ " , Maj Gen$^{\text{PRE}}_{\text{PER}-2}$ Yitzhak Eitan$^{\text{NAM}}_{\text{PER}-2}$ told reporters$^{\text{NOM}}_{\text{PER}-10}$ at a briefing hours after three missiles$^{\text{NOM}}_{\text{WEA}-11}$ fired from an Apache$^{\text{PRE}}_{\text{VEH}-6}$ helicopter$^{\text{NOM}}_{\text{VEH}-6}$ killed Hussein Obaiyat$^{\text{NAM}}_{\text{PER}-8}$, along with two middle-aged women$^{\text{NOM}}_{\text{PER}-12}$ standing near his$^{\text{PRO}}_{\text{PER}-8}$ van$^{\text{NOM}}_{\text{VEH}-13}$ in Beit Sahur$^{\text{NAM}}_{\text{GPE}-14}$, near Bethlehem$^{\text{NAM}}_{\text{GPE}-15}$. Instead , it has touched off one of the bloodiest and most intense weekends of fighting yet in the six-week-old conflict , with gunfire crackling through the West Bank$^{\text{NAM}}_{\text{LOC}-5}$ and Gaza Strip$^{\text{NAM}}_{\text{LOC}-16}$. Five Palestinians$^{\text{NOM}}_{\text{PER}-17}$ and an Israeli$^{\text{PRE}}_{\text{GPE}-3}$ soldier$^{\text{NOM}}_{\text{PER}-18}$ were shot dead on Friday .

Figure 5.1: An example paragraph extract from a document from our training data with entities identified; reproduced from Figure 1.2.

Mention Type	**Description and Examples**
Name (NAM)	A proper name reference to an entity; for instance "Georg Cantor," "Congress," "Jerusalem," and "Microsoft" are all named references.
Nominal (NOM)	A common noun reference to an entity; for instance "the man," "the president," "the group," "the country" and "the company" are all nominal references.
Pronominal (PRO)	A pronoun reference to an entity; for instance "he," "they," "it," etc. This also includes words such as "who" in the construction "Georg Cantor, who died in an asylum, ..."
pre-modifier (PRE)	A reference by a salutation, position or other modifier; for instance "Palestinian" in the case of "Palestinian leader" or "President" in the case of "President Nixon."

Table 5.1: A list of the four possible mention types with descriptions and examples.

In the data set I use, there are four mention types, shown in Table 5.1, including names, pronouns, nominals and pre-modifiers. Further, there are seven entity types, shown in Table 5.1, with corresponding subtypes (there are no subtypes for the 'person' entity type). Of the entity types, I focus primarily on 'person,' 'organization,' 'location,' and 'GPE,' since these are the most common and the most general across different domains.

Like all natural language processing problems, the primary difficulty in the EDT task is ambiguity and the multiple diverse sources of information required to resolve this ambiguity. Consider, for instance, the example paragraph shown in Figure 1.2. Identifying that the "I" in the second sentence is the same person as the "commander" in the first sentence is an extremely challenging inference to make. In fact, it is possible that the two mentions actually refer to two different entities who happen to agree in what they say. Identifying that the "Gen" entity is the same as "Yitzhak Eitan" requires some knowledge of syntax, as does linking this entity with the pronoun "I." On the other

Entity Type	Description and Subtypes
Person (PER)	Person entities are limited to humans. A person may be a single individual or a group. *Subtypes:* None
Organization (ORG)	Organization entities are limited to those with an established organizational structure. *Subtypes:* government, commercial, educational, non-profit, other
GPE (GPE)	A Geographical-Political Entity. GPE entities are politically defined geographical regions. *Subtypes:* address, boundary, celestial, land-region-natural, region-local, region-subnational, region-national, region-international, water-body, other
Location (LOC)	Location entities are limited to geographic entities with physical extent. *Subtypes:* continent, nation, state-or-province, county-or-district, population-center, other
Facility (FAC)	Facility entities are human-made artifacts in the domains of architecture and civil engineering. *Subtypes:* building, subarea-building, bounded-area, conduit, path, barrier, plant, other
Vehicle (VEH)	Vehicle entities are human-made artifacts that are used for transportation purposes. *Subtypes:* land, air, water, subarea-vehicle, other
Weapon (WEA)	Weapon entities are human-made artifacts whose primary purpose is to cause damage. *Subtypes:* blunt, exploding, sharp, chemical, biological, shooting, projectile, nuclear, other

Table 5.2: A list of the seven entity types, with descriptions and subtypes.

hand, identifying that the "Apache" referred to in the second sentence is coreferent with "helicopter" form the first sentence requires external knowledge that an Apache is a type of helicopter. Identifying that "his" in the second sentence is coreferent with "Hussein Obaiyat" and not "Yitzhak Eitan" requires further syntactic knowledge.

In terms of the set of mention types (name, nominal, pronoun and pre-modifier), there are essentially three distinct sorts of coreference links that must be made: name-to-name, name-to-nominal and name-to-pronoun (pre-modifiers can typically be identified as either names or nominals and so they fall into one of the other three classes). Of these, name-to-name is by far the simplest and can be fairly easily solved with a set of features that look for string similarity. Name-to-pronoun is the case of pronoun resolution or anaphora resolution, which has been studied extensively by linguists, both pure and computational. It is still a difficult problem, but most of the information necessary for making this sort of decision can be found within the document (except for gender resolution issues). The name-to-nominal problem is hard and typically requires external knowledge to solve. The example from Figure 1.2 of the "Apache"/"helicopter" link is a prime example of this problem. In this case, there is a hint within the document that "Apache" is a type of "helicopter" (due to the pre-modifier position), but this cannot be counted on in general. A significant contribution of this thesis is in developing techniques for handling the name-to-nominal case.

5.2 Prior Work

The majority of prior work on the entity detection and tracking task splits it into two separate subproblems: first, one performs *mention detection* (finding the text spans that correspond to mentions of entities and identifying their entity types and mention types); then one performs *coreference resolution*, which groups the previously-identified mentions into *coreference chains*. (There are two notable exceptions. First, (Wellner et al., 2004) performs integrated extraction and coreference in the context of citation matching using conditional random field techniques. This is similar to the model discussed below in Section 5.2.2.3. Very recently, a second approach based entirely on reranking, has shown improved tagging performance when coreference is included (Ji and Grishman, 2004; Ji, Rudin, and Grishman, 2006).) Although my eventual goal is to build an integrated system, from the perspective of comparisons to prior work, I discuss these two subtasks and corresponding approaches separately. The machine learning approach to both of these tasks typically involves specifying a set of *features* that identify aspects of both the input (the document currently being processed) and the output (in the case of mention detection, the output tags; in the case of coreference resolution, the decisions about coreference chains).

5.2.1 Mention Detection

One can reduce the mention detection problem to a simple *sequence labeling problem*, such as those discussed in Chapter 4. Most typically, one applies the "BIO encoding" (Ramshaw and Marcus, 1995b) in order to cast the chunking problem as a tagging problem. Figure 5.2.1 shows the second half of the second sentence from Figure 1.2 in both the original sequence-based tagging and the BIO encoding.

Once one has reduced the mention detection problem to a sequence labeling problem, one can apply any of a plethora of models that have been developed for this task. All such models can be considered to be finite state machines with a Markovian independence assumption. In other words: the score assigned to a particular label sequence is a combination of the scores obtained by comparing each word to its corresponding label, and the scores obtained by comparing each label to the k labels that precede it. When $k = 0$, there is no dependence among labels. The common case of $k = 1$ is a first order Markov model, $k = 2$ is a second order Markov model and so on. The advantage of making such a Markov assumption is that training and decoding in such models becomes tractable through standard variants of the Viterbi algorithm (Viterbi, 1967) and

ORIG	Maj <u>Gen</u>$^{\text{PRE}}_{\text{PER}}$ <u>Yitzhak Eitan</u>$^{\text{NAM}}_{\text{PER}}$ told <u>reporters</u>$^{\text{NOM}}_{\text{PER}}$ at a briefing hours after three <u>missiles</u>$^{\text{NOM}}_{\text{WEA}}$ fired from an <u>Apache</u>$^{\text{PRE}}_{\text{VEH}}$ <u>helicopter</u>$^{\text{NOM}}_{\text{VEH}}$ killed <u>Hussein Obaiyat</u>$^{\text{NAM}}_{\text{PER}}$, . . .

BIO									
	Maj	Gen	Yitzhak	Eitan	told	reporters	at	a	
	OUT	B-PER	B-PER	I-PER	OUT	B-PER	OUT	OUT	
	briefing	hours	after	three	missiles	fired	from	an	Apache
	OUT	OUT	OUT	OUT	B-WEA	OUT	OUT	OUT	B-VEH
	helicopter	killed	Hussein	Obaiyat	,	. . .			
	B-VEH	OUT	B-PER	I-PER	OUT				

Figure 5.2: A partial sentence from our example text in original sequence-based format and in the BIO-encoding.

the forward-backward algorithm (Baum and Petrie, 1966; Baum and Eagon, 1967). Under these assumptions, any of the standard structured prediction methods discussed in Chapter 2 can be applied (structured perceptron, CRF, M^{3}N and SVM$^{\text{struct}}$).

The set of features historically used for mention detection is typically not very diverse. In the tagging case, features are functions of both the input document, x, and the output tag sequence, y. Specifically, an input element $x \in \mathcal{X}$ is simply a word sequence $x = \langle x_1, \dots, x_N \rangle$ and an output element y is a tag sequence $y = \langle y_1, \dots, y_N \rangle$. Features are computed over all positions in these sequences n and are typically broken into a set of features over the input, $i_f(x, n)$, and a set of features over the output, $o_g(y_n, \dots, y_{n-k})$. The restriction to using only the k most recent tags in the prediction is done purely for modeling reasons: to allow arbitrary-distance dependencies would result in exponential-time search and parameter estimation algorithms. For instance, an input feature might be of the form $i_1(x, l) = 1$ if the word x_l is "Cantor" and $i_1(x, l) = 0$, otherwise. An output feature might be of the form $o_1(y_l, \dots) = 1$ if y_l is the label B-PER and 0 otherwise. Combining all pairs of such features results in features having the form $i_1(x, l)o_1(y_l, \dots) = 1$ exactly when x_n is "Cantor" and y_n is B-PER.

Typical input features include: word identity, word stem identity, word prefixes and suffixes, word case form (uppercase, lowercase, initial upper case, etc.) and part of speech. These features are typically applied within a window of two or three words to the left and to the right of the word under current consideration. (For instance, the fact that the previous word is "Mr." might be a good indication that the current word will be B-PER.) Additionally, one typically considers word membership on lists drawn from gazetteers, such as lists of states, countries, common names, etc. The output features used are typically simply the identity of the current tag and the pair of the current tag and the previous tag. Sometimes the latter "Markov feature" is not combined with the input features and simply left as a "bias." I will discuss these features and more in Section 5.4.3, but the features listed here nearly exhaust those considered in the literature.

5.2.2 Coreference Resolution

When treated separately from mention detection, one can consider the input to a coreference resolution system to be a document where mentions and their types have been previously identified. The task then is to group these mentions into coreference sets. The first large-scale and successful machine learning approach to coreference resolution is based on a reduction to *binary classification* (Soon, Ng, and Lim, 2001; Ng and Cardie, 2002; Strube, Rapp, and Müller, 2002; Yang et al., 2003; Luo et al., 2004). Since this is a theme that will carry through nearly all previous attempts to solve this problem with machine learning, I spend some time discussing it here. (Note that this is not a formal reduction in the sense of Section 2.3, but rather an ad hoc mapping.)

5.2.2.1 Binary Classification

Reducing the coreference resolution problem to binary classification is appealing because there are many very good models for solving the binary classification problem. The general approach is to build a classifier that takes a pair of mentions (m_1, m_2) and produces a binary response: are m_1 and m_2 coreferent or not (i.e., do they refer to the same real-world entity). Given such a classifier, the task at test time is to use these predictions to generate coreference chains.

The choice of which classifier to use is essentially a matter of personal preference, under the constraints of computational efficiency. The most popular choices in the literature have been decision trees (Quinlan, 1993; Cohen, 1995), used by (Aone and Bennett, 1995; McCarthy and Lehnert, 1995; Soon, Ng, and Lim, 2001; Ng and Cardie, 2002; Strube, Rapp, and Müller, 2002; Strube and Müller, 2003; Yang et al., 2003), and maximum entropy models (Della Pietra, Della Pietra, and Lafferty, 1997; Berger, Della Pietra, and Della Pietra, 1996), used by (Kehler, 1997; Morton, 2000; Luo et al., 2004). This choice is essentially arbitrary, though different learning models have different biases and *fortés*, and further investigation may show one to be better on this task than another (though it is more likely that performance differences between models are due more to features than to the choice of learning algorithm).

One significant issue for applying a binary classification algorithm to coreference resolution is that of choosing pairs of mentions on which to train a binary classifier. The most obvious answer would be to select *all pairs*, and, indeed, many systems do precisely this (Aone and Bennett, 1995; Ng and Cardie, 2002; Cohen and Richman, 2002; Iida et al., 2003; Harabagiu, Bunescu, and Maiorano, 2001). There are, however, several drawbacks to choosing all pairs. First, from a computational perspective, a document that originally contained 60 mentions (which is about average for our data) will give rise to $60 * 59/2 = 1770$ training instances. Summing this over a corpus of roughly 400 documents leads to just over $700k$ training instances. This is too large for many machine learning techniques, especially the kernel-based learners such as support vector machines which tend to scale at least quadratically in the number of training inputs.

Another, more subtle, issue with using all pairs is that the resulting training set will be severely biased toward negative instances. On average, the 60 mentions in a document will refer to only about 20 actual entities (most of which are singletons), resulting in a positive-to-negative ratio on the order of roughly $1 : 12$ (in the case of our data). This means that from a *0/1 loss* perspective, a classifier that predicts everything to be "not coreferent" will attain a loss of only about 7.5%, a figure which is difficult to beat. The problem here lies in the fact that 0/1 is not an appropriate loss function to minimize, but it is also not immediately clear what *is* appropriate. One solution commonly employed would be to *weight* the positive examples as 12 times more important than the negative examples, but it is unclear how this particular choice effects the learned classifier, especially as relates to question 3 above (choice of clustering algorithm). See (Karakoulas and Shawe-Taylor, 1999; Caruana, 2000; Chawla et al., 2002) for further discussion of the imbalanced data set problem in general machine learning tasks.

A second approach to the problem of selecting training instances is to only compare a mention with the *most recent* mention in each coreference chain (Soon, Ng, and Lim, 2001). For instance, in the example shown in Figure 1.2, when generating the training examples for the name "Yitzhak Eitan," one would only generate a positive instance for this being coreferent with "Gen," and not with "commander" or "I." This alleviates, to some extent, the first problem with all pairs (namely that there are too many training instances), but actually makes the second problem (biased training set) worse, since one is now extracting far fewer *positive* instances, with no change in the number of *negative* instances extracted.

One general problem with the binary classification scheme is that the reduction is not exact: there are answers that can be given by the binary classifier that do not correspond to any possible decision at the level of coreference chains. This is essentially a problem in transitivity. Coreference chains, being essentially an equivalence class on mentions, must be transitive: if m_1 and m_2 are coreferent and m_2 and m_6 are coreferent, then it must be that m_1 and m_6 are coreferent. Unfortunately, the binary classifier that is treating each pair independently has no knowledge of this constraint. It might be very happy to say that the pair (m_1, m_2) is positive (they are coreferent), that the pair (m_2, m_6) is positive (also coreferent) but that the pair (m_1, m_6) is negative. This gives rise to an impossible set of constraints at the chain level.

There are essentially two approaches to solving this problem. The first is simply to only ever apply the binary classifier to the most recent mention from each chain, which will remove the problem of transitivity all together (Soon, Ng, and Lim, 2001; Strube, Rapp, and Müller, 2002). Unfortunately, this choice also makes the problem needlessly hard. For instance, in the example shown in Figure 1.2, this would mean that in order to resolve the reference of the second instance of the word "helicopter," one would only compare it to the word "Apache" and not to the previous instance of the word "helicopter." Presumably the latter would be a much simpler link to make.

The second approach to solving the transitivity problem, which is appropriate only when the classifier provides a numeric response rather than a class (which is the case for all the classifiers I have discussed), is to use clustering techniques to build the chains. Essentially, the responses produced by the classifier are transformed into similarities so that positive examples have high "similarity" and negative examples have low "similarity." These similarities are then fed into a generic clustering algorithm and the resulting clustering is used as the output of coreference resolution system (Aone and Bennett, 1995; Ng and Cardie, 2002; Cohen and Richman, 2002; Iida et al., 2003). One difficulty with this approach is that the classifier is trained completely independently from the clustering model that will eventually be used, so it is difficult to know whether the loss being optimized by the classifier is appropriate for the clustering algorithm. Moreover, to tune both the classifier's parameters and the clustering algorithm's parameters through cross-validation requires a non-trivial amount of re-computation to ensure unbiasedness.

5.2.2.2 Multilabel Classification

One can easily extend the binary classification scheme: rather than reducing the coreference problem to a binary classifier, one reduces it to a multiclass multilabel problem (i.e., a multiclass problem with multiple "correct" classes) (Florian et al., 2004; Luo et al., 2004). The idea here is that for the nth mention in a document, one constructs a single training instance over n-many classes. The first $n-1$ classes correspond to each of the previous mentions and the nth class corresponds to a "this is a new entity" class. From an optimization perspective, this approach seems preferable to the binary classification approach, since the number of training examples will no longer grow quadratically with the number of mentions, and there is a much less significant bias problem.

When applied at test time, the multilabel classifier will produce probabilities that the current mention is either a new entity or is coreferent with any of the previous mentions, individually: $p\,(m_n \text{ is new})$ versus $p\,(m_n \text{ is coreferent with } m_k)$ for all $k < n$. Of course, at this point one needs to combine all of this information in order to make a decision. For instance, at the second occurrence of the word "helicopter" in Figure 1.2, the model might say that there is a 60% chance this is coreferent with the first occurrence of "helicopter," a 20% chance it is coreferent with "Apache" and with the remaining 10% spread out across the other options. In (Florian et al., 2004; Luo et al., 2004), the choice was made that the probability of the new mention being in the "helicopter/Apache" cluster is the *maximum* of the probabilities over the elements of that cluster. In this case, that would mean that this has a probability of 60%.[1] The clustering technique described by (Florian et al., 2004; Luo et al., 2004) is based on a construction they term the *Bell tree*. The idea is simply to perform a standard greedy beam search over coreference chains in a left to right manner, where the score for an entire chain is simply the product of the probabilities obtained as decisions are made along this chain.

Despite these differences from the binary classifier, this model still potentially suffers from similar problems. The largest shortcoming is still that the model is trained completely independently of the clustering algorithm (the Bell tree algorithm) used for decoding the coreference chains. And since the classifier is still trained essentially by looking at pairs, one cannot include features that span entire chains.

[1] This choice seems incredibly strange. From the calculus of probabilities, this should be computing by summing. By taking the max, one is (a) throwing out information and (b) ending up with probabilities that do not sum to 100%. It is perhaps from experimental observation that taking the max rather than the sum performs better in practice, but it is not very well justified.

5.2.2.3 Random Fields

One recent work that is very similar to the work described in this thesis is based on treating the coreference resolution problem in the conditional random field framework (McCallum and Wellner, 2004). This framework identifies the coreference resolution problem with graph partitioning, where the vertices are mentions and the graph is fully connected. A partition is sought which maximizes a conditional probability expression. Unfortunately, training weights to maximize the log likelihood in this model is intractable, so the alternative approach of using the structured perceptron is employed (see Section 2.2.3). In this framework, the clustering algorithm used is a generic graph partitioning algorithm, which can be shown to be equivalent to solving the problem of finding the coreference sets with maximal probability. Unfortunately, graph partitioning is NP-complete, so an approximate algorithm is used (Bansal, Chawala, and Blum, 2002). The algorithm works by initializing the weights to zero, running the graph partitioning algorithm on training instances until a mistake is made, and using a simple additive update on the weights.

Of all the systems presented to this point, this model is probably the least objectionable from a theoretical perspective. It is unfortunate that so much information is lost by going from the full CRF framework to the perceptron framework, and that there is no "middle ground" that maintains tractability, but McCallum and Wellner's model *is* able to account for both the classification and clustering aspects of the problem through the integration of the learning and the graph cut algorithm. In the same context, one can employ a max-margin approach (Finley and Joachims, 2005), based on the SVM$^{\text{struct}}$ framework (Tsochantaridis et al., 2005).

5.2.2.4 Coreference Resolution Features

The classic features employed by coreference resolution systems focus on pairs of mentions; see (Ng and Cardie, 2002; Luo et al., 2004) for two prototypical examples. The most commonly computed features include: distance between the mentions (in terms of words, sentences and paragraphs), information about whether either (or both) are pronominal, information about whether the strings of the two mentions look "similar" (either exact match, substring or string edit distance), information about whether they are definite or demonstrative, information about whether they agree in terms of number, gender or semantic class (typically resolved using WordNet (Fellbaum, 1998)), and information about whether they appear in apposition to one another (for instance, "George Cantor, a famous mathematician, ..."). Hobbs' distance (Hobbs, 1976), or flat Hobbs' distance, attempts to assist with the resolution of pronouns by looking at dominance structure in parse trees. Other approaches have attempted (without success) to use verb subcategorization information to assist in pronoun resolution (Kehler et al., 2004). All of these features (and many more) will be discussed in the context of my coreference system in Section 5.5.3.

5.2.3 Shortcomings

During the foregoing description of mention detection and coreference systems, I attempted to point out individual flaws of various approaches. Here, I summarize some of the shortcomings of the previously described models, especially those that I attempt to solve by applying Searn to the EDT problem. However, the majority of the discussion of *how* these problems are solved will be relegated to the remainder of this chapter.

Pipelined Detection and Coreference: Without exception, all systems described up to this point have solved the EDT task (or, in the case of the older models, the named entity detection and coreference resolution task) in a pipelined fashion: first mentions are identified with type, then these are grouped into chains. This separation of tasks forces the mention detection phase to make many guesses it should not have to. For instance, upon encountering the pronoun "it," it is an incredibly difficult decision to decide on the entity type of this mention. If this decision is made incorrectly, the subsequent step of coreference resolution stands no chance of success. A similar argument can be made in the case of unfamiliar names or roles. For instance, suppose there is a very uncommon profession called a "flufferbugger" and one encounter a document which reads, "John Doe, a famous flufferbugger, ..." Identifying the occurrence of "flufferbugger" as a person is aided by the fact that one knows from syntactic cues that it is likely that this word is coreferent with "John Doe," but in order to be coreferent, you have to be an entity of the same type. This effect has been alluded to by some research which has shown that *reranking* candidate hypotheses from a mention detection model by a coreference model can result in better performance (Ji and Grishman, 2004). However, it is very likely that a model that *can* actually incorporate this knowledge in one step is going to outperform any reranking system; see Section 2.2.9 for more details.

Weak Features: The features employed by the mention detection systems described are only able to take advantage of local information. Linguistic studies of text structure have repeatedly shown though that there are clear dependencies between references throughout an entire document (Morris and Hirst, 1991; Stairmand, 1996; Halliday and Hasan, 1976), which is not yet captured by existing features. Furthermore, in the context of coreference, the string-matching features proposed to date are somewhat adequate at capturing name-to-name links, less adequate at capturing nominal-to-nominal links, and horribly inadequate at any other sort of link. Pronoun-to-name is weakly captured by distance-based functions, and, in some cases, through the implementation of the Hobbs' distance, but this case still remains a problem. The name-to-nominal case is largely ignored and further effort is necessary to solve this difficult aspect of the coreference resolution task. Finally, all decisions in current coreference systems are made pair-wise: features only ever talk about pairs of mentions. As I show empirically, it is very helpful to be able to define features over larger sets of mentions.

Non-integrated Classification and Clustering for Coreference: Separating the coreference problem into a classification (or distance metric learning) problem and a clustering (or search) problem is appealing because we know how to solve these problems, but it does not attend to the fundamental issue of learning in structured, highly interdependent domains. I have previously attempted to make some progress toward developing an integrated model in a Bayesian framework (Daumé III and Marcu, 2005a); unfortunately, the model proposed in that work is far too inefficient for practical purposes and it is unclear the extent to which it can deal with the largely overlapping features necessary to build a full coreference system (or to integrate it with a mention detection model). It is clear that to date, no one has built a model that allows for overlapping features, that is fully integrated and that can be trained efficiently. This thesis provides such a model.

5.3 EDT Data Set and Evaluation

All the experiments described in this thesis are based on data from the Automatic Content Extraction (ACE) workshop sponsored by the National Institute of Standards and Technology (NIST). Specifically, I use the 2004 training and evaluation corpora annotated and released by the Linguistic Data Consortium (LDC). The training data contains 451 English documents, consisting of 220 broadcast news documents, 128 newswire documents, 37 documents translated from the Chinese Treebank, 58 documents translated from the Arabic Treebank and 8 files from the Fisher telephone conversations set. Overall, the data is drawn from a wide variety of sources, including: Agence France-Presse, New York Times, Associated Press Newswire, Zaobao, An-Nahar, Al-Hayat, Nile TV, Voice of America, Public Radio International, Cable News Network, American Broadcasting Corporation, National Broadcasting Company, China National Radio, China Television System, China Central TV, China Broadcasting System and Xinhua News Agency.

The evaluation criteria (loss functions) used for the EDT problem are quite complex. This complexity is, to some degree, necessary. If one only cares about entity mention detection (finding the mentions, but not performing coreference), then one can apply F-measure loss as before (Eq (4.3)). Moreover, if one has access to *correctly identified* entities, then one can score the coreference resolution alone by similar metrics; F-measure is a reasonable choice here as well, though other measures may be more appropriate (Luo, 2005).

The primary difficulty with evaluating *both* mention detection *and* coreference simultaneously is that one desires a loss function that both corresponds to how useful the system will be if deployed *and* is intuitively understandable. The metric used in the ACE competitions is the "ACE metric," defined formally by Doddington (2004a). This metric is "entity centric" in the sense that it operates on an entity-by-entity basis (rather than a mention-by-mention basis). At a high level, it operates as follows. One creates a bipartite graph. In one partition is the set of *reference* entities. In the other partition is the set of *system* entities. One attempts to find a matching between the reference entities and system entities. Each match has a score, essentially reflecting how "compatible" the entities are. The overall matching is computed to maximize this score.

The matching score is computed as follows. For a given *entity* discovered by the system, one computes an *entity value* and a set of *mention values* (one mention value for each mention of this entity). The *entity value* is based on (1) the type of the entity (Person, Organization, etc.) and (2) whether it has a correspondence on the reference. If (2) is not satisfied, a "false alarm" penalty is incurred. The *mention value* is computed the same way as the entity value, but based on mention types rather than entity types. Mentions lacking correspondences in the reference document incur both a false alarm penalty and a coreference penalty. The score for a given entity is the *product* of its entity value with the sum of the mention values.

5.4 Entity Mention Detection

Before applying Searn to the full EDT problem, I will apply it individually to the entity mention detection (EMD) and coreference resolution problems. To apply Searn to the entity mention detection (EMD) problem, one is required to specify two components: the search space (and corresponding actions) and the features. These two are inherently tied, since the features rely on the search space, but for the time being I ignore the issue of the feature functions and focus on the search.

5.4.1 Search Space and Actions

Like the sequence labeling case (Section 4.3), for the entity mention detection problem, I structure search in a left-to-right decoding framework: a hypothesis is a complete identification of the mentions for an initial segment of a document. For instance, on a document with N words, a hypothesis that ends at position $0 < n < N$ is essentially what you would get if you took the full structured output for this output and chopped it off at word n. In the example given in the introduction, one hypothesis might correspond to "Bill Clinton gave a" (which has no loss thus far), or to "Bill Clinton gave a" (which has made two errors).

A hypothesis is expanded through the application of the search actions. In this case, the search procedure first chooses the number of words it is going to consume (for instance, to form the mention "Bill Clinton," it would need to consume two words). Then, it decides on an entity type and a mention type (or it opts to call this chunk not an entity (NAE), corresponding to non-underlined words). This is the chunk-at-a-time search style described in Section 4.3.2. All these decisions are made simultaneously, and the given hypothesis is then scored. I restrict that a NAE may only consume a single word.

5.4.2 Optimal Policy

Computing an optimal policy step for the entity mention detection problem is roughly the same as the computation for the chunking problem (Section 4.1.3). However, the loss function for EMD is, in some ways, simpler than F_1 measure. Essentially, each *extraneous* entity incurs a weighted loss, while each *missed* entity incurs a weighted loss. These are simply combined linearly.

As in the chunking problem, the optimal policy for EMD is as follows. If we are at a position where the true output exactly specifies what to do next: do it (i.e., if we are at the beginning of an entity, or outside of one). Otherwise, we are at a position that is *internal* to a true entity; in this case, we should simply skip via "outs" until the end of the entity. While nearly correct, there is a small caveat to this policy. According to the ACE metric, one can miss an entity by a percentage of the total number of characters and still received credit. It would be straightforward to extend this optimal policy to take this complication into account, but I do not do so.

Computing the optimal approximation score for EMD is equally easy: it is simply the loss accumulated up to the given point (with, perhaps, an additional component if we wind up inside a true entity that we cannot create). Since the ACE score is additive over the output, we need not consider any future decisions for computing the optimal approximation score.

5.4.3 Feature Functions

All the features I consider are of the form *base-feature* × *decision-feature*, where base features are functions of the input and decisions are functions of the hypothesis. For instance, a base feature might be something like "the current chunk contains the word 'Clinton'" and a decision feature might be something like "the current chunk is a named person."

5.4.3.1 Base Features

For pedagogical purposes and to facilitate model comparisons, I have separated the base features into eleven classes: lexical, syntactic, pattern-based, semantic, knowledge-based, class-based, list-based, inference-based features and history-based features. I discuss with each of these in turn. Finally, I discuss how these base features are combined into *meta-features* that are actually used for prediction.

Lexical features. The class of lexical features contains simply computable features of single words. This includes: the number of words in the current chunk; the unigrams (words) contained in this chunk; the bigrams; the two character prefixes and suffixes; the word stem (according to the Porter stemmer (Porter, 1980)); the case of the word, computed by regular expressions like those given by (Bikel, Schwartz, and Weischedel, 1999); simple morphological features (number, person and tense when applicable); and, in the case of coreference, pairs of features between the current mention and an antecedent.

Syntactic features. The syntactic features are based on running the joint part-of-speech tagger and syntactic chunker described in Section 4.1.4 on the data. The syntactic features include unigrams and bigrams of part of speech as well as unigram chunk features. I do not use any parsing for this task.

Pattern-based features. I include a whole slew of features based on lexical and part of speech patterns surrounding the current word. These features include: eight hand-written patterns for identifying pleonastic "it" and "that" (as in "It is raining" or "It seems to be the case that …"); identification of pluralization features on the previous and next head nouns (this is intended to help make decisions about entity types); the previous and next content verb (also intended to help with entity type identification); the possessor or possessee in the case of simple possessive constructions ("The president 's speech" would yield a feature of "president" on the word "speech", and vice-versa; this is indented to be a of weak sub-categorization principle); a similar feature but applied to the previous and next content verbs (again to provide a weak sort of sub-categorization).

Semantic features. The semantic features used are drawn from WordNet (Fellbaum, 1998). These include: the two most common synsets from WordNet for the all words in the chunk; all hypernyms of those synsets. Finally, I include the synset and hypernym information of the preceding and following verbs, again to model a sort of sub-categorization principle.

Class-based features. The class-based features I employ are designed to get around the sparsity of data problem while simultaneously providing new information about word usage. The first class-based feature I use is based on word classes derived from the web corpus mentioned earlier and computed as described by (Ravichandran, Pantel, and Hovy, 2005). The second attempts to instill knowledge of collocations in the data; I use the technique described by (Dunning, 1993) to compute multi-word expressions and then mark words that are commonly used as such with a feature that expresses this fact.

List-based features. I have gathered a collection of about 40 lists of common places, organization, names, etc. These include the standard lists of names gathered from census data and baby name books, as well as standard gazetteer information listing countries, cities, islands, ports, provinces and states. I supplement these standard lists with lists of airport locations (gathered from the FAA) and company names (mined from the NASDAQ and NYSE web pages). I additionally include lists of semantically plural but syntactically singular words (e.g., "group") which were mined from a large corpus by looking for patterns such as ("members of the . . . "). Finally, I use a list of persons, organizations and locations that were identified at least 100 times in a large corpus by the BBN IdentiFinder named entity tagger (Bikel, Schwartz, and Weischedel, 1999).

Inference-based features. The inference-based features are computed by attempting to infer an underlying semantic property of a given mention. In particular, I attempt to identify gender and semantic number (e.g., "group" is semantically plural although it is syntactically singular). To do so, I created a corpus of example mentions labels with number and gender, respectively. This data set was automatically extracted from our EDT data set by looking for words that corefer with pronouns for which I know the number or gender. For instance, a mention that corefers with "she" is known to be singular and female, while a mention that corefers with "they" is known to be plural. In about 5% of the cases, this was ambiguous – these cases were thrown out. I then used essentially the same features as described above to build a maximum entropy model for predicting number and gender. Additionally, I use several pre-existing classifiers as features. These are simple maximum entropy Markov models trained off of the MUC data (Grishman and Sundheim, 1995; Chinchor, 1997).

History-based features. Finally, I include features having to do with long-range dependencies between words. For instance, if at the beginning of the document we tagged the word "Arafat" as a person's name (perhaps because it followed "Mr." or "Palestinian leader"), and later in the document we again see the word "Arafat," we should be more likely to call this a person's name, again. Such features have previously been explored in the context of information extraction from meeting announcements using conditional random fields augmented with long-range links (Sutton and McCallum, 2004), but the

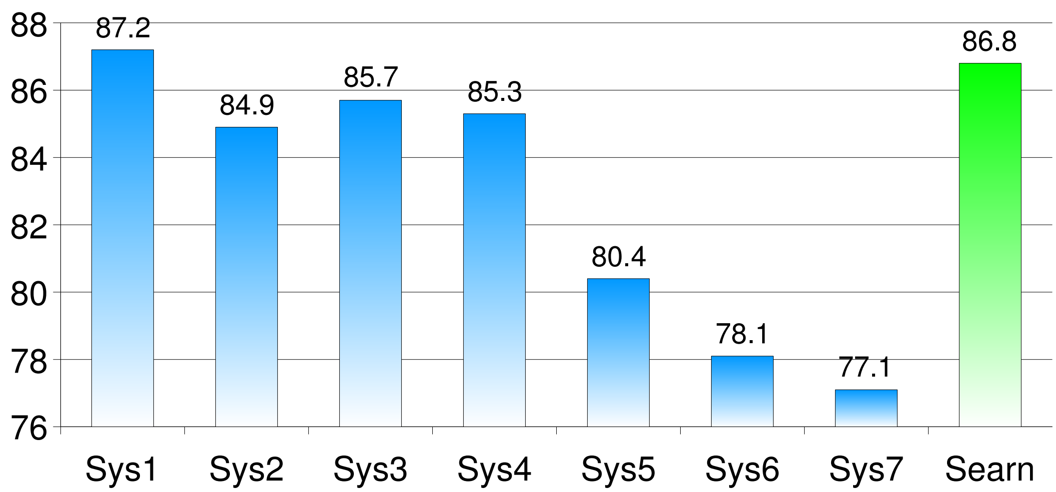


Figure 5.3: ACE scores on the mention detection task for all ACE 2004 systems as well as my SEARN-based system.

SEARN framework makes no Markov assumption, so there is no extra effort required to include such features.

5.4.3.2 Decision Features

The decision features are divided into two classes: simple decision features and boundary decision features.

Simple. The simple decision features include: is this chunk tagged as an entity; what is its entity type; what is its entity subtype; what is its mention type; what is its entity type/mention type pair.

Boundary. The boundary decision features include: the second and third order Markov features over entity type, entity subtype and mention type; features appearing at the previous (and next) words within a window of three; the words that appear and the previous and next mention boundaries, specified also by entity type, entity subtype and mention type.

5.4.4 Experimental Results

In Figure 5.3, I show the ACE scores of the top four systems that competed at the ACE 2004 evaluation workshop (Sys1 through Sys7) as well as the scores for the SEARN-based system. The SEARN system achieves a score of 86.8, roughly halfway between the first and second best ACE 2004 system (with scores of 87.2 and 85.7, respectively). It is worth noting that the two best performing ACE 2004 systems were trained on extra in-house data, thus rendering absolute comparisons difficult.

5.4.5 Error Analysis

One advantage to the SEARN framework is that error analysis is comparatively easy. One can simply observe, for each individual decision, which errors are most common. Or, perhaps more usefully, which errors lead to the greatest increase in loss. As it turns out, the majority of the loss in the current system comes from mis-segmentation, followed by entity type mis-identifications and lastly mention type mis-identifications. Roughly 78% of the loss suffered by the EMD system is due to segmentation errors, followed by 14% for entity type mis-identifications and 8% for mention type.

A common segmentation error is the following. In the sentence beginning "The United Nations Relief and Works Agency for Palestinian Refugees (UNRWA) made an urgent appeal today," the correct segmentation is to identify that the nine word phrase beginning with "United" and ending with "Refugees" is a single entity of type organization. The current model misidentifies this as three separate entities: "United Nations Relief" (an organization), "Works Agency" (an organization) and "Palestinian Refugees" (a person). Errors such as this are common, especially with very long entity names. It seems reasonable that one could add specific conjunction features to enable the model to correctly identify "United Nations Relief and Works Agency" as a single organization, but adding in "for Palestinian Refugees" seems difficult. This is especially difficult because the following abbreviation ("UNRWA") does not include any letters for "Palestinian Refugees", which implies that performing coreference simultaneously is unlikely to remedy this error.

With respect to missed mentions, the model currently errs most frequently on nominals. For instance, a common missed example is the organization nominal mention "auto-maker" in the segment "... Goldman Sachs downgraded the German auto-maker after ..." For the most part, if nominal mentions are observed in the training data, the model will correctly learn to spot them. However, such strings are quite productive and many new examples are seen regularly. Improved performance on this task may require better clustering technology to identify when new nominal mentions are observed.

5.5 Coreference Resolution

For the pure coreference resolution problem, the (unrealistic) assumption is that all *mentions* have been correctly identified and all one needs to do is group them into coreference classes. In this section, I explore the application of SEARN to this problem.

5.5.1 Search Space and Actions

The search procedure I use for the coreference resolution task is, like the mention detection search procedure in Section 5.4.1, structured in a left-to-right fashion. I assume that we are given all the mentions already tagged and that they have been assigned the correct mention types and entity types (with the small exception that the entity type for pronouns is not given: this would make the task too simple). Like the mention detection case, a hypothesis is formed by assigning coreference chains to the first k mentions. A given hypothesis is extended by assigning the $(k+1)$st mention to a coreference chain, be it a new chain or one that appears before.

One significant issue that arises in the context of assigning a hypothesis to a coreference chain is how to compute features over that chain. As I discuss in Section 5.5.3, the majority of our coreference-specific features are over *pairs* of chunks: the proposed new mention and an antecedent. However, since a proposed mention can have well more than one antecedent, one is left with a decision about how to combine this information.

The first, most obvious solution, is to essentially do nothing: simply compute the features over all pairs and add them up as usual. This method, however, intuitively has the potential for over-counting the effects of large chains. To compensate for this, one might advocating the use of an *average link* computation, where the score for a coreference chain is computed by averaging over its elements. One might also consider a *max link* or *min link* scenario, where one of the extrema is chosen as the value. Other research has suggested that a simple *last link*, where a mention is simply matched against the most recent mention in a chain might be appropriate, while *first link* might also be appropriate because the first mention of an entity tends to carry the most information.

In addition to these standard linkages, I also consider an *intelligent link* scenario, where the method of computing the link structure depends on the *mention type*. The intelligent link is computed as follow, based on the mention type of the current mention, m:

If m =**NAM then:** match *first* on NAM elements in the chain; if there are none, match against the *last* NOM element; otherwise, use *max link*.

If m =**NOM then:** match against the *max* NOM in the chain; otherwise, match against the *last* NAM; otherwise, use *max link*.

If m =**PRO then:** use *average link* across all PRO or NAM; if there are none, use *max link*.

The construction of this methodology was guided by intuition (for instance, matching names against names is easy, and the first name tends to be the most complete) and subsequently tuned by experimentation on the development data. One might consider *learning* the best link method, and this may result in better performance, but I have not explored this option to date. The initial results I present are based on using intelligent link, but I also compare the different linkage types explicitly (see Section 5.5.4).

5.5.2 Optimal Policy

Computing an optimal policy step for coreference resolution is significantly more difficult than for entity mention detection (or any of the other problems discussed thus far). To see why, consider Figure 5.4 (easier to view in color). In this figure, I have drawn two versions of a document: one with the true coreference links (on the top) and a partial hypothesis (on the bottom). Circles correspond to mentions and their coloring/labeling corresponds to coreference decisions. For example, the first three mentions are linked together in the true output ("A"), as is the 7th mention. In the hypothesis output, an error is made with the second mention (which was said to create a new entity, when in fact it should have been linked to the first) and the fourth mention (as well as others).

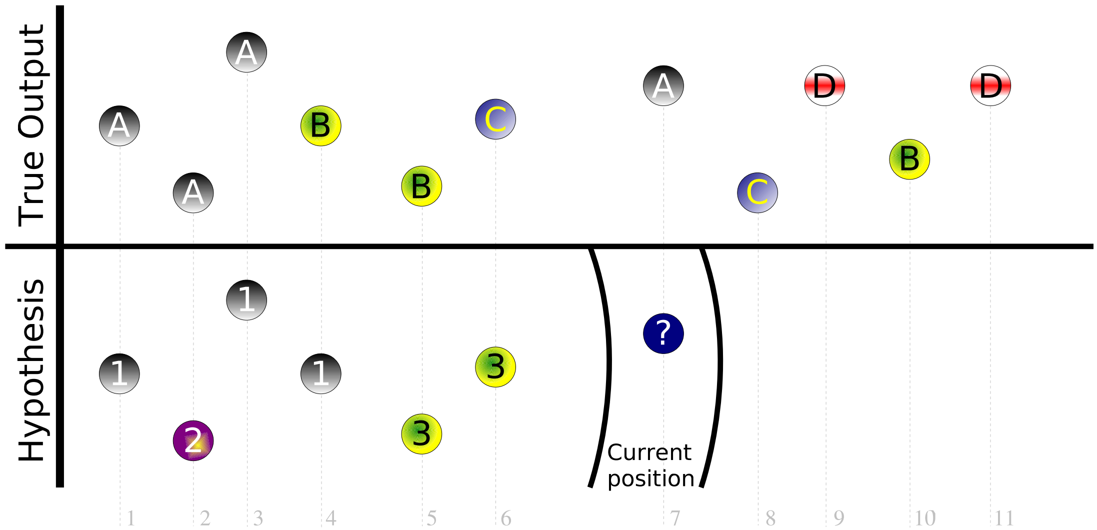


Figure 5.4: Running example for the computation of the optimal policy step for the coreference task.

Consider now attempting to make an optimal decision for the 7th mention: the "current position." We must decide that this is coreferent with cluster 1, cluster 2, cluster 3 or a new cluster. It is clear that it will never be optimal to say that it is coreferent with cluster 3. Moreover, it is actually the case that we will never say that this is a *new entity* either: this is because doing so will never be better than saying it is coreferent with cluster 2. However, under different circumstances, the other two options are both reasonable:

Cluster 1: If mention 4 (the "incorrect" entity in cluster 1) is of low weight, then attaching the current mention to cluster 1 will be the right thing to do (the error incurred by the previous mistake will not count against us significantly). Alternatively, if mention 4 is of very *high* weight, and there are many high-weight "B" mentions later in the document, then it may be worthwhile to link the new mention into cluster 1 with the goal of then linking in all the remaining "B" clusters to enable cluster "1" to match to "B" (instead of "A").

Cluster 2: Suppose mentions 1, 3 and 4 all have low weight. Further, suppose that cluster "A" has 100 more mentions laster in the document (and not pictured in the Figure). In this case, it will be preferable to "give up" on the current entity (cluster 1) and begin (nearly) from scratch with cluster 2.

This qualitative analysis shows that choosing an optimal policy step is intuitively difficult. The difficulty lies in the fact that the ACE scoring function is a combinatorial optimization problem itself, based on bipartite matching. This makes the choice of the best-next-action difficult because it strongly depends on *future* decisions. In fact, there exist exceptionally difficult cases, for which it appears necessary to compute an exponential number of possibilities.[2] Fortunately, there are *approximate* loss functions that behave very similarly to the ACE metric that *are* efficiently optimizable.

The approximate loss function I use for solving the coreference problem is based on *removing* the bipartite matching step from the ACE metric. That is, instead of using matching to decide which hypothesis entities link to which true entities, I allow the coreference module to make this decision itself. This small change makes it straightforward to compute scores for each of the two options for the current position in Figure 5.4.

The computation proceeds as follows. We assume that however we link m_7, it will end up matching with "A" (i.e., with its true cluster). Now the question simply becomes: of all the links we consider (in the example, linking to "1" or linking to "2"), which will give the higher score *for this entity.* This is also simple to compute. For linking to "1", we will get credit for mentions 1, 3 and 7 but will incur a false-alarm penalty for mention 4 and a miss penalty for mention 2. For linking to 2, we will get credit for mentions 2 and 7 but will incur miss penalties for mentions 1 and 3. Each of these credits/penalties will have a weight which we can compute based only on the mentions themselves, and we simply sum. This leads to two scores, which can be compared. The greater score is the choice made by this (approximately) optimal policy. In general, this computation will need to be made once for each *hypothesis* class that intersects with the true class of the current position, and each of these computations will involve as many mentions as are in the union of these sets. In the worst case, this will be $\mathcal{O}(m^2)$, where m is the current position, but in practice it is quite efficient.

5.5.3 Feature Functions

As in the EMD case, features are broken into *base-features* and *decision-features.* These are described separately. Many are replicated from the EDT task and simply applied in a pair-wise fashion over elements of the chains. I additionally add three new classes of base-features: count-based, knowledge-based and string match features.

5.5.3.1 Base Features

The base features include most of those described in Section 5.4.3.1, but applied over pairs. For instance, with the base lexical feature (which looks at word identity), we would extend this to look at, for instance, the word pair "Clinton" and "president," a feature which would one would expect to have fairly high weight when paired with the "is coreferent" decision feature.

Lexical features. These features are identical to the EMD case.

Syntactic features. These features are identical to the EMD case.

[2] This leads me to believe that this problem may be formally hard; unfortunately, at this time, I do not know of a reduction from a problem in NP. This is an open question, but perhaps an unimportant one.

Pattern-based features. In addition to the features described in the EMD model, I include a list of part of speech and word sequence patterns that match up to four words between nearby mentions that are either highly indicative of coreference (e.g., "of," "said," "am" ", a") or highly indicative of non-coreference (e.g., "'s," "and," "in the," "and the"). This last set was generated by looking at intervening strings and finding the top twenty that had maximal mutual information with the class (coreferent or not coreferent) across the training data.

Count-based features. The count-based features apply only to the coreference task and attempt to capture regularities in the size and distribution of coreference chains. These include: the total number of entities detected thus far; the total number of mentions; the entity to mention ratio; the entity to word ratio; the mention to word ratio; the size of the hypothesized entity chain; the ratio of the number of mentions in the current entity chain to the total number of mentions; the number of intervening mentions between the current mention and the last one in our chain; the number of intervening mentions of the same type; the number of intervening sentence breaks; the Hobbs distance computed over syntactic chunks; and the "decayed density" of the hypothesized entity, which is computed as $\sum_{m=e} 0.5^{d(m)} / \sum_{m} 0.5^{d(m)}$, where m ranges over all previous mentions (constrained in the numerator to be in the same coreference chain as our mention) and $d(m)$ is the number of entities away this mention is. This final feature is intended to capture the notion that some entities are referred to consistently across a document, while others are mentioned only for short segments, but it is relatively rare for an entity to be mentioned once at the beginning and then ignored again until the end.

Semantic features. In addition to the semantic features described for EMD, I also consider the distance in the WordNet graph between pairs of head words (defined to be the final word in the mention name) and whether one is a part of the other. Finally, I include the synset and hypernym information of the preceding and following verbs, again to model a sort of sub-categorization principle.

Knowledge-based features. Based on the hypothesis that many name to nominal coreference chains are best understood in terms of background knowledge (for instance, that "George W. Bush" is the "President"), I have attempted to take advantage of recent techniques from large scale data mining to extract lists of such pairs. In particular, I use the name/instance lists described by (Fleischman, Hovy, and Echihabi, 2003) and available on Fleischman's web page to generate features between names and nominals (this list contains ... pairs mined from ... words of news data). Since this data set tends to focus mostly on person instances from news, I have additionally used similar data mined from a 138 gigabyte web corpus, for which more general "ISA" relations were mined (Ravichandran, Pantel, and Hovy, 2005).

Class-based features. I use identical features to the EMD model.

List-based features. In addition to those features described in the EMD model, for coreference, we look for word pairs that appear on the same list but are not identical (for instance, “Russia” and “England” appearing on the “country” list but not being identical hints that they are different entities). Finally, I look for pairs where one element in the pair is the head word from one mention and the other element in the pair is a list. This is intended to capture the notion that a word that appears on the “country list” is often coreferent with the word “country.”

Inference-based features. I use identical features to the EMD case.

String match features. I use the standard string match features that are described in every other coreference paper. These are: string match; substring match; string overlap; pronoun match; and normalized edit distance. In addition, we also use a string nationality match, which matches, for instance “Israel” and “Israeli,” “Russia” and “Russian,” “England” and “English,” but not “Netherlands” and “Dutch.” This is done by checking for common suffixes on nationalities and matching the first half of the of the words based on exact match. I additionally use a linguistically-motivated string edit distance, where the replacement costs are lower for vowels and other easily confusable characters. I also use the Jaro distance as an additional string distance metric (Jaro, 1989; Jaro, 1995). Finally, I attempt to match acronyms by looking at initial letters from the words in long chunks.

5.5.3.2 Decision Features

The coreference decision features include the following: is this entity the start of a chain or continuing an existing chain; what is the entity type of this started (or continued) chain; what is the entity subtype of this started (or continued) chain; what is the mention type of this started chain; what is the mention type of this continued chain and the mention type of the most recent antecedent.

5.5.4 Experimental Results

In Figure 5.5, I plot the ACE scores of the three ACE 2004 systems that competed in the coreference-only subtask, one baseline system that operates by matching mention heads (the final words of each mention), and my Searn-based system. My system performs significantly better than the baseline and Sys5, slightly better than Sys1 and worse than Sys2. However, it should be noted that Sys1 and Sys2 annotated extra data, so explicit comparisons are difficult.

As stated in the previous section, the coreference-only task with intelligent link achieves an ACE score of 89.1. The next best score is with min link (88.7) followed by average link with a score of 88.1. There is then a rather large drop with max link to 86.2, followed by another drop for last link to 83.5 and first link performs the poorest, scoring 81.5. These results are depicted in Figure 5.6.

5.5.5 Error Analysis

Error analysis for coreference resolution is a difficult endeavor, primarily because it is challenging to define classes of errors that can be easily counted. One way of looking at

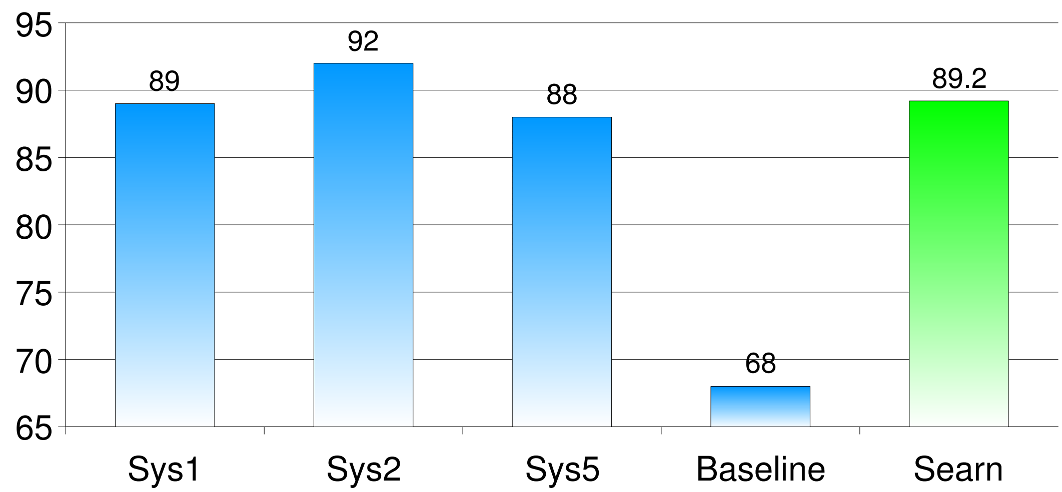


Figure 5.5: ACE scores on the coreference subtask for the three ACE 2004 systems that competed in this subtask, one baseline, and the SEARN-based system.

the coreference errors is to first compute the matching, as defined by the ACE evaluation metric (Section 5.3) and then inspect what types of entities are not in their correct clusters.

According to this evaluation, from the perspective of which entity types cost the most, roughly 20% of the loss came from entities each of type person, organization and GPE. 14% can from weapons and 10% from facilities. The remaining loss was split roughly equally between vehicles and locations. One can also compute these numbers by count, rather than by loss (recall that the ACE metric favors different entity types). By count, 25% of the errors were on each of vehicles and weapons, 18% on people and the remainining error split roughly evenly between the other classes. As this shows, the fact that I have focused away from vehicles and weapons because they are relatively uncommon and have low weight has significantly affected the performance of the system on these types.

One can also evaluate the performance on the basis of the sizes of the entity sets. That is: when errors are made, are they made on singleton entities or large entities. Somewhat surprisingly, there is not a strong bias. 17% of the errors is on singleton entities, 18% on doubletons, 22% on entites with 3-4 mentions, 23% on entities with 5-8 mentions and 20% on entities with more than eight mentions.

An alternative way of dividing up errors is by treating the coreference problem as a binary classification problem. This enables us to evaluate the success of the model on incremental decisions. In Table 5.3, I show the percentage of errors broken down by mention type. For instance, the upper right cell in any of the four tables is the percentage of errors that were made on a name-to-name link (where the reference was a name and the proposed antecedent was also a name). The left tables show the raw percentages; the right tables normalize these numbers by frequency of the true mention types (that is, pronouns are comparatively rare, so on a percentage basis, the system makes more pronoun errors

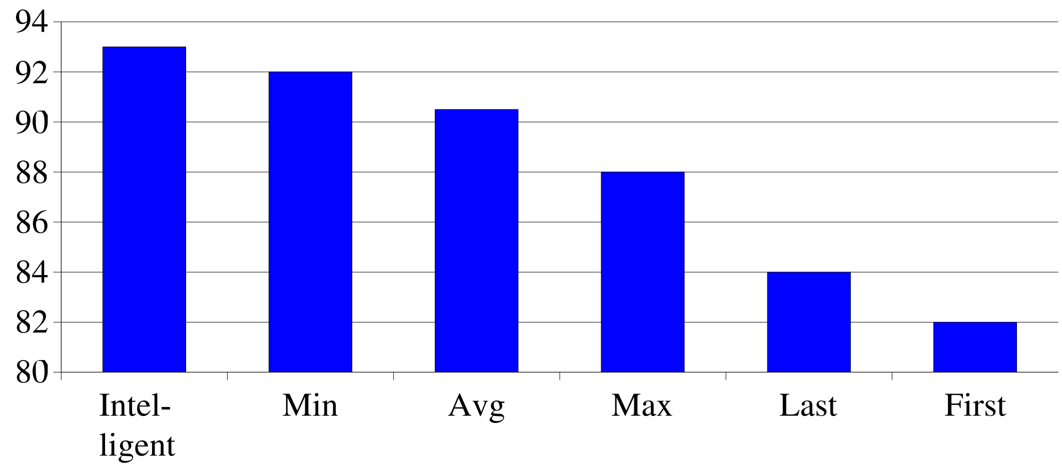


Figure 5.6: Comparison of different linkage types on the coreference task.

than is obvious from the left-most tables). The top tables show the performance of the system without the knowledge-based features; the bottom tables show the performance with these features.

As we can see, the majority of the errors without the knowledge-based features are name-to-nominal (10%), nominal-to-name (8%) and pronoun-to-anything. Once the knowledge-based features are added, the name-to-nominal errors drop to only 8% and the nominal-to-name errors drop to 6.6%. At this point, the only significant error source is with pronouns.

5.6 Joint Detection and Coreference

Merging the model described for entity mention detection (Section 5.4) and the model for coreference resolution (Section 5.5) into a single joint model is, in fact, a rather trivial change. The triviality of this change is one of the significant advantages of working withing the Searn framework.

5.6.1 Search Space and Actions

The search space and search operations for the joint EDT model simply merges those for the corresponding halves of the problem. Again, we decode in a left-to-right manner, where a hypothesis represents a complete decision for an initial segment of a document. As in the EMD model, a hypothesis is extended by first choosing a number of words, then choosing an entity type and mention type. The difference in the joint model is that after the types are selected, we also select (as in the coreference model) the chain to which we wish to link this mention (so long as it is not "not an entity").

WITHOUT KNOWLEDGE-BASED FEATURES

Error Percentage				Normalized by Frequency			
Ante \ Ref	**NAM**	**NOM**	**PRO**	**Ante \ Ref**	**NAM**	**NOM**	**PRO**
NAM	6.2%	11.6%	12.5%	**NAM**	5.8%	7.9%	16.3%
NOM	15.2%	9.7%	12.8%	**NOM**	10.4%	4.1%	11.9%
PRO	10.9%	7.9%	13.3%	**PRO**	16.1%	8.9%	18.7%

WITH KNOWLEDGE-BASED FEATURES

Error Percentage				Normalized by Frequency			
Ante \ Ref	**NAM**	**NOM**	**PRO**	**Ante \ Ref**	**NAM**	**NOM**	**PRO**
NAM	6.6%	9.9%	13.4%	**NAM**	6.1%	6.6%	17.0%
NOM	12.0%	10.3%	13.7%	**NOM**	8.0%	4.3%	12.5%
PRO	11.6%	8.4%	14.1%	**PRO**	16.8%	9.3%	19.5%

Table 5.3: Coreference errors evaluated on a mention-type basis.

5.6.2 Optimal Policy

As discussed in Section 5.4.2, the optimal policy for the mention detection problem is straightforward to compute. Unfortunately, as discussed in Section 5.5.2, the optimal policy for the *coreference* problem is very difficult to compute. Thus, for coreference, an approximation was advocated. Based on this observation, we use a similar approximation for computing the optimal policy for the full EDT task. This is derived by simply combining the optimal policies for the two subtasks. That is, first we take a step according to the mention detection optimal policy, then, so long as this step created a mention, we take a step according to the coreference optimal policy.

The only complication is that when taking a coreference step, it is not necessarily the case that previous mentions have been correctly identified. For instance, consider Figure 5.4. When the policy arrives at mention 7, it may be that, for instance, *all* of the preceding "A" mentions were missed. Or, alternative, there many be *extraneous* mentions in the hypothesis that do not correspond to any mention in the true output.

It is straightforward to derive the optimal decision, even in light of these two complications. First, notice that it is *never* optimal to decide a mention is coreferent with an entity comprised solely of spurious mentions. Next, note that any *missing* mentions may be ignored: they will neither help nor hinder the score. Thus, we may make coreference decisions exactly as in Section 5.5.2, where any missed mentions are ignored in the score computation and any wholly spurious entities are also ignored.

5.6.3 Experimental Results

The results on the full EDT task are shown in Figure 5.7, again compared against all competing ACE 2004 systems. As one can see from these results, our EDT model is on par with the first and second best systems (again, Sys1 annotated extra data). The differences

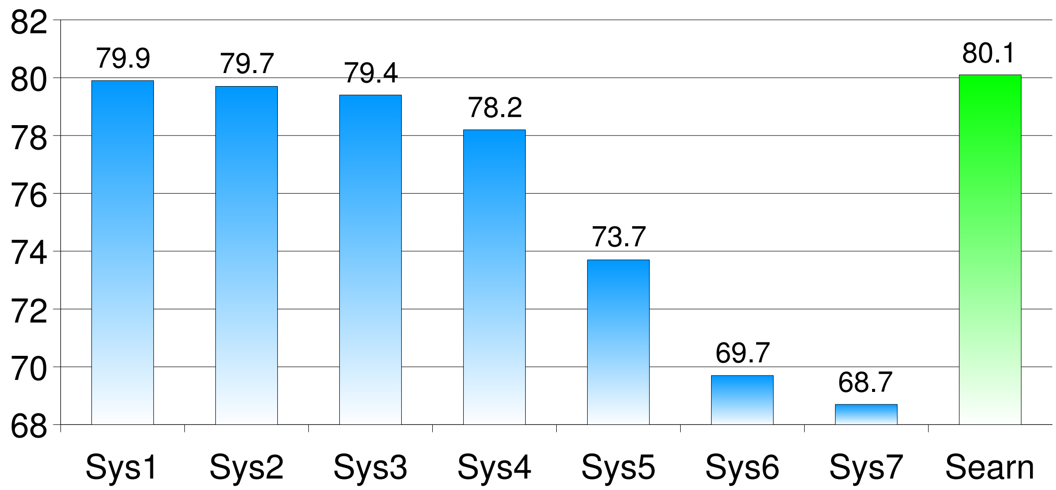


Figure 5.7: ACE scores on the full EDT task for all ACE 2004 system, and my SEARN-based joint system.

between the SEARN-based system and the top three other systems are not statistically significant at the 95% level, though the difference to system S4 *is* statistically significant.

In Figure 5.8, I show the same results as in Figure 5.7, but with an additional column for running SEARN in a *pipelined* manner. That is, first mention detection is run; then coreference is run on top. Both of these sytems are trained *separately* (and the coreference module is trained on the output of the mention detection module). As we can see from these results, but not running jointly, the SEARN-based system falls to a solid fifth place, rather than being tied for first.

To further show the usefulness of the joint decoding, I also consider using the joint system to perform the simple mention detection task. That is, I run the full joint EDT system, but then throw out the coreference links. This enables us to determine if coreference information is useful for the plain mention detection task. When run in pipelined mode (i.e., mention detection only), the SEARN-based system achieves a score of 86.8, as shown in Figure 5.3. When run in pipelined mode, SEARN achieves a score of only 87.1, tightly closing the gap to the best mention detection system (Sys1, with a score of 87.2). Thus, running jointly makes the difference between a system that is (roughly) tied for first place among competing systems and one that is strongly in fifth place. This difference is largely due to two factors. First, some decisions (such as entity type) are easy to make in a joint system. Second, and perhaps more importantly, the coreference module is trained with respect to the errors the mention detection system *will make* when applied. This allows the two components to trade off errors against one another.

5.7 Discussion and Conclusions

In this chapter, I applied the SEARN framework to the entity detection and tracking task. SEARN is an excellent choice for this problem, due to the fact that many relevant features

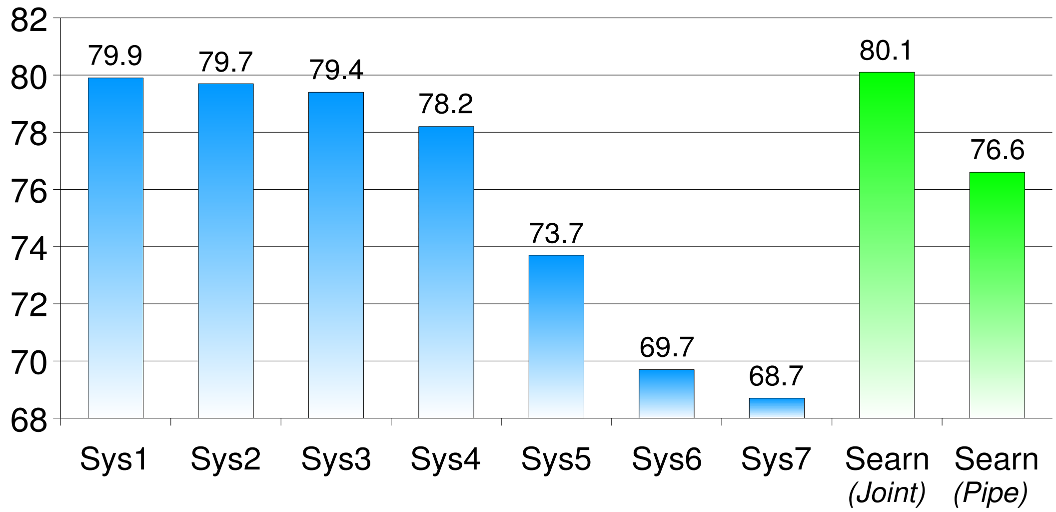


Figure 5.8: ACE scores on the full EDT task for all ACE 2004 system, my SEARN-based joint system and a pipeline version of my SEARN-based system.

for the coreference task (and even for the mention detection task) are highly non-local. This non-locality makes models like Markov networks intractable, and SEARN provides an excellent framework for tackling this problem. I have introduced a large set of new, useful features for this task, most specifically the use of knowledge-based features for helping with the name-to-nominal problem, which has led to a substantial improvement in performance. I have shown that performing joint learning for mention detection and coreference results in a better performing model that pipelined learning. I have also provided a comparison of the contributions of our various feature classes and compared different linkage types for coreference chains. In the process, I have developed an efficient model that is competitive with the best ACE systems.

Despite these successes, the learned model is not perfect: the largest source of error is with pronouns. This is masked by the fact that the ACE metric weights pronouns low, but a solution to the EDT problem should handle pronouns well. Future work involves the exploration more complex features for resolving pronouns, and to incorporating these features into the current model.

Chapter 6

Multidocument Summarization

Multidocument summarization is the task of creating a summary out of a collection of documents on a focused topic. In query-focused summarization, this topic is given explicitly in the form of a user's query. The dominant approach to the multidocument summarization problem is sentence extraction: a summary is created by greedily extracting sentences from the document collection until a pre-defined word limit is reached. Teufel and Moens (1997) and Lin and Hovy (2002) describe representative examples. Current winning systems in the Document Understanding Conference (which has focused recently on 250 word summaries) are based nearly entirely on sentence extraction techniques, with a minor amount of pre- and post-processing (Dang, 2005; Daumé III and Marcu, 2005b). Unfortunately, the granularity of sentence extraction systems is often inadequate for producing short summaries of large document sets. Recent work in sentence compression (Knight and Marcu, 2002; Riezler et al., 2003; Turner and Charniak, 2005; McDonald, 2006a) and document compression (Daumé III and Marcu, 2002) attempts to take small steps beyond sentence extraction. Compression models can be seen as techniques for extracting sentences then dropping extraneous information. They are more powerful than simple sentence extraction systems, while remaining trainable and tractable.

One significant difficulty with these sentence compression models is that their training hinges on the existence of $\langle$ sentence, compression $\rangle$ pairs, where the compression is obtained from the sentence by only dropping words and phrases (the work of Turner and Charniak (2005) is an exception). Obtaining such data is quite challenging, especially in arbitrary domains. Due to this, most of the previous work on the sentence compression problem evaluates on a corpus of product reviews from Ziff-Davis, which was originally collected by Knight and Marcu (2002). The difficulty of obtaining explicit training data for training sentence compression (or document compression) models leads us to desire training techniques that *do not* require such data.

6.1 Vine-Growth Model

In this section, I introduce the *vine-growth* model for multidocument summarization. For now, I ignore the issue of evaluation criteria (see Section 6.3) and training (see Section 6.4). Like most previous work on sentence compression, the vine-growth model makes use of the syntactic structure of the sentences to be compressed. However, *unlike* most previous work (see McDonald (2006a) for an exception), the vine-growth method

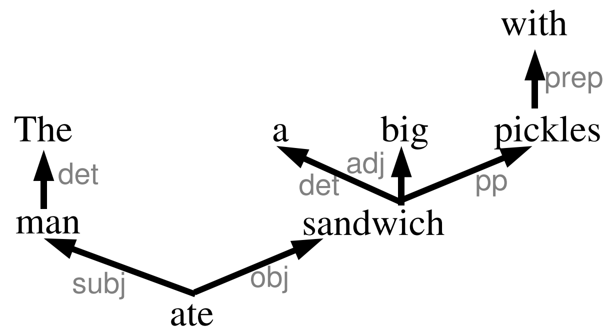


Figure 6.1: The dependency tree for the sentence “The man ate a sandwich with pickles”.

uses *dependency* structures instead of *constituent* structures. An example dependency tree for the sentence “The man ate a sandwich with pickles” is shown in Figure 6.1.[1]

There are several advantages to working with dependency trees rather than constituent trees. Most importantly, dependency trees allow for easy lexicalization, which seems important in a summarization task. Moreover, the dependency tree structure makes clear the relationship between the words in a sentence; this is not the case for constituent trees. Similar arguments for dependency structures in lieu of constituent structures have been made for machine translation systems as well (Quirk, Menezes, and Cherry, 2005).

The vine-growth model produces compressions of documents by simultaneously selecting and compressing sentences. The assumption made in the compression model is that if a word w is to be included in the summary, then all words that w depends on should also be included. In the language of trees, all ancestors of w should be included. For instance, if one decides that the word “pickles” should be included in a summary of the sentence from Figure 6.1, then one is forced to also include the words “ate” and “sandwich” in the summary. This is, intuitively, a reasonable requirement, though see Section 6.7 for an analysis of exceptions. Though not strictly necessary, in order to produce more grammatical summaries, the model requires that all closed class children (determiners, prepositions, modals and punctuation) of summary words are also included (e.g., we could not include “man” without also including “the” and we could not include “pickles” without also including “with”).

Under this model, the set of valid summaries of the sentence from Figure 6.1 are:

1. ate
2. The man ate
3. ate a sandwich
4. ate a sandwich with pickles

[1]For all experiments described in this chapter, I have used an in-house implementation of Collins' Model II parser (Collins, 2003) and hand-written head-finding rules also due to Collins. The implementation of the parser is due to Radu Soricut.

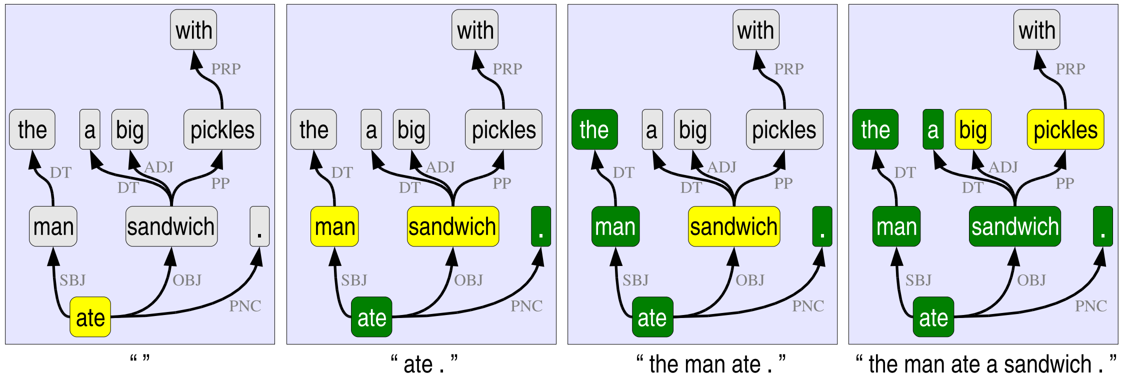


Figure 6.2: An example of the creation of a summary under the vine-growth model.

5. The man ate a sandwich
6. The man ate a sandwich with pickles

Half of these possible summaries are grammatical (including the “non-summary” that includes the entire sentence). Two (“The man ate” and “The man ate a sandwich”) are reasonable summaries. The goal of learning will be to find these reasonable summaries on the basis of training data.

6.2 Search Space and Actions

The Searn algorithm I employ for implementing the vine-growth model is based on incrementally *growing* summaries. In essence, beginning with an empty summary, the algorithm incrementally adds words to the summary, either by beginning a new sentence or growing existing sentences.

More formally, the algorithm maintains a set of *summary nodes* and a set of *frontier nodes.* The set of summary nodes is initialized to be empty, and the set of frontier nodes is initialized to the set of the head words of each sentence. In each step of search, a single frontier node is selected and added to the summary (together with any closed class children). All other children of this newly-added node are then added to the frontier.

To see more clearly how the vine-growth model functions, consider Figure 6.2. This figure shows a four step process for creating the summary “the man ate a sandwich .” from the original document sentence “the man ate a big sandwich with pickles .” In this figure, frontier nodes are colored yellow and summary nodes are colored green; all other nodes are gray. The algorithm proceeds as follows:

1. In the initialization, the root word (“ate”) is marked as a frontier node.
2. In the first step, the frontier node “ate” is added to the summary and colored green. Since the period is punctuation and is a child of “ate” it is also added to the summary. The remaining children—“man” and “sandwich”—are added to the frontier.

Number:	d324e
Title:	Argentine British relations post Falkland War
Narrative:	How have relations between Argentina and Great Britain developed since the 1982 war over the Falkland Islands? Have diplomatic, economic, and military relations been restored? Do differences remain over the status of the Falkland Islands?

Figure 6.3: An example query from the DUC 2005 summarization corpus.

3. In the second step, the word "man" is added to the summary and its closed-class child ("the") is also added. It has no other children, so no new words are added to the frontier.

4. Finally, the frontier node "sandwich" is selected and added to the summary along with its closed-class child ("a"). Its remaining children—"big" and "pickles"—are added to the frontier.

Note that steps 3 and 4 are interchangeable: from the perspective of the summary produced, these steps could occur in any order.

When there is more than one sentence in the source documents, the search proceeds asynchronously across all sentences. When the sentences are laid out adjacently, the end summary is obtained by taking all the green summary nodes once a pre-defined word limit has been reached. This final summary is a collection of subtrees grown off a sequence of underlying trees: hence the name "vine-growth."

6.3 Data and Evaluation Criteria

For data, I use the DUC 2005 data set (Dang, 2005). This consists of 50 document collections of 25 documents each (average document length is about 700 words); each document collection includes a human-written query. An example query is shown in Figure 6.3. Each document collection additionally has five human-written "reference" summaries (250 words long, each) that serve as the gold standard. The "best" human summary (in the sense that it is most like the others) for the query shown in Figure 6.3 is shown in Figure 6.4.

In the official DUC evaluations, all 50 collections are "test data." However, since the DUC 2005 task is significantly different from previous DUC tasks, it is not a good source of training data. Therefore, we report results based on 10-fold cross validation. We train on 45 collections and test on the remaining 5. Rotating the test set gives us system summaries on all 50 collections.[2]

Evaluation is a notoriously difficult problem for document summarization. The current popular choice for metric is Rouge (Lin and Hovy, 2003), which (roughly speaking)

[2]Note that running the model in a cross validation setting is perhaps overly optimistic: the results are not directly comparable against other DUC systems, since these did not have access to any of this "training data."

Since the 1982 war over the Falkland Islands, relations between Argentina and Britain have steadily improved. Full diplomatic relations resumed in 1990, and senior British and Argentine officials have visited in London and Buenos Aires. In 1994, Prince Andrew made the first royal visit to Argentina since the war's end. British-Argentine economic relations have slowly recovered. Argentina lifted financial and trade restrictions on British imports and has encouraged British investment, including in its soon-to-be privatized nuclear industry. Representatives from both countries have discussed the possibility of jointly developing offshore gas and oil fields bordering the Falklands' waters. Also, British and Argentine scientists have begun joint research in fish conservation. Issues remain concerning military ties between the two countries. Britain still has not lifted the arms embargo it imposed against Argentina following the 1982 war, and refuses to do so. However, relations have progressed steadily to the point where Britain and Argentina cooperated militarily during the 1991 Gulf War. Britain and Argentina have seen rapid improvement in relations since President Carlos Menem took office and adopted pro-western foreign policies. However, one issue continues to divide the two. Argentina refuses to surrender sovereignty over the Falkland Islands, despite losing the 1982 war. Consequently, the Falklands' fishing and oil resources continue to be a source of friction. Although they made a one-year arrangement to share fish resources, Argentina is tying any long-term agreement to British concessions to share oil development and lift the arms embargo. Two things Britain is not prepared to do.

Figure 6.4: An example summary from the DUC 2005 summarization corpus.

computes n-gram overlap between a system summary and a set of human written summaries. In various experiments, Rouge has been seen to correspond with human judgment of summary quality. In the experiments described in this chapter, I use the "Rouge 2" metric, which uses evenly weighted bigram scores. Formally, let H be the multiset of bigrams in a hypothesis summary and let T be the multiset of bigrams in union of all reference summaries. Then, Rouge 2 is computed as in Eq (6.1), where the sums are over all possible word pairs and $H_{w,w'}$ ($T_{w,w'}$) is the number of times the bigram ww' occurs in the system hypothesis (reference summaries).

$$\text{Rouge}_2(T,H) = \frac{\sum_{(w,w')} \max\{T_{w,w'}, H_{w,w'}\}}{\sum_{(w,w')} T_{w,w'}} \tag{6.1}$$

The choice of which version of Rouge to use is relatively insubstantial: most Rouge metrics would be equally easy to optimize. (In fact, the recent Basic Element-based Rouge (Hovy et al., 2006) would be *easier* to optimize, due to its reliance on dependency pairs rather than raw bigrams. Nevertheless, for simplicity, I work exclusively with Rouge 2 for the remainder of this chapter.)

6.4 Optimal Policy

Computing the optimal policy under the Rouge metric for the vine-growth model is intractable. The intractability stems from the model constraint that a word can only be added to a summary after its parent is added. I therefore use an approximately optimal policy. In order to approximate the cost of a given partial summary, I *search* for the best possible policy. That is, if our goal is a 100 word summary and we have already created a 50 word summary, then I execute search for the remaining 50 words that maximize the Rouge score. In all experiments, I use a greedy beam search with a beam of 20. One observes significantly diminishing returns for beams larger than 20.

6.5 Feature Functions

Features in the vine-growth model may consider any aspect of the currently generated summary, and any part of the input document set. The features I employ for this problem are as follows, letting w be the word under consideration and s being the sentence that contains it:

- Word identity, stem and part of speech of w.
- Syntactic relation between w and its parent, or "ROOT".
- The position of s.
- The length of s.
- Whether or not w is enclosed in quotes.
- The length of the document containing w.
- The number of pronouns in s; number in the subtree rooted at w.
- The number of attribution verbs ("say," "state," "observe," etc.) in s; number in the subtree rooted at w.
- Probability of w under a language model for the document containing w.
- Probability of w under a language model for the document collection.
- Probability of w under a language model for the query (derived using the BayeSum technique).
- KL divergence between (a language model for) the sentence containing w and (a language model for) the query (using BayeSum).
- KL divergence between the *subtree* rooted at w and the query (using BayeSum).
- Probability of w under a language model for the *previously extracted summary.*
- KL divergence between the subtree rooted at w and the previously extracted summary.

These are, with a few minor additions, the same features I used in previous work (Daumé III and Marcu, 2005b).

Argentina and Britain announced an agreement Thursday to restore full diplomatic ties , nearly eight years after they fought a 74 – day war over the Falkland islands , a sparsely populated archipelago off Argentina 's coast in the South Atlantic Ocean . Britain is to invite Argentina 's economy and foreign ministers to London this year in official visit to Britain by Argentine ministers since the Falklands war in 1982 . Argentina announced that it has decided to lift financial and trade restrictions on imports from Britain that were imposed during the 1982 Falkland islands War . Hurd to visit Argentina . British boost for Argentina .

Figure 6.5: Example 100-word output from the BayeSum system after rule-based sentence compression and post-processing.

Argentina and Britain announced an agreement to restore diplomatic ties , eight years after they fought a 74 – day war over the Falkland islands . Argentina gets out the red carpet for the UK 's Duke of York , the first official visitor since the end. Douglas Hurd will meet President Menem in Argentina . Britain is to invite Argentina 's ministers to London this year in the first official visit to Britain since the Falklands war in 1982 . Argentina announced that it has decided to lift financial and trade restrictions on imports from Britain that were imposed during the War .

Figure 6.6: Example 100-word output from the Searn-based Vine Growth model after post-processing.

6.6 Experimental Results

Experimental results are shown in Table 6.1. I report Rouge scores for summaries of length 100 and length 250. I compare the following systems. First, an oracle system that performs the summarization task *with* knowledge of the true output. The oracle systems create summaries so as to maximize the Rouge score. I present results for an oracle sentence extraction system (Extr) and an oracle vine-growth system (Vine). Second, I present the results of the Searn-based systems, again for both sentence extraction (Extr) and vine-growth (Vine). Both of these are trained with respect to the oracle system. (Note that it is impossible to compare against competing structured prediction techniques. This summarization problem, even in its simplified form, is far too complex to be amenable to other methods.)

For comparison, I next present results from the BayeSum system (Daumé III and Marcu, 2005b; Daumé III and Marcu, 2006), which achieved the highest score according to human evaluations of responsiveness in DUC 05 and which scored third according to the Rouge 2 metric among roughly 30 systems. This system, as submitted to DUC 05, was *trained* on DUC 2003 data; the results for this configuration are shown in the "D03" column. For the sake of fair comparison, I also present the results of this system, trained in the same cross-validation approach as the Searn-based systems (column "D05"). Finally, I present the results for the baseline system and for the best DUC 2005 system (according to the Rouge 2 metric).

As we can see from Table 6.1 at the 100 word level, sentence extraction is a nearly solved problem for this domain and this evaluation metric. That is, the oracle sentence

	ORACLE		SEARN		BAYESUM			
	Vine	Extr	Vine	Extr	D05	D03	Baseline	Best D05
100 W	0.0729	0.0362	0.0415	0.0345	0.0340	0.0316	0.0181	-
250 W	0.1351	0.0809	0.0824	0.0767	0.0762	0.0698	0.0403	0.0725

Table 6.1: Summarization results; values shown are Rouge 2 scores (higher is better).

extraction system yields a Rouge score of 0.0362, compared to the score achieved by the Searn system of 0.0345. This difference is on the border of statistical significance at the 95% level.

The next noticeable item in the results is that, although the Searn-based *extraction* system comes quite close to the theoretical optimal, the oracle results for the *vine-growth* method are significantly higher. At 100 words, the best vine-growth system can achieve a two-fold improvement in score over the best sentence extraction system. The difference at 250 words is slightly less pronounced (1.6-fold) but still large.

The next thing to notice is that, under Searn, the summaries produced by the vine-growth technique are uniformally better than those produced by raw extraction. This difference is more pronounced at the 100 word level (1.2 times better) than at the 250 word level (1.07 times better), though both are statistically significant. This conforms with the prior expectation that simple sentence extraction is "okay" for long summaries, but is less appropriate for very short summaries.

The last aspect of the results to notice is how the Searn-based models compare to the best DUC 2005 system, which achieved a Rouge score of 0.0725. The Searn-base systems uniformly dominate this result (0.824 for vine-growth and 0.0767 for extraction), but this comparison is not fair due to the training data. We can approximate the expected improvement for having the new training data by comparing the BayeSum system when trained on the DUC 2005 and DUC 2003 data: the improvement is 0.0064 absolute. When this result is added to the best DUC 2005 system, its score rises to 0.0789, which is better than the Searn-based extraction system but not as good as the vine-growth system. It should be noted that the best DUC 2005 system *was* a purely extractive system (Ye et al., 2005).

6.7 Error Analysis

6.8 Discussion and Conclusions

This chapter presents an approach to the summarization problem based on the *vine-growth model*. Under this model, a summary is created on a word-by-word basis by "growing" subtrees that have previously been added. This methodology leads to a compact search space and maintains some grammaticality in the outputs. The trade-off is that there is no closed form solution for the standard Rouge metrics under this model. To remedy this, I apply the search-based optimal policy approximation described in Section 3.6.2. This leads to an efficient and easy-to-implement solution. In experimental results, the vine-growth technique outperforms raw sentence extraction approaches, including the winner of the DUC 2005 competition.

One attractive quality of the vine-growth method is that it is easy to imagine how to generalize it to an *abstractive* summarization system, but including actions that correspond to "replace word X with word Y" or "replace this phrase" or "rotate this subtree." The advantage to building an abstractive system in this manner—by building up upon the vine-growth method—is that one does not necessarily need to incur a performance hit by moving to an abstract, as might be expected if one were to build an abstractive system from scratch.

Chapter 7

Conclusions and Future Directions

In this thesis, I have:

- Presented an algorithm, Searn, for solving complex structured prediction problems with minimal assumptions on the structure of the output and loss function.

- Compared the performance of Searn against standard structured prediction algorithms on standard sequence labeling tasks, showing that it is competitive with existing techniques.

- Developed a large number of useful features for the entity detection and tracking task, and demonstrated that Searn is able to solve this problem with high performance and small computational overhead, even with new features that break model tractability.

- Described a novel approach to summarization—the vine-growth method—and applied Searn to the underlying learning problem, yielding state-of-the-art performance on standardized summarization data sets.

The applications described have shown that Searn is, indeed, a general framework for a large variety of structured prediction problems. Moreover, as it is simple to implement, I have high hopes that researchers both inside and outside of NLP will find it useful for solving even more problems. I have shown that Searn obeys a desirable theoretical property: given a good classification algorithm, one is guaranteed a good structured prediction algorithm. Importantly, this result is independent of the size of the search space or the tractability of the search method. To my knowledge, it is also the first theoretical result that shows that local learning—when done properly—can lead to good global performance.

Despite these successes, there are many structured prediction problems that are currently outside the umbrella covered by Searn. In the next two sections, I describe two areas—weak-feedback models and hidden variable models—for which Searn requires some extensions to be applicable. I further present preliminary results that suggest that it is possible to apply Searn in both of these settings.

7.1 Weak Feedback Models

Weak feedback is a relatively unconventional machine learning setting. The situation that weak feedback learning attempts to model is the following. Suppose you work for an online store. For each product listed on your web site, you would like to have an automatically generated product summary. You are able to spend some fixed amount of money paying humans to write example summaries off which you will train a summarization system, like that described in Chapter 6. Now you deploy this summarization system on your web site. Some users have the option of reporting whether they found the summary useful or not. Or perhaps they rank the summary on a scale from 1 to 5. Or they are presented with two summaries and may select which is better (or "neither"). In all three user situations, you would like to be able to use the feedback received to improve the summarization system. This is the weak feedback setting: your system receives a loss, but one that is only a very weak approximation to the true loss.

7.1.1 Comparison Oracle Model

In this section, I consider a simplified model for weak feedback based on an pairwise oracle setting. The model is as in structured prediction: we have a fixed by unknown distribution $\mathcal{D}$ over pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{Y}$ is structured, and a loss function $l : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$. Additionally, we have a *comparison oracle*, $o : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to [-1, 1]$. The intuition is that $o(x, y, y')$ returns a negative value if y is better than y', a positive value if y is worse and 0 if the two are indistinguishable.

In order to proceed, one must make an assumption about the relationship between o and l. I make the most basic assumption possible. Let y^t be the true output for x; then I assume that Eq (7.1) holds, which clearly satisfies the intuition.

$$\begin{aligned} o(x, y, y') &= \frac{1}{Z_x}\Big[\ell(x, y^t, y) - \ell(x, y^t, y')\Big] \\ Z_x &= \max_{y,y'}\Big|\ell(x, y^t, y) - \ell(x, y^t, y')\Big| \end{aligned} \tag{7.1}$$

The oracle assumption made in Eq (7.1) is of course overly strong. Ideally, one would prefer a model that can accommodate a noisy oracle, or one that can abstain, but for simplicity I consider only this basic setting. This cleanly separates two components of the weak feedback setting: the first component is the fact that one only has implicit access to the loss function (which I retain), while the second is the fact that the oracle is not exactly the same as the loss function (this component I ignore).

7.1.2 Algorithm

Under the comparison oracle model, I assume that we have a small labeled structured prediction data set $D = \langle x_n, y_n \rangle_{1:N}$ and a large unlabeled data set $D^U = \langle x_m \rangle_{1:M}$ together with an oracle function o for the unlabeled data.

Naïvely, one could solve this problem by transforming D^U into a labeled data set by searching, for each $x_m \in D^U$, for the $y \in \mathcal{Y}$ with minimal loss (by searching over all pairs and applying the oracle). Unfortunately, without unrealistic assumptions on the loss function, this would require a search over an exponential number of possible output pairs (think about learning in the context of the classic board game Mastermind, which is NP-complete even to play (Stuckman and Zhang, 2005)). Moreover, in any generalized setting with an imperfect or noisy oracle, such a solution would not be robust.

The solution I propose here (though it is by no means the best possible solution) is to use the labeled data set D to learn a policy π by applying Searn. One may then run a variant of Searn over the unlabeled data, treating the learned π (which does *not* contain the optimal policy) in the role of the current policy. As in Searn, for any step in the search process, this will lead to $|\mathcal{A}|$-many actions. The remaining question is how to compute the corresponding losses, ℓ_a^π from Eq (3.2). Computing the losses can be done by using the oracle to find the action that leads to the smallest loss using $|\mathcal{A}| - 1$ comparisons. Once the best action a^t is found, the loss ℓ_a^π for any action a can be computed using $o(x, a^t, a)$. Again, this takes at most $|\mathcal{A}| - 1$ comparisons.

The full algorithm proceeds as follows:

1. Learn a policy $\pi = \text{Searn}(D, \pi^*, \text{Learn})$ for a given π^* on the labeled data and some classifier Learn.

2. Learn a new policy π' on the union of the standard data set D (using π^* as the optimal policy) and the unlabeled data set D^U (using π as the optimal policy and computing losses as described above).

3. Set $\pi = \pi'$ and go to (2) until performance drops on development data.

7.1.3 Analysis

The algorithm described for learning in the comparison oracle model works intuitively precisely because Searn works. The only potential concern is that on the unlabeled data, the policy used as "optimal" is not, in fact, optimal. However, since the algorithm still uses the optimal *losses*, this does not appear to present a problem. Nevertheless, I do not currently have a theoretical analysis of this algorithm akin to Theorem 3.6.

An important question is: how many times will the oracle be called? Since the model is of a situation where a human is asked for opinions on outputs, one does not wish the number of queries to be too large (otherwise it might be less expensive to simply have the humans directly annotate the data). Suppose there are N unlabeled examples, each of length T and there are $|A|$ actions at each time step. Then, in the worst case, $2N\,|A|\,(T-1)$ queries will be asked for each iteration of the algorithm.

While the number of questions is linear in all important variables, it may still be too large in practice. In this case, one could directly apply research in active learning; see (Cohn, Ghahramani, and Jordan, 1996; Greiner, Grove, and Roth, 1996; Roy and McCallum, 2001) for representative work. This would also fit nicely in extending these results to a noisy setting. The most similar active learning results is incomplete boundary query (IBQ) model of Blum et al. (1998) (also, see the intermediate concepts model of Kwek (2001)). The IBQ model is an active learning PAC-style learning framework (Valiant, 1994; Angluin et al., 1997) in which the "labeler" can abstain. In particular, in

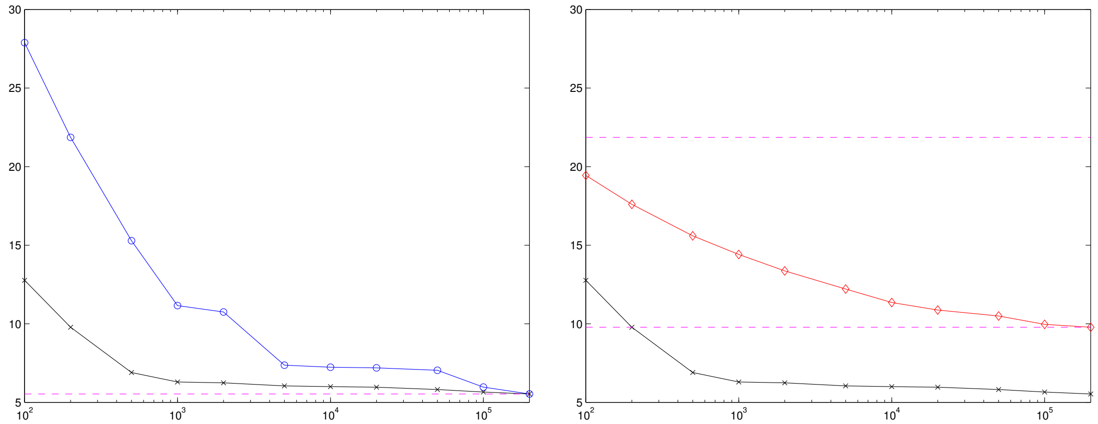


Figure 7.1: Learning curves for weak-feedback experiments on syntactic chunking; y-axes are $1 - F$. (Left) X-axis is amount of supervised data available. The higher (circled blue) curve is the purely supervised setting; the lower (crossed black) curve is when the remaining data is used as a weak-feedback oracle. (Right) The higher (red diamond) curve is keeping the amount of supervised data constant (200 words) and varying the amount of oracle data; the lower (crossed black) curve is replicated from left.

the binary classification setting, one defines a radius of uncertainty r around a boundary function. Whenever an input x is within a distance of r to the boundary, the labeler can abstain. However, to obtain positive results, (Blum et al., 1998) assume that the boundary region has zero measure under the data generating distribution. Evaluating the implications of these assumptions in the weak feedback model is left to future work.

7.1.4 Experimental Results

I have performed two experiments, both based on the syntactic chunking task described in Section 4.1.3. The results of the experiments are shown in Figure 7.1.

The first experiment aims to determine how much the performance of a system is adversely affected by using weak feedback rather than true labeled examples. The experiment as as follows. We vary the amount of supervised data available (from 100 words to $200,000$ words) and train a Searn-based model on this data. This gives an upper bound on the error we can expect with a weak feedback model. Next, we vary the amount of supervised data and use the remaining data as unlabeled data for which a weak feedback oracle is available.

The results are shown in the Left of Figure 7.1. On the very left of this figure, we see that the purely supervised model (upper, circled blue curve) achieves an error rate of roughly 28. When we throw in $200,000-100$ words of weak feedback data (lower, crossed black curve), this drops substantially to roughly 13. On the far right of the figure, the two curves converge at an error rate of roughly 6 (in this case, no weak feedback data is used). As we can see, the weak feedback curve drops very quickly (after roughly 1000 words of standard labeled data) to an error rate very near that of optimal. This suggests that, at least under the idealized model, this is a viable approach.

The second experiment aims to understand how much oracle data is needed to achieve good performance. In this experiment, we hold the amount of standard labeled data constant at 200 words. Then, we vary the amount of weak feedback data available (again, from 100 words to $200,000$ words).

The results are shown in the Right of Figure 7.1. The upper horizontal line is the performance of a system trained *only* on the 200 words of supervised data. The lower horizontal line is the performance of a system trained on 200 words of supervised data and $200,000$ words of weak feedback data. The lower (crossed black) curve is replicated from the Left of the figure. The upper (diamond red) curve is the results of varying the amount of weak feedback data. As we can see, comparing the upper and lower curves, there is a significantly greater bang-for-the-buck of adding supervised data rather than weak feedback data (the error rate of the lower curve drops more sharply).

7.1.5 Discussion

In this section, I described a simple model for a weak feedback setting in which an oracle provides comparisons between proposed outputs. I presented an algorithm based on Searn for learning in this setting and provided preliminary experimental results that suggest this is a viable approach.

The experimental results support two conclusions. First, given access to a large amount of weak feedback data, one need significantly less standard supervised data to learn well. In fact, one can obtain comparable results to having $100k$ labeled words with only 1000 labeled words and a large amount of oracle data. Second, one can achieve the same performance level of 500 words of labeled data with only 200 words of labeled data and 500 words of weak feedback data. These results suggest that this is a promising direction for further investigation, both at a theoretical and practical level.

7.2 Hidden Variable Models

Hidden variable models are extremely popular in natural language processing, perhaps first popularized by Brown et al. (1993). The hidden variable framework allows for the construction of models with apparently complex interdependencies between variables that can be modeled in a straightforward manner. While most hidden variable models are formulated in a generative fashion, recent work has shown that conditional random fields are also amenable to such settings (McCallum, Bellare, and Pereira, 2005).

The setting I am interested in is *supervised* learning with hidden variables, rather than the more common setting of *unsupervised* learning with hidden variables. In particular, the setting I am concerned with is the case where the problem to be solved is a standard supervised learning problem, but there are underlying hidden variables that will be useful

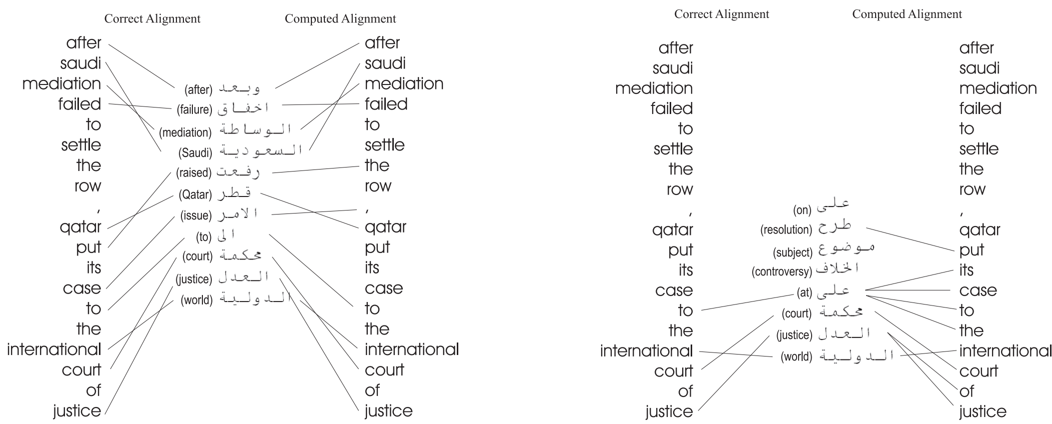


Figure 7.2: Two example alignments used in the translation classification task. The left alignment is for a positive example; the right alignment is for a negative example.

to solving it.[1] While direct applications of hidden variable models to supervised natural language processing problems are rare (though see Matsuzaki, Miyao, and Tsujii (2005) and Koo and Collins (2005) for two related exceptions in the realm of parsing), there seems to be a significant amount of potential in the framework.

7.2.1 Translation Classification

As a running example task, I will use the translation classification problem. This problem arises when one wishes to use *comparable corpora* (instead of the standard *parallel corpora*) for learning models for machine translation. In this setting, one extracts a large set of English sentences and a large set of (say) Arabic sentences from two corpora on roughly the same topic. One then wishes to identify which English/Arabic sentence pairs are mutual translations. This is a pure binary classification task: given an input (English/Arabic sentence pair), determine whether or not the two are mutual translations. If done well with sufficient data, this approach can lead to improved machine translation performance (Munteanu and Marcu, 2005). Although this discussion will focus on the translation classification problem, it is strongly related to several other tasks. In paraphrase identification, one attempts to solve virtually the same problem, but with both sentences in the same language (Barzilay and McKeown, 2001; Dolan, Quirk, and Brockett, 2004). Similarly, in textual entailment, one attempts to build a classifier to detect if one of two sentences logically follows from the other (Raina, Ng, and Manning, 2005; Oren Glickman, 2005). Related approaches are used for all three of these problems.

Munteanu and Marcu (2005) solve the translation classification problem by learning a standard binary classifier (in practice, they use a maximum entropy model). The

[1] This contrasts with the less supervised setting where one only has *partial* annotations and treats the true output as a hidden variable problem; as done by, for example, Riezler et al. (2002) and Clark and Curran (2004). These tasks fit more closely to the weak feedback setting discussed in Section 7.1.

interesting aspect of their approach is the feature set. First, they compute word-to-word and phrase-to-phrase *alignments* between the English sentence and its hypothesized Arabic translation. Example alignments are shown in Figure 7.2. Features are then defined over the alignments, rather than over the sentence pairs. The key intuition is that the alignment for the positive example is significantly more "natural" that the one for the negative example (more words aligned, closer to a monotone order, smaller fertilities, etc.). The features are designed to reflect these factors.

The key question I am interested in is: how does one derive these alignments. In their original work, Munteanu and Marcu use a small parallel corpus to build an unsupervised alignment model using standard techniques (Brown et al., 1993) and the GIZA++ implementation (Och and Ney, 2003). The alignment model and its implementation have been heavily tuned to produce alignments that are good for the purpose of translation. It is a priori unclear that such alignments are also good for the purpose of distinguishing parallel and non-parallel sentence pairs. What we seek is a method to learn how to perform alignments that are optimal from the perspective of the end task we wish to solve. In this case, that task is translation classification.

7.2.2 Search-based Hidden Variable Models

In the majority of the applications of hidden variable models in the natural language processing community, the desired hidden variables are structured, much like the alignments in the translation classification task. Given that the hidden variables are structured, it appears natural to attempt to apply Searn to such problems. The difficult that arises is in defining a loss function. This difficulty exists because we do not actually have annotated data for these hidden variables. Moreover, we do not necessarily have access to a loss function for them.

Abstracting from a particular application, consider the following general settings. We have three spaces: an input space $\mathcal{X}$, an output space $\mathcal{Y}$ (which may or may not be structured) and a hidden variable space $\mathcal{Z}$. We also have a loss function ℓ over the true output space, but no such function over the hidden variable space. Our training data is a set of examples drawn from a distribution over $\mathcal{X} \times \mathcal{Y}$. Our goal is a function $h : \mathcal{X} \to \mathcal{Y}$ with low expected loss over examples drawn from the source distribution (see Section 2.1).

However, in the hidden variable model case, we *believe* that it is easier to solve this task in two steps. First, we use the input $x \in \mathcal{X}$ to find a value for the hidden variables $z \in \mathcal{Z}$. Then, we use the input x and the hidden variable z to produce a classification. This breakdown is shown in Eq (7.2).

$$f : \mathcal{X} \to \mathcal{Z} \qquad , \qquad g : \mathcal{X} \times \mathcal{Z} \to \mathcal{Y} \qquad , \qquad h(x) = g(x, f(x)) \tag{7.2}$$

Those familiar with the expectation maximization family of algorithms (Dempster, Laird, and Rubin, 1977) might be put off by the fact that f "selects" a single hidden variable $z \in \mathcal{Z}$ rather than a distribution over hidden variables. It is straightforward to generalize this set up to the case where f produces more than a single output, as in Eq (7.3).

$$f : \mathcal{X} \to \Upsilon(\mathcal{Z}) \qquad , \qquad g : \mathcal{X} \times \Upsilon(\mathcal{Z}) \to \mathcal{Y} \qquad , \qquad h(x) = g(x, f(x)) \tag{7.3}$$

In Eq (7.3), $\Upsilon(\mathcal{Z}) \subseteq \{ p : \mathcal{Z} \to R \}$ for some set R. That is, $\Upsilon(\mathcal{Z})$ can be thought of as selecting a weighted subset of $\mathcal{Z}$. An element of $\Upsilon(\mathcal{Z})$ is a function that maps an element of $\mathcal{Z}$ to a "score" in the set R. When $\Upsilon(\mathcal{Z})$ contains only a single element, Eq (7.3) reduces to Eq (7.2). On the other hand, when R is the tropical semi-ring and $p \in \Upsilon(\mathcal{Z})$ satisfies $p(z) \geq 0^R$ and $\sum_{z \in \mathcal{Z}}^R p(z) = 1^R$, then this can be interpreted as a probability distribution over hidden variables. In intermediate settings, $\Upsilon(\mathcal{Z})$ can be thought of as a weighted beam (priority queue) over the "best" outputs.

For simplicity, I will focus on the single-output case (Eq (7.2)) for the remainder of this section. However, the extension to the full expectation case and to the beam case is straightforward. See Section 7.2.5 for a comparison to the EM approach. Note that, by assumption on $\mathcal{Z}$, learning f (from Eq (7.2)) is a structured prediction problem.

7.2.2.1 Iterative Algorithm

It turns out that this formulation introduces another chicken and egg problem. That is, we seek a pair of functions (f, g) so that f produces hidden variables that are optimal for g, and g produces classifications based on features that depend on the output of f. As before, we tackle the chicken and egg problem via iteration. The algorithm is straightforward. It begins by initializing either f or g according to prior knowledge. Suppose f is initialized by prior knowledge. Then, g is optimized on the basis of the output of f, holding f fixed. Once g has been optimized, f is *re-learned* on the basis of the new g. This process of alternating learning ends when the performance of f ceases to increase.

There are two important aspects of this algorithm: first, unlike Searn, there is no interpolation; second, the success hinges on the initialization.

Interpolation is unnecessary in the hidden variable case because we will use an optimal policy for learning f that is sufficient to guarantee convergence. This choice is described in Section 7.2.2.2, but, briefly, the optimal policy is chosen so that successfully learning the hidden variable model will immediately imply that the classifier g can do no worse.

It is somewhat undesirable that the algorithm should be sensitive to initialization. Ideally, one would like an algorithm that can initialize itself, or one whose performance guarantees do not depend on the quality of the initialization. Unfortunately, this is currently not possible with the approach described in this chapter. It is some conciliation that this sensitivity to initialization is not limited to this Searn-based approach. The hidden variable CRF models (McCallum, Bellare, and Pereira, 2005) are highly sensitive to initialization; in the work on learning string-edit distances, the authors had to initialize the models by hand-setting some feature weights (Bellare, 2005). Even unsupervised models trained with EM are highly sensitive to initialization; in applying the IBM machine translation models, one must follow a strict sequence of initialization steps in order to obtain competitive results (Och and Ney, 2003).

One advantage to the approach described in this section is that one can initialize *either* the hidden variable model f *or* the classifier g. In some cases—such as the translation classification problem—it may be easier to initialize the hidden variable model f by beginning with a given unsupervised model, such as that given by GIZA++. In other cases, it may be easier to initialize the final classifier g by hand-setting some weights according to prior information. Note that an obvious idea—initializing g by training a model that ignore the hidden variables—will not be possible in our setting. This is because the definition of the optimal policy for learning f will assume that g is *not* invariant to the values of the hidden variables.

7.2.2.2 Optimal Policy

In order to apply SEARN to the hidden variable framework, we need only define the optimal policy. The key to the definition is the notion that a good value for the hidden variable is one that makes the classification problem *easier*. The optimal policy will be defined to be the policy that allows the *fixed* classification function g to do best with respect to its loss function (note that the loss function optimized for g may *not* be the same as the overall loss function ℓ defined for the problem—more on this later).

Formally, let the classifier g be fixed and let l^g be the loss function optimized for g. For all classification pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$, define the *cost* of a hidden variable $z \in \mathcal{Z}$ (denoted $c_{x,y}(z)$) to be $l^g(y, g(x, f(x)))$. That is, the cost of the hidden variable is the loss associated with the classification made when z is assumed to be correct. Intuitively, a low cost means that the classification is easy.

This definition of cost enables us to formally construct a distribution over structured prediction problems as defined in Def 3.1. This distribution, $\mathcal{D}^{\text{SP}}$, is over the input space $\mathcal{X}$ and has cost vectors indexed by elements of $\mathcal{Z}$. Given a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, a classifier g and a loss function l^g, we simulate a structured prediction distribution $\mathcal{D}^{\text{SP}}$ by first drawing a pair $(x, y) \sim \mathcal{D}$, then producing a structured prediction example whose input is x and whose cost vector $\boldsymbol{c}$ is given by $c_{x,y}(\cdot)$. That is, the cost of a structure z is simply the cost of the final classification algorithm.

In most cases, an analytic solution to the optimal policy for a cost vector constructed thusly is impossible. We must therefore result to a search-based optimal policy (see Section 3.6.2). That is, we execute a standard search algorithm (say, beam search) over the cost vectors $\boldsymbol{c}$ to find an approximately optimal step. One aspect of this search, as opposed to the classical search described in Section 3.6.2 is that the argument that search will always succeed does not hold. Moreover, one cannot easily ascertain whether search errors are being made. One must therefore be more judicious in constructing the search space and actions for hidden variable problems than perhaps for standard structured prediction problems.

7.2.3 Features and Data

The experimental results I present in this section are intended only as a proof of concept. The task is the translation classification problem (see Section 7.2.1). For simplicity, I use a parallel corpus (English/Spanish) that was hand crafted by Kevin Knight as a educational tool to explain alignment models. The full corpus of twelve parallel sentences is shown in Figure 7.3.

Garcia and associates . Garcia y asociados .	the clients and the associates are enemies . los clientes y los asociados son enemigos .
Carlos Garcia has three associates . Carlos Garcia tiene tres asociados .	the company has three groups . la empresa tiene tres grupos .
his associates are not strong . sus asociados no son fuertes .	its groups are in Europe . sus grupos estan en Europa .
Garcia has a company also . Garcia tambien tiene una empresa .	the modern groups sell strong pharmaceuticals . los grupos modernos venden medicinas fuertes .
its clients are angry . sus clientes estan enfadados .	the groups do not sell zenzanine . los grupos no venden zanzanina .
the associates are also angry . los asociados tambien estan enfadados .	the small groups are not modern . los grupos pequenos no son modernos .

Figure 7.3: Custom corpus used for proof of concept experiments for hidden variable alignments model.

In order to turn this corpus into a translation classification corpus, I use the twelve displayed sentence pairs as positive examples. To create the negative examples, I took each English sentence and paired it with a random incorrect Spanish sentence. I perform leave-two-out cross validation (always one correct and one incorrect sentence left out) where the incorrect sentence pairs are chosen exclusively from the training data.

When applying hidden-variable SEARN, one needs to specify two sets of features: the structured prediction features for learning the hidden variables and the classification features for the final classifier. For this task, I use an identical feature set for both:

- Number of unaligned words (and fraction of total words)
- Distortion distance
- Aligned word pairs
- Null-aligned words
- Number of crossing alignments
- Maximum fertility
- Difference in sentence lengths

To draw a comparison, I also train a raw binary classifier. This classifier uses the following features:

- Difference in sentence lengths
- Pairs of *all* words

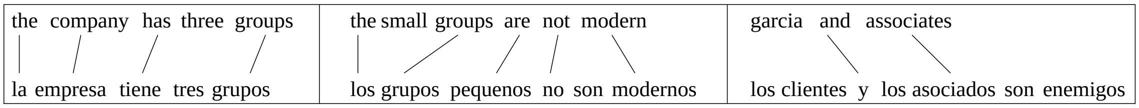


Figure 7.4: Three alignments found during the Searn-based hidden variable training; the left two are positive examples, the right-most example is negative.

7.2.4 Experimental Results

As a baseline, I use a binary classifier using similar features to those described in the previous section. However, lacking alignments, I use all pairs of words as features, and do not use any of the distortion/fertility/crossing features. I use a vanilla maximum entropy classifier and tune the prior to achieve good results on development data. The results of this baseline system under the cross-validation experimental setting described above is an error rate of 40.67%, which is rather close to the worst possible performance of 50%. (If, instead, the prior parameter is chosen to achieve the best possible test set performance—i.e., if we cheat—the error rate drops to 38.33%.) When the Searn-based model is trained with alignments enabled (still using maximum entropy classifiers), the error rate under the same cross-validation setting drops to 32.33% (and down to 30% under the cheating setting). A 30% error rate is still quite bad, but the addition of the hidden variable does enable a significant increase in performance.

For an intuitive analysis of what the model is doing, I have shown the hypothesized alignment for three sentences (all test data) created by Searn in Figure 7.4. The first two examples are positive; the third is negative. Both positive examples are reasonable. In the left-most sentence pair, the model missed the alignment of “three” to “tres” precisely because this word pair did not appear in any of the training examples. In the middle sentence pair, the model misaligns “are” to “pequenos.” This misalignment is likely due to learned distortion features, rather than word features, since “pequenos” does not appear in any of the training sentences. In the final, negative example, the model aligns “and” and “y” and “associates” to “asociados”, which are both reasonable alignments. The rest of the words are left unaligned, which is appropriate for this example.

7.2.5 Comparison to Expectation Maximization

Expectation maximization (Dempster, Laird, and Rubin, 1977) is a family of algorithms for performing maximum likelihood estimation in probabilistic models with hidden variables. That is, we have a density $p(x \mid \theta)$ that we wish to maximize with respect to θ. We do so by introducing a hidden variable and instead seek to maximize $sum_z p(z \mid \theta) p(x \mid z, \theta)$. The introduction of the sum over hidden variables is necessary according to the probability calculus (Cox, 2001), but makes standard maximum likelihood estimation difficult. This difficulty stems from the fact that to perform MLE, one typically takes the log of the probability distribution. This, whenever p belongs to the exponential family, allows straightforward computation of the gradient of $\log p$ with respect to θ in a closed form.

Unfortunately, in the hidden variable model, one cannot move logs inside sums, making the straightforward approach difficult. However, one can instead optimize a tight lower bound on the joint probability by applying Jensen's inequality (Jensen, 1906) to the $\log \sum_z$ yielding a resulting distribution of the form $\mathbb{E}_{z \sim p(\cdot \mid x, \theta)}[\log p(x \mid z, \theta)]$, which can be optimized the the standard form. This procedure involves two steps: computing the expectations of the hidden variables (the "E" step) and maximizing the log probability with respect to these expectations (the "M" step). By iterating these steps, one will converge to a local maximum in the incomplete log likelihood space obtained by marginalizing over the hidden variables: $p(x \mid \theta)$. Bilmes (1997) and Knight (1999) provide excellent introductions to EM from an NLP perspective.

Before drawing parallels to EM, it is worth noting that the algorithm described above for applying Searn to hidden variable problems is quite different from the probabilistic approach. Foremost, Searn does not produce probabilistic models. It seems to minimize an empirical loss. Moreover, Searn is discriminative rather than generative. That is, the probabilistic approach models a density $p(x)$, and does not explicitly seek to be able to predict. Even if Searn were retrofitted to function in a probabilistic setting, vanilla EM would be inappropriate (though one could attempt to apply conditional EM (Jebara and Pentland, 1998)).

The key similarity between EM and hidden-variable Searn is in the iterative approach to both algorithms. The necessity of iteration is not surprising: the problem being solved is similar in both cases. The problem is a chicken-and-egg problem, which leads naturally to an iterative approach. Both algorithms solve this chicken and egg problem by first guessing at the hidden structure, then learning a new model (a new density for EM or a new predictor for Searn) on the basis of this guessed hidden structure. The primary difference between the two is that in the "guessing" step, EM guesses an entire distribution over hidden variables, while Searn guesses a single hidden variable.

The reason why the Searn-based approach does not require a full expectation is that training against an optimal policy is a stronger requirement than maximizing a density $p(z)$. In particular, the optimal policy is carefully constructed so that a successful application of Searn will yield convergence guarantees for the hidden variable algorithm. The simple computation of expectations in EM is a weaker requirement and thus leads to weaker theoretical results.

A stronger connection to EM can be seen when Searn is run in exact-search mode, rather than single-best mode. In this case, it can be made to produce every value for z, each with a different score (or at the very least in a ranked order). These scores can be fed through a soft-max function to produce "probabilities." This induces a distribution over the hidden variables, much akin to the EM case. If we then structure the g function from Eq (7.3) as shown in Eq (7.4), we obtain a maximization step that parallels that of EM (for the pure classification setting).

$$\begin{aligned} g &: \mathcal{X} \times \Upsilon(\mathcal{Z}) \to \mathcal{Y} \\ g(x, p) &= \operatorname{sgn}\left[\sum_{z \in \operatorname{dom}(p)} g'(x, z) \exp p(z)\right] \\ g'(x, z) &= \text{any binary classifier} \end{aligned} \tag{7.4}$$

Under this definition of g, we use the soft-max probabilities (i.e., $\exp p(z)$) given by the scoring function to induce a distribution over which our classifier votes. The binary classifier g' is trained as in any standard classification algorithm. In this case, however, f and g are structured so that the algorithm more closely resembles EM, though this is of course not necessary. The g function could directly use the beam output by f, rather than simply voting over it.

7.3 Other Applications for Searn

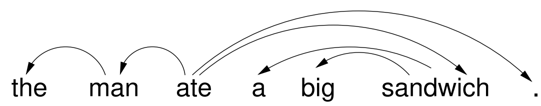
In this thesis, I have presented strong empirical results demonstrating the efficacy of Searn on three types of problems: sequence labeling problems, the entity detection and tracking problem, and an automatic document summarization problem. In this chapter, I have presented preliminary evidence that Searn can also be effectively applied for a supervised word alignment task with hidden variables. In this section, I briefly discuss the possibilities of applying Searn to two other well-known problems in natural language processing: parsing and machine translation.

7.3.1 Parsing

In this section, I consider the problem of dependency parsing in a bottom-up, left-to-right framework (Nivre, 2003; Sagae and Lavie, 2005; Turian and Melamed, 2006). The choice of a dependency model is primarily for convenience. The extension to the constituency case is a bit more involved, but still possible. The extension to a non-left-to-right framework (i.e., to CKY parsing (Kasami, 1965; Younger, 1967)) is somewhat less obvious. The primary difficulty in the CKY model is that the search algorithm and the dynamic programming table (the chart) are intrinsically tied. Perhaps the easiest method for thinking about applying Searn in a standard constituency-based parsing framework is in the hyper-graph formalism (Klein and Manning, 2003a), where a Searn-action corresponds to traversing a hyper-edge in the chart.

As a running example, consider the sentence "the man ate a bit sandwich ." The correct (unlabeled) dependency parse for this sentence is shown in Figure 7.5. The standard assumption for dependency parsing is that of projectivity: essentially, none of the arcs cross. This assumption is true in most languages, but untrue, for instance, in Czech (McDonald, Lerman, and Pereira, 2006). In the shift-reduce framework, a dependency tree is built through a sequence of steps. The parser maintains an active *stack* onto which words are pushed using the *shift* action. The top two elements on the stack (the most recent two) can be combined using a reduce action. There are two reduce actions: one for each possible direction the arrow could point. The complete derivation of the tree is shown in the right of Figure 7.5.

It is clear from this analysis that the decision of shift/reduce-left/reduce-right could be accomplished using Searn. The standard loss function for this problem is Hamming loss over dependencies (sometimes directed, sometimes undirected). I consider the undirected case for simplicity. Again, the key questions are defining the optimal policy and (perhaps) the optimal approximation loss. It is surprisingly easy to define the optimal policy for



shift(the)
shift(man)
reduce(the ← man)
shift(ate)
reduce([the man] ← ate)
shift(a)
shift(big)
shift(sandwich)
reduce(big ← sandwich)
reduce(a ← [big sandwich])
reduce([. . . ate] → [a. . .])
shift(.)
reduce([the. . . sandwich] → .)

Figure 7.5: (Left) The dependency tree for the sentence "the man ate a big sandwich ." (Right) The sequence of shift-reduce steps that leads to this parse structure.

this problem. Note that a partial hypothesis (state in the search space) for this problem is represented by the stack, which I denote $s_1, \ldots, s_I$.

$$\pi^*(x, y, s_{1:i-1}) = \begin{cases} \text{shift} & i \leq 2 \\ \text{reduce} & \text{there are no words left to shift} \\ \text{reduce} & \text{there is an arc between } s_{i-2} \text{ and } s_{i-1} \\ \text{shift} & \text{otherwise} \end{cases} \tag{7.5}$$

The reason this is optimal is as follows. If there should be a reduction between the two most recent words, we have to create it now because we will never have a chance again. Otherwise, any mistakes we have made so far are hopeless: we cannot recover. We might as well shift until we have nothing left to shift and then start reducing. There are a few degrees of freedom: we should alternatively reduce until we cannot reduce any more and then start shifting. In practice, one might want to randomize these choices.

The computation of the optimal approximation loss is even easier. Any incorrectly specified arcs encountered thus far cannot be fixed, so one must accumulate error for them. Any arcs not encountered thus far can always be satisfied. So the optimal approximation Hamming loss is simply the Hamming loss up until the current step.

The application of Searn to a left-to-right dependency parsing model appears to be a promising application. In fact, in a quantitative comparison between a greedy left-to-right dependency parsing algorithm (Nivre, 2003) and a parser based on exact inference (McDonald, Lerman, and Pereira, 2006), the left-to-right algorithm achieves nearly the same performance as the exact model. Qualitatively, the left-to-right system tends to make better parsing decisions early, but suffers from error propagation (McDonald, 2006b). The Searn algorithm is designed explicitly to cope with such problems, which implies that a direct application to Searn to the greedy parser might close the gap in accuracy with little loss in efficiency.

7.4 Machine Translation

Machine translation is, to many people, the Holy Grail of NLP. In this section, I discuss an incredibly simple model for MT, but the extension to more complex models is relatively straightforward. Specifically, I only consider left-to-right translation. This framework covers most models of word-based (Germann et al., 2003) and phrase-based translation (Och, 1999; Koehn, Och, and Marcu, 2003), though not necessarily all recent models of syntactic translation (Melamed, 2004; Chiang, 2005; Quirk, Menezes, and Cherry, 2005; Galley et al., 2006).

There are at least two ways to think about applying SEARN in the machine translation setting. The first is to use SEARN to play the same role that min-BLEU training (Och, 2003) and to hold the various translation features fixed (typically there are roughly twelve, including translation probabilities, language model probabilities, length penalties, etc.). It seems likely that, used in this setting, SEARN should perform roughly comparably to the standard parameter tweaking methods based on Powell's method or grid search.

The second approach to applying SEARN in a translation setting is in an end-to-end manner. In this setting, one would forgo the offline construction of the various probabilistic table (eg., the translation table) and directly optimize a SEARN model. This approach is the one described below. The advantage to applying SEARN in this setting is that one can easily optimize over hundreds of thousands of parameters, rather than being limited to tens of features, as required by Powell's method and grid search. However, as in the hidden variable setting, it is likely that one would need to initialize SEARN appropriately, since the MT problem falls under the heading of hidden variables (the hidden variables are the phrase segmentations and the phrasal alignments). One could easily apply a standard word- or phrase-alignment model (Och and Ney, 2003) to obtain initial estimates to bootstrap SEARN.[2]

In the left-to-right setting, the optimal policy question becomes the following: given a set of reference translations R, an English translation *prefix* $e_1, \dots, e_{i-1}$, what word (or phrase) should be produced next (or is the translation complete?). The construction of the optimal policy is driven by some loss function l. The choice of l depends on what evaluation criteria we wish to optimize; eg., BLEU (Papineni et al., 2002), NIST (Doddington, 2004b), Meteor (Banerjee and Lavie, 2005), etc. In some cases, it may be possible to analyze a particular evaluation criteria and derive a closed form optimal policy (though this seems doubtful for the more complex measures). So, to maintain generality *and* to demonstrate the SEARN is applicable even when the optimal policy is *not* available in closed form, I suggest taking the search-based approach, as in the summarization example from Chapter 6.

For machine translation, the optimal policy search is a very natural search problem. We have a search space over prefixes of translations. Actions include adding a word (or

[2] I have just discovered that a very similar model has been proposed by Liang et al. (2006) and evaluated on French-English translation. This approach uses standard structured perceptron updates, requires initialization of alignments, and is applied only to $67k$ short (5-15 word) sentences. One interesting result of this work is that updating the structured perceptron toward the true output does not work; rather, one has to update it toward the best output the model proposes from an n-best list. This is essentially due to a shortcoming in the structured perceptron algorithm and would not be a problem for SEARN.

phrase) to the end of an existing translation. Our reward function is, for example, Bleu. We want to find the best *full* output starting at some given prefix. Once we have the best full output, we simply inspect the first *decision* of that output. In order to make this search process more tractable, it is useful to restrict the search space. In fact, it is easy to verify that, for the purpose of computing the optimal policy, we only ever need to consider adding words that actually occur in a reference summary (and are not already covered). Moreover, for both Bleu and NIST, we can easily compute an admissible heuristic based on uncovered unigram counts: we can simply assume that we will get all of them correct but no corresponding bigrams or trigrams. It is likely possible to come up with better heuristics, but the unigram statistic is simple.

7.5 Limitations

One potential limitation to Searn is that when one trains a new classifier on the output of a previous iteration's classifier, it is usually going to be the case that previous iteration's classifier performs better on the training data than it will on the test data. This means that, although training via Searn is likely preferable to training against *only* an optimal policy, it is probably still overly optimistic. Based on the experimental evidence, it appears that this has yet to be a serious concern, but it remains worrisome. There are two easy ways to combat this problem. The first is simply to attempt to ensure that the learned classifiers do not overfit at all. In practice, however, this can be difficult. Another approach that would likely completely remove this problem—albeit at a computational cost—is to use cross-validation. Instead of training one classifier in each Searn step, one could train ten, each holding out a different 10% of the data. When asked to run the "current" classifier on an example, one uses the classifier that was not trained on that example.

A second limitation, pointed out by Zhang (2006), is that there is a slight disparity between what Searn does at a theoretical level and how Searn functions in practice. In particular, Searn does not *actually* start with the optimal policy. Even when we can construct an exactly optimal policy, the "true outputs" on which this optimal policy are based are potentially noisy. This means that while the π^* is optimal for the *noisy* data, it is not optimal for the *true* data distribution. In fact, it is possible to construct noisy distributions where Searn will perform poorly.[3] The result of this is that the *theory* gives us a regret bound, but the practice gives us only a loss bound, unless we are in a noise-free setting. This drawback bears consideration, but I currently do not know how to fix it.

7.6 Conclusions

From the perspective of natural language processing, Searn serves as a interpreter through which NLP researchers can talk to machine learning researchers. The availability

[3]One can construct such a noisy distribution as follows. Suppose there is fundamental noise and a "safe" option which results in small loss. Suppose this safe option is always more than a one step deviation from the highly noisy "optimal" sequence. Searn will be confused by this divergence.

of easy-to-use software packages for classification and regression problems have enabled the NLP community to make use of such algorithms for these two simple applications. Unfortunately, the majority of interesting problems in NLP involve producing structured outputs, and there is some desire in NLP to apply new, state-of-the-art machine learning methods to such complex problems. Unfortunately, the amount of effort required to do so means that one must become an expert in machine learning. Searn solves this problem for structured prediction by allowing the NLP researcher to largely ignore the machine learning component that goes on under the hood, and focus exclusively on designing good models and good features for a problem. Searn then acts as a link between this model and features and the underlying machine learning algorithm.

In the context of structured prediction algorithm, Searn lies somewhere between global learning algorithms, such as M^3Ns and CRFs, and local learning algorithms, such as those described Punyakanok and Roth (2001). The key difference between Searn and global algorithms is in how uncertainty is handled. In global algorithms, the search algorithm is used at test time to propagate uncertainty across the structure. In Searn, the prediction costs are used during training time to propagate uncertainty across the structure. Both contrast with local learning, in which no uncertainty is propagated.

From a wider machine learning perspective, Searn makes more apparent the connection between reinforcement learning and structured prediction. In particular, structured prediction can be viewed as a reinforcement learning problem in a degenerate world in which all observations are available at the initial time step. However, there are clearly alternative middle-grounds between pure structured prediction and full-blown reinforcement learning (and natural applications—such as planning—in this realm) for which this connection might serve to be useful.

Despite these successes, there is much future work that is possible. In addition to the extensions proposed in this thesis (which themselves entail a significant amount of future work to scale and analyze), there are many possibilities for future directions. One significant open question on the theoretical side is that of sample complexity: how many examples do we need in order to achieve learning. Related problems of semi-supervised and active learning in the Searn framework are also very interesting and likely to produce powerful extensions.

Another vein of research is in applying Searn, or another *search-based structure prediction* algorithm to domains other than language. Structured prediction problems arise in a large variety of settings (vision, biology, system design, compilers, etc.). For each of these domains, different sorts of search algorithms and different sorts of features are necessary. Exploring such applications is likely to give rise to many more interesting research questions.

References

Altun, Yasemin, Thomas Hofmann, and Alexander Smola. 2004. Gaussian process classification for segmenting and annotating sequences. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Ando, Rie and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research (JMLR)*, 6:1817–1853.

Angluin, Dana, Martins Krikis, Robert Sload, and Gyorgy Turan. 1997. Malicious omissions and errors in answers to membership queries. *Machine Learning (ML)*, 28:211–255.

Aone, C. and S.W. Bennett. 1995. Evaluating automated and manual acquisition of anaphora resolution strategies. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 122–129.

Averick, Brett M. and Jorge J. Moré. 1994. Evaluation of large-scale optimization problems on vector and parallel architectures. *SIAM Journal of Optimization*, 4.

Bagnell, J. Andrew, Nathan Ratliff, and Martin Zinkevich. 2006. Maximum margin planning. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Banerjee, Satanjeev and Alon Lavie. 2005. Meteor: An automatic metric for MT evaluation with improved correlation with human judgments. In *In Proceedings of Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization at ACL*.

Bansal, N., S. Chawala, and A. Blum. 2002. Correlation clustering. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 238–247.

Bartlett, Peter, Michael Jordan, and Jon McAuliffe. 2005. Convexity, classification, and risk bounds. *Journal of the American Statistical Association (JASA)*. To appear.

Bartlett, Peter and John Shawe-Taylor. 1999. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT Press, pages 43–54.

Bartlett, Peter L., Michael Collins, Ben Taskar, and David McAllester. 2004. Exponentiated gradient algorithms for large-margin structured classification. In *Advances in Neural Information Processing Systems (NIPS)*.

Barzilay, Regina. 2003. *Information Fusion for Mutlidocument Summarization: Paraphrasing and Generation*. Ph.D. thesis, Columbia University.

Barzilay, Regina and Kathleen McKeown. 2001. Extracting paraphrases from a parallel corpus. In *Meeting of the Association for Computational Linguistics*, pages 50–57.

Baum, Leonard E. and J.E. Eagon. 1967. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model of ecology. *Bulletins of the American Mathematical Society*, 73:360–363.

Baum, Leonard E. and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 37:1554–1563.

Bellare, Kedar. 2005. Personal communication.

Berger, A. 1997. The improved iterative scaling algorithm: A gentle introduction.

Berger, Adam L., Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Bertsekas, D.P., A. Nedic, and A.E.O Daglar. 2003. *Convex Analysis and Optimization*. Athena Scientific.

Beygelzimer, Alina, Varsha Dani, Tom Hayes, John Langford, and Bianca Zadrozny. 2005. Error limiting reductions between classification tasks. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Bikel, Dan. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*. To appear.

Bikel, Daniel, Richard Schwartz, and Ralph Weischedel. 1999. An algorithm that learns what's in a name. *Machine Learning (ML)*, 34, February.

Bilmes, Jeff. 1997. A gentle tutorial on the EM algorithm and its application to parameter estimation for gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-021, ICSI.

Bishop, Christopher M. 1995. *Neural Networks for Pattern Recognition*. Claredon Press.

Blum, Avrim, Prasad Chalasani, Sally A. Goldman, and Donna K. Slonim. 1998. Learning with unreliable boundary queries. *Journal of Computer and System Sciences*, 56(2):209–222.

Boser, B.E., I.M. Guyon, and V. Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the Conference on Computational Learning Theory (COLT)*.

Bottou, Léon. 1991. *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. Ph.D. thesis, Université de Paris XI, Orsay, France.

Boyan, Justin and Andrew W. Moore. 1996. Learning evaluation functions for large acyclic domains. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Brill, Eric. 1995. Transformation-based error-driven learning and natural language processing: a case study in part of speech tagging. *Computational Linguistics*, December.

Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Burges, Christopher J. C. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167.

Caruana, Rich. 1997. Multitask learning: A knowledge-based source of inductive bias. *Machine Learning (ML)*, 28:41–75.

Caruana, Rich. 2000. Learning from imbalanced data: Rank metrics and extra tasks. In *Proceedings of the AAAI Workshop on Learning from Imbalanced Data Sets*.

Chang, Chih-Chung and Chih-Jen Lin, 2001. *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Chawla, Nitesh, Kevin W. Bowyer, Lawrence O. Hall, and W. Phillip Kegelmeyer. 2002. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research (JAIR)*, 16:321–357.

Chen, S. and R. Rosenfeld. 1999. A gaussian prior for smoothing maximum entropy models.

Chiang, David. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Chinchor, Nancy A., editor. 1997. *Seventh Message Understanding Conference (MUC-7)*. DARPA Tipster Text Program, Morgan Kaufmann Publishers.

Christianini, Nello and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, London.

Ciaramita, Massimiliano and Yasemin Altun. 2005. Named-entity recognition in novel domains with external lexical knowledge. In *NIPS Workshop on Advances in Structured Learning for Text and Speech Processing*.

Clark, Stephen and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 103–110.

Cohen, William. 1995. Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 115–123.

Cohen, William and Jacob Richman. 2002. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD*.

Cohn, David, Zoubin Ghahramani, and Michael Jordan. 1996. Active learning with statistical models. *Journal of Artificial Intelligence Research (JAIR)*, 4:129–145.

Collins, Michael. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning (ICML–2000)*, Stanford University, Palo Alto, CA, June 29–July 2.

Collins, Michael. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Collins, Michael. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4), December.

Collins, Michael and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Collobert, Ronan and Samy Bengio. 2004. Links between perceptrons, MLPs and SVMs. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Cox, Richard T. 2001. *Algebra of Probable Inference*. The Johns Hopkins University Press, December.

CPLEX Optimization, Inc. 1994. Cplex.

Dang, Hoa, editor. 2005. *Fifth Document Understanding Conference (DUC-2005)*, Ann Arbor, MI, June.

Daumé III, Hal. 2004a. From zero to reproducing kernel hilbert spaces in twelve pages or less. Available from `http://pub.hal3.name/#daume04rkhs`.

Daumé III, Hal. 2004b. Notes on CG and LM-BFGS optimization of logistic regression. Paper available at `http://pub.hal3.name/#daume04cg-bfgs`, implementation available at `http://hal3.name/megam/`, August.

Daumé III, Hal, John Langford, and Daniel Marcu. 2005. Search-based structured prediction as classification. In *NIPS Workshop on Advances in Structured Learning for Text and Speech Processing*, Whistler, Canada.

Daumé III, Hal and Daniel Marcu. 2002. A noisy-channel model for document compression. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 449–456.

Daumé III, Hal and Daniel Marcu. 2005a. A Bayesian model for supervised clustering with the Dirichlet process prior. *Journal of Machine Learning Research (JMLR)*, To appear.

Daumé III, Hal and Daniel Marcu. 2005b. Bayesian summarization at DUC and a suggestion for extrinsic evaluation. In *Document Understanding Conference*.

Daumé III, Hal and Daniel Marcu. 2005c. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Daumé III, Hal and Daniel Marcu. 2006. Bayesian query-focused summarization. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, Sydney, Australia.

Della Pietra, Steven, Vincent J. Della Pietra, and John D. Lafferty. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393.

Dempster, A.P., N.M. Laird, and D.B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B39.

Doddington, George. 2004a. The ACE 2004 evaluation plan. `http://www.nist.gov/speech/tests/ace/ace04/doc/ace04-evalplan-v7.pdf`.

Doddington, George, editor. 2004b. *NIST Machine Translation Evaluation*, Alexandria, Virginia.

Dolan, Bill, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Dunning, Ted. 1993. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1).

Fellbaum, Christiane, editor. 1998. *Wordnet: An Electronic Lexical Database*. The MIT Press, Cambridge, MA.

Finley, Thomas and Thorsten Joachims. 2005. Supervised clustering with support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, July.

Fleischman, Michael, Eduard Hovy, and Abdessamad Echihabi. 2003. Offline strategies for online question answering: Answering questions before they are asked. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, Sapporo, Japan, July.

Florian, R., H. Hassan, A. Ittycheriah, H. Jing, N. Kambhatla, X. Luo, N. Nicolov, and S. Roukos. 2004. A statistical model for multilingual entity detection and tracking. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*.

Freund, Y. and R.E. Shapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning (ML)*, 37(3):277–296.

Galley, Michel, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, , and Ignacio Thayer. 2006. Scalable inferences and training of context-rich syntax translation models. In *Proceedings of the Joint International Conference on Computational Linguistics and Association of Computational Linguistics (COLING/ACL).*

Gentile, Claudio. 2001. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research (JMLR)*, 2:213–242, December.

Germann, Ulrich, Mike Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2003. Fast decoding and optimal decoding for machine translation. *Artificial Intelligence*, 154(1-2):127–143.

Giménez, Jesús and Lluís Màrquez. 2004. SVMTool: A general POS tagger generator based on support vector machines. In *Proceedings of the 4th LREC.*

Goodman, Joshua. 2004. Exponential priors for maximum entropy models. In *Proceedings of the Conference on Human Language Technology and the North American Chapter of the Association of Computational Linguistics.*

Greiner, Russell, Adam J. Grove, and Dan Roth. 1996. Learning active classifiers. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 207–214.

Grishman, Ralph and Beth Sundheim, editors. 1995. *Sixth Message Understanding Conference (MUC-6).* DARPA Tipster Text Program, Morgan Kaufmann Publishers, November.

Halliday, M.A. and Ruqaiya Hasan. 1976. *Cohesion in English.* Number 9 in English Language Series. Longman Publishing Group, 1 July.

Harabagiu, Sanda, Razvan Bunescu, and Steven Maiorano. 2001. Text and knowledge mining for coreference resolution. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 55–62.

Hobbs, Jerry. 1976. Pronoun resolution. Technical Report 76-1, Department of Computer Sciences, City College, City University of New York, August.

Hovy, Eduard, Chin-Yew Lin, Liang Zhou, and Junichi Fukumoto. 2006. Automated summarization evaluation with basic elements. In *Proceedings of the Fifth Conference on Language Resources and Evaluation (LREC).*

Huang, Liang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT).*

Iida, R., K. Inui, H. Takamura, and Y. Matsumoto. 2003. Incorporating contextual cues in trainable models for coreference resolution. In *Proceedings of the Workshop on the Computational Treatment of Anaphora at EACL.*

James, Gareth and Trevor Hastie. 1998. The error coding method and PICTs. *Journal of Computational and Graphical Statistics*, 7:377–387.

Jaro, Matthew A. 1989. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association (JASA)*, 84:414–420.

Jaro, Matthew A. 1995. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14:491–498.

Jebara, Tony and Alex Pentland. 1998. Maximum conditional likelihood via bound maximization and the CEM algorithm. In *Advances in Neural Information Processing Systems (NIPS)*.

Jensen, J.L.W.V. 1906. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30:175–193.

Ji, Heng and Ralph Grishman. 2004. Applying coreference to improve name recognition. In *ACL Workshop on Reference Resolution and its Applications*.

Ji, Heng, Cynthia Rudin, and Ralph Grishman. 2006. Re-ranking algorithms for name tagging. In *In Proceedings of HLT/NAACL Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*.

Kääriäinen, Matti. 2006. Lower bounds for reductions. Talk at the Atomic Learning Workshop (TTI-C), March.

Kakade, Sham and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Karakoulas, Grigoris and John Shawe-Taylor. 1999. Optimizing classifiers for imbalanced training sets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 253–259.

Kasami, Tadao. 1965. An efficient recognition and syntax algorithm for context-free languages. Technical report, Air Force Cambridge Research Laboratory.

Kassel, Robert. 1995. *A Comparison of Approaches to On-line Handwritten Character Recognition*. Ph.D. thesis, Massachusetts Institute of Technology, Spoken Language Systems Group.

Kearns, Michael and Umesh Vazirani. 1997. *An Introduction to Computational Learning Theory*. The MIT Press, Cambridge, Massachusetts, second edition.

Kehler, Andrew, Douglas Appelt, Lara Taylor, and Aleksandr Simma. 2004. The (non)utility of predicate-argument frequencies for pronoun interpretation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*.

Kehler, Andy. 1997. Probabilistic coreference in information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 163–173.

Klein, Dan and Chris Manning. 2003a. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT).*

Klein, Dan and Christopher Manning. 2003b. Accurate unlexicalized parsing. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL).*

Knight, Kevin. 1999. A statistical MT tutorial workbook. Unpublished, August.

Knight, Kevin and Daniel Marcu. 2002. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1).

Koehn, Philipp, Franz Joseph Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*, Vancouver, Canada.

Koo, Terry and Michael Collins. 2005. Hidden-variable models for discriminative reranking. In *Proceedings of the Joint Conference on Human Language Technology Conference and Empirical Methods in Natural Language Processing (HLT/EMNLP).*

Kudo, Taku and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL).*

Kudo, Taku and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL).*

Kwek, Stephen S. 2001. Learning intermediate concepts. *Lecture Notes in Computer Science*, 2225, January.

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

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

Langford, John and John Shawe-Taylor. 2002. Pac-bayes & margins. In *Advances in Neural Information Processing Systems (NIPS).*

Langford, John and Bianca Zadrozny. 2003. Reducing t-step reinforcement learning to classification. In *Proceedings of the Machine Learning Reductions Workshop.*

Langford, John and Bianca Zadrozny. 2005. Relating reinforcement learning performance to classification performance. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Lebanon, Guy and John Lafferty. 2002. Boosting and maximum likelihood for exponential models. In *Advances in Neural Information Processing Systems (NIPS)*.

Lewis, David. 2001. Applying support vector machines to the TREC-2001 batch filtering and routing tasks. In *Proceedings of the Conference on Research and Developments in Information Retrieval (SIGIR)*.

Liang, Percy, Alexander Bouchard-Côté, Dan Klein, and Ben Taskar. 2006. An end-to-end discriminative approach to machine translation. In *Proceedings of the Joint International Conference on Computational Linguistics and Association of Computational Linguistics (COLING/ACL)*.

Lin, Chin-Yew and Eduard Hovy. 2002. From single to multi-document summarization: A prototype system and its evaluation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, July.

Lin, Chin-Yew and Eduard Hovy. 2003. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*, Edmonton, Canada, May 27 – June 1.

Luo, X., A. Ittycheriah, H. Jing, N. Kambhatla, and S. Roukos. 2004. A mention-synchonous coreference resolution algorithm based on the Bell tree. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 136–143.

Luo, Xiaoqiang. 2005. On coreference resolution performance metrics. In *Proceedings of the Joint Conference on Human Language Technology Conference and Empirical Methods in Natural Language Processing (HLT/EMNLP)*.

Malouf, Robert. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of CoNLL*.

Manning, Christopher and Hinrich Schutze. 2000. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA.

Matsuzaki, Takuya, Yusuke Miyao, and Junichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

McAllester, David. 2003. Simplified PAC-Bayesian margin bounds. In *Proceedings of the Conference on Computational Learning Theory (COLT)*.

McAllester, David. 2004. Relating training algorithms to generalization bounds in structured classification. NIPS Workshop on Learning with Structured Outputs.

McAllester, David, Michael Collins, and Fernando Pereira. 2004. Case-factor diagrams for structured probabilistic modeling. In *Proceedings of the Converence on Uncertainty in Artificial Intelligence (UAI)*.

McCallum, Andrew, Kedar Bellare, and Fernando Pereira. 2005. A conditional random field for discriminatively-trained finite-state string edit distance. In *Proceedings of the Converence on Uncertainty in Artificial Intelligence (UAI)*.

McCallum, Andrew, Dayne Freitag, and Fernando Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the International Conference on Machine Learning (ICML)*.

McCallum, Andrew and Ben Wellner. 2004. Conditional models of identity uncertainty with application to noun coreference. In *Advances in Neural Information Processing Systems (NIPS)*.

McCarthy, J. and W. Lehnert. 1995. Using decision trees for coreference resolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (AJCAI)*, pages 1050–1055.

McDonald, Ryan. 2006a. Discriminative sentence compression with soft syntactic constraints. In *Proceedings of the Conference of the European Association for Computational Linguistics (EACL)*.

McDonald, Ryan. 2006b. Personal communication, 8 May.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2004. Large margin online learning algorithms for scalable structured classification. In *NIPS Workshop on Learning with Structured Outputs*.

McDonald, Ryan, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency parsing with a two-stage discriminative parser. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*.

McDonald, Ryan and Fernando Pereira. 2005. Identifying gene and protein mentions in text using conditional random fields. *BMC Bioinformatics*, 6(Suppl 1).

Megiddo, N. and C.H. Papadimitriou. 1991. A note on total functions, existence theorems and complexity. *Theoretical Computer Science*, 81:317–324.

Melamed, I. Dan. 2004. Statistical machine translation by parsing. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Minka, Thomas P. 2001. Algorithms for maximum-likelihood logistic regression. Technical Report 758, Carnegie Mellon University.

Minka, Thomas P. 2003. A comparison of numerical optimizers for logistic regression. `http://www.stat.cmu.edu/~minka/papers/logreg/`.

Morris, J. and G. Hirst. 1991. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, 17(1):21–48.

Morton, T. 2000. Coreference for NLP applications. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL).*

Munteanu, Dragos Stefan and Daniel Marcu. 2005. Improving machine translation performance by exploiting non-parallel corpora. *Computational Linguistics*, 31(4), December.

Musicant, David, Vipin Kumar, and Aysel Ozgur. 2003. Optimizing F-measure with support vector machines. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, pages 356–360.

Nash, S.G. and J. Nocedal. 1991. A numerical study of the limited memory BFGS method and the truncated Newton method for large scale optimization. *SIAM Journal of Optimization*, 1:358–372.

Ng, Andrew and Michael Jordan. 2000. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Converence on Uncertainty in Artificial Intelligence (UAI).*

Ng, Vincent and Claire Cardie. 2002. Improving machine learning approaches to coreference resolution. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL).*

Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. In *In Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Och, Franz Josef. 1999. *Statistical Machine Translation: From Single-Word Models to Alignment Templates.* Ph.D. thesis, RWTH Aachen University.

Och, Franz Josef. 2003. Minimum error rate training for statistical machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, Sapporo, Japan, July.

Och, Franz Josef and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Och, Franz Josef, Richard Zens, and Hermann Ney. 2003. Efficient search for interactive statistical machine translation. In *Proceedings of EACL*, pages 287–293, Budapest, Hungary, April.

Oren Glickman, Ido Dagan, Moshe Koppel. 2005. A probabilistic classification approach for lexical textual entailment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI).*

Pal, Chris, Charles Sutton, and Andrew McCallum. 2006. Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).*

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 311–318.

Pearl, Judea. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, London.

Platt, John. 1999. Using analytic QP and sparseness to speed training of support vector machines. In *Advances in Neural Information Processing Systems (NIPS)*, pages 557–563.

Porter, M.F. 1980. An algorithm for suffix stripping. *Program*, 14:130–137.

Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. 2002. *Numerical Recipes in C*. Cambridge University Press, second edition, January.

Punyakanok, Vasin and Dan Roth. 2001. The use of classifiers in sequential inference. In *Advances in Neural Information Processing Systems (NIPS)*.

Punyakanok, Vasin, Dan Roth, and Wen-Tau Yih. 2005. The necessity of syntactic parsing for semantic role labeling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1117–1123.

Punyakanok, Vasin, Dan Roth, Wen-Tau Yih, and Dav Zimak. 2005. Learning and inference over constrained output. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1124–1129.

Quinlan, J.R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Quirk, Chris, Arul Menezes, and Collin Cherry. 2005. Dependency tree translation: Syntactically informed phrasal SMT. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Raina, Rajat, Andrew Ng, and Christopher Manning. 2005. Robust textual inference via learning and abductive reasoning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Ramshaw, Lance A. and Michell P. Marcus. 1995a. Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*. Association for Computational Linguistics.

Ramshaw, Lance A. and Mitchell P. Marcus. 1995b. Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*.

Ratliff, Nathan, J. Andrew Bagnell, and Martin Zinkevich. 2006. Subgradient methods for maximum margin structured learning. In *Workshop on Learning in Structured Outputs Spaces at ICML*.

Ravichandran, Deepak, Eduard Hovy, and Franz Josef Och. 2003. Statistical qa - classifier vs re-ranker: What's the difference? In *In Proceedings of the ACL Workshop on Multilingual Summarization and Question Answering–Machine Learning and Beyond*, Sapporo, Japan.

Ravichandran, Deepak, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and NLP: Using locality sensitive hash functions for high speed noun clustering. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Riezler, Stefan, Tracy Holloway King, Richard Crouch, and Annie Zaenen. 2003. Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for lexical-functional grammar. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*.

Riezler, Stefan, Tracy Holloway King, Ronald Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Rosenblatt, Frank. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408. Reprinted in *Neurocomputing* (MIT Press, 1998).

Roy, Nicholas and Andrew McCallum. 2001. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Russell, Stuart and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey.

Sagae, Kenji and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT 2005)*, Vancouver, Canada.

Sarawagi, Sunita and William Cohen. 2004. Semi-Markov conditional random fields for information extraction. In *Advances in Neural Information Processing Systems (NIPS)*.

Schapire, Robert. 2003. The boosting approach to machine learning: An overview. In D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, and B. Yu, editors, *Nonlinear Estimation and Classification*. Springer.

Schraudolph, Nicol and Thore Graepel. 2003. Combining conjugate direction methods with stochastic approximation of gradients. In *Proceedings of the 9th International Conference on Artificial Intelligence and Statistics*.

Sha, Fei and Fernando Pereira. 2002. Shallow parsing with conditional random fields. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*.

Shen, Libin, Anoop Sarkar, and Franz Josef Och. 2004. Discriminative reranking for machine translation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*.

Singh, Satinder. 1993. An (almost) tutorial on reinforcement learning. Unpublished.

Smyth, Padhraic, David Heckerman, and Michael I. Jordan. 2001. Probabilistic independence networks for hidden Markov probability models. In Michael I. Jordan and Terrence J. Sejnowski, editors, *Graphical Models: Foundations of Neural Computation*. The MIT Press, Cambridge, Massachusetts, chapter 1, pages 1–44.

Soon, Wee Meng, Hwee Tou Ng, and Daniel Chung Yong Lim. 2001. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521 – 544.

Stairmand, M.A. 1996. *A Computational Analysis of Lexical Cohesion with applications in Information Retrieval*. Ph.D. thesis, University of Manchester Institute of Science and Technology.

Strube, M. and C. Müller. 2003. A machine learning approach to pronoun resolution in spoken dialog. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 168–175.

Strube, M., S. Rapp, and C. Müller. 2002. The influence of minimum edit distance on reference resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 312–319.

Stuckman, Jeff and Guo-Qiang Zhang. 2005. Mastermind is NP-complete. arXiv cs.CC/0512049.

Sutton, Charles and Andrew McCallum. 2004. Collective segmentation and labeling of distant entities in information extraction. In *ICML workshop on Statistical Relational Learning*.

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

Sutton, Charles, Khashayar Rohanimanesh, and Andrew McCallum. 2004. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 783–790.

Sutton, Charles, Michael Sindelar, and Andrew McCallum. 2005. Feature bagging: Preventing weight undertraining in structured discriminative learning. Technical Report IR-402, University of Massachusetts, Center for Intelligent Information Retrieval.

Sutton, Richard and Andrew Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Taskar, Ben, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: A large margin approach. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 897–904.

Taskar, Ben, Carlos Guestrin, and Daphne Koller. 2003. Max-margin Markov networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Teufel, Simone and Mark Moens. 1997. Sentence extraction as a classification task. In *In ACL/EACL-97 Workshop on Intelligent and Scalable Text Summarization*, pages 58–65.

Toutanova, Kristina, Aria Haghighi, and Christopher Manning. 2005. Joint learning improves semantic role labeling. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 589–596.

Tsochantaridis, Ioannis, Thomas Hofmann, Thorsten Joachims, and Yasmine Altun. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6:1453–1484, Sep.

Turian, Joseph and I. Dan Melamed. 2006. Advances in discriminative parsing. In *Proceedings of the Joint International Conference on Computational Linguistics and Association of Computational Linguistics (COLING/ACL)*.

Turner, Jenine and Eugene Charniak. 2005. Supervised and unsupervised learning for sentence compression. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Valiant, Leslie G. 1994. A theory of the learnable. *Annual ACM Symposium on Theory of Computing*, pages 436–445.

Vapnik, Vladmir. 1979. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka. (English translation: Springer Verlag, New York, 1982).

Vapnik, Vladmir N. 1995. *The Nature of Statistical Learning Theory*. Springer.

Vishwanathan, S.V.N., Nicol N. Schraudolph, Mark W. Schmidt, and Kevin Murphy. 2006. Accelerated training of conditional random fields with stochastic meta-descent. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Viterbi, Andrew J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2), April.

Wainwright, Martin. 2006. Estimating the "wrong" graphical model: Benefits in the computation-limited setting. Technical report, University of California Berkeley, Department of Statistics, February.

Wallach, Hanna. 2004. Conditional random fields: An introduction. Technical report, University of Pennsylvania, Deptartment of Computer and Information Science.

Wellner, Ben, Andrew McCallum, Fuchun Peng, and Michael Hay. 2004. An integrated, conditional model of information extraction and coreference with application to citation matching. In *Proceedings of the Converence on Uncertainty in Artificial Intelligence (UAI)*.

Wen, Tong, Alan Edelman, and David Gorsich. 2003. A fast projected conjugate gradient algorithm for training support vector machines. *AMS Contemporary Mathematics*, 323:245–263.

Weston, Jason, Olivier Chapelle, Andre Elisseeff, Bernhard Schoelkopf, and Vladimir Vapnik. 2002. Kernel dependency estimation. In *Advances in Neural Information Processing Systems (NIPS)*.

Yamada, Kenji and Kevin Knight. 2002. A decoder for syntax-based statistical MT. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Yang, X., G.D. Zhou, J. Su, and C.L. Tan. 2003. Coreference resolution using competitive learning approach. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 176–183.

Ye, Shiren, Long Qiu, Tat-Seng Chua, and Min-Yen Kan. 2005. NUS at DUC 2005: Understanding documents via concept links. In *Document Understanding Conference*.

Yedidia, Jonathan, William Freeman, and Yair Weiss. 2003. Understanding belief propagation and its generalizations. In *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann Publishers, pages 239–269.

Younger, Daniel H. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Zadrozny, Bianca, John Langford, and Naoki Abe. 2003. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the IEEE Conference on Data Mining (ICMD)*.

Zajic, David, Bonnie Dorr, and Richard Schwartz. 2004. BBN/UMD at DUC-2004: Topiary. In *Proceedings of the Fourth Document Understanding Conference (DUC 2004)*, Boston, MA, May 6 – 7.

Zhang, Tong. 2002. Covering number bounds of certain regularized linear function classes. *Journal of Machine Learning Research (JMLR)*, 2:527–550.

Zhang, Tong. 2006. Personal communication, June.

Zhang, Tong, Fred Damerau, and David Johnson. 2002. Text chunking based on a generalization of Winnow. *Journal of Machine Learning Research (JMLR)*, 2:615–637, March.

Zinkevich, Martin. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Appendix A

Summary of Notation

There is a some amount of notation used in this thesis that might be unfamiliar to some readers. This appendix summarizes most of the symbols used.

A.1 Common Sets and Functions

$\boldsymbol{w}$	The weight vector; typically this is a vector in $\mathbb{R}^F$.
$\mathcal{F}$	The feature space; typically this is $\mathbb{R}^D$.
D	The dimensionality of the feature space.
$\mathcal{X}$	The "input space:" the input to a learning system.
$\mathcal{Y}$	The "output space:" the output of a learning system.
$\mathbb{R}$	The real numbers, $(-\infty, \infty)$.
$\mathbb{R}^+$	The non-negative real numbers, $[0, \infty)$.
$\mathbb{N}$	The natural numbers, $\{0, 1, 2, \dots\}$, which *always* include zero.
Φ	A function that takes an input and produces a feature vector; typically we either use $\boldsymbol{\Phi}(x)$ or $\boldsymbol{\Phi}(x, y)$ for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, though not always. This function produces a vector in the feature space $\mathcal{F}$.
$\mathcal{D}$	A probability distribution over examples for learning.
$\mathcal{Z}$	The hidden variable space.
ℓ	A cost-sensitive loss.
γ	The margin for a linear classifier.
π	A policy.
π^*	The optimal policy.
$\oplus$	Vector concatenation.

A.2 Vectors, Matrices and Sums

To reduce the number of sums required, I make heavy use of standard vector and matrix notation. If this notation is unfamiliar, it might take a bit of getting used to, but it makes equations much cleaner. The most common notation used is $\boldsymbol{w}^\top$
$Ph(x)$, for instance. The $^\top$ operator denotes transpose and flips a column vector (in this case, $\boldsymbol{w}$) into a row vector, so that when multiplied with another vector (here, $\Phi(x)$), one obtains a real number. This is, therefore, shorthand for $\sum_{i=1}^{I} w_i \Phi(x)_i$. Vectors are always written italicized in bold and matrices are always written non-italicized in bold;

for instance $\mathbf{A}$. Furthermore, vectors are typically represented with lower case letters (with the exception of $\mathbf{\Phi}(\cdot)$), while upper case letters are reserved for matrices.

When sums are necessary, I maintain the convention that the upper bound and index variable of the sum are the same letter. For instance, I write $\sum_i$ to mean $\sum_{i=1}^{I}$, when it is clear from context that i corresponds to a sum up to I.

The l_p norm of a vector is given by:

$$||\boldsymbol{u}||_p = \left[\sum_i |u_i|^p\right]^{1/p} \tag{A.1}$$

With a slight abuse of notation, I refer to the l_0 norm of $\boldsymbol{u}$ to be the number of non-zero elements of $\boldsymbol{u}$: $||\boldsymbol{u}||_0 = \sum_i \delta_{u_i \neq 0}$; and the infinity norm as the size of the maximal element: $||\boldsymbol{u}||_\infty = \max_i |u_i|$. The "dual norm" to l_p is l_q, where q is such that $(1/p + 1/q)^{-1} = 1$; l_2 is self-dual, and the dual of l_1 is l_∞ and vice-versa.

A.3 Complexity Classes

There are three important complexity classes for the purpose of this thesis: functional polynomial (FP), total function non-deterministic polynomial (TFNP) and functional non-deterministic polynomial (FNP). These are related to the standard polynomial and non-deterministic polynomial (P and NP) classes familiar to most. The key difference between the functional classes and the standard classes is that the standard classes are concerned with decision problems: the response is binary. The functional classes are concerned with computing complex functions.

In the case of the function variants, it turns out it is easier to define FNP first. FNP is the class of function problems of the following form: Given an input X and a polynomial-time predicate $F(X, Y)$, if there exists a Y satisfying $F(X, Y)$ then output *any* such Y; otherwise, output "no." FP is the subclass of FNP that has a polynomial time solution. More interesting to use is the class TFNP (total function NP), which is defined identically to FNP with the exception that we are guaranteed that there exists a Y such that $F(X, Y)$ holds (and thus we must never output "no"). It is easy to verify that FP$\subseteq$TFNP$\subseteq$FNP, with equality if and only if P=NP. See (Megiddo and Papadimitriou, 1991) for more information. Any time I discuss computational complexity issues in this thesis, I will use the notions of FP and TFNP to maintain clarity. I will also always go under the assumption that P$\neq$NP (and hence FP$\neq$TFNP).

Appendix B

Proofs of Theorems

This appendix contains the proofs for most of the theoretical claims made in this thesis.

Section 3.5

Proof (Lemma 3.5: Policy Degradation). The proof largely follows the proofs of Lem 6.1 and Theorem 4.1 in CPI (Kakade and Langford, 2002). The three differences are that (1) we must deal with the finite horizon case; (2) we move *away from* rather than *toward* a good policy; and (3) we expand to higher order.

The proof works by separating three cases depending on whether h^{CS} or h is called in the process of running h^{new}. The easiest case is when h^{CS} is never called. The second case is when it is called exactly once. The final case is when it is called more than once. Denote these three events by $c = 0$, $c = 1$ and $c \geq 2$, respectively.

$$\begin{aligned}
L(\mathcal{D}, h^{\mathrm{new}}) =& Pr(c=0) L(\mathcal{D}, h^{\mathrm{new}} \mid c=0) \\
&+ Pr(c=1) L(\mathcal{D}, h^{\mathrm{new}} \mid c=1) \\
&+ Pr(c \geq 2) L(\mathcal{D}, h^{\mathrm{new}} \mid c \geq 2)
\end{aligned} \tag{B.1}$$

$$\begin{aligned}
\leq& (1-\beta)^T L(\mathcal{D}, h) + T\beta(1-\beta)^{T-1} \left(L(\mathcal{D}, h) + \ell_h^{\mathrm{CS}}(h') \right) \\
&+ \left(1 - (1-\beta)^T - T\beta(1-\beta)^{T-1}\right) c_{\max}
\end{aligned} \tag{B.2}$$

$$\begin{aligned}
=& L(\mathcal{D}, h) + T\beta(1-\beta)^{T-1} \ell_h^{\mathrm{CS}}(h') + \left(\sum_{i=2}^{T} (-1)^i \beta^i \binom{T}{i} \right) L(\mathcal{D}, h) \\
&+ \left(1 - (1-\beta)^T - T\beta(1-\beta)^{T-1}\right) c_{\max}
\end{aligned} \tag{B.3}$$

$$\begin{aligned}
\leq& L(\mathcal{D}, h) + T\beta \ell_h^{\mathrm{CS}}(h') \\
&+ \left(1 - (1-\beta)^T - T\beta(1-\beta)^{T-1}\right) (c_{\max} - L(\mathcal{D}, h))
\end{aligned} \tag{B.4}$$

$$\leq L(\mathcal{D}, h) + T\beta \ell_h^{\mathrm{CS}}(h') + \left(1 - (1-\beta)^T - T\beta(1-\beta)^{T-1}\right) c_{\max} \tag{B.5}$$

$$=L(\mathcal{D},h) + T\beta\ell_h^{\text{CS}}(h') + \left(\sum_{i=2}^{T}(-1)^i\beta^i\binom{T}{i}\right)c_{\max} \tag{B.6}$$

$$\leq L(\mathcal{D},h) + T\beta\ell_h^{\text{CS}}(h') + \frac{1}{2}T^2\beta^2 c_{\max} \tag{B.7}$$

The first inequality is by bounding the probabilities of each event and the corresponding losses. The second is by the assumption that the cost-sensitive regret is negative (we are moving away from the optimal policy). The third uses the assumption that $\beta < T/2$. Others are by algebra. □

Proof (Theorem 3.6: Convergence). The proof involves invoking Lemma 3.5 repeatedly. After C/β iterations, we can verify that:

$$L(\mathcal{D},h) \leq L(\mathcal{D},h_0) + CT\ell_{\text{avg}} + c_{\max}\left(\frac{1}{2}CT^2\beta\right)$$

Last, if we call the optimal policy, we fail with loss at most $c_{\max}$. The probability of failure after C/β iterations is at most $T(1-\beta)^{C/\beta} \leq T\exp[-C]$. □

Section 3.6.1

Proof (Theorem 3.8). For the first part, we use a vector encoding of y that maintains the decomposition over regions. Given a prefix $y_1, \dots, y_i$, solve *opt* on the future choices, which gives us an optimal policy.

For the second part, we simply make Φ complex: for instance, include long-range dependencies in sequence labeling. At the extreme, for non-zero $\boldsymbol{w}$, this means computing a minimum-energy configuration of a fully-connected Boltzmann machine, which is hard. □

Appendix C

Relevant Publications

Some of the material in this thesis has appeared in the proceedings of natural language processing and machine learning conferences. The first paper relevant to this thesis, though it has been completely subsumed at this point, was the introduction of the "Learning as Search Optimization" framework. This appeared at ICML 2005:

> Hal Daumé III and Daniel Marcu. (2005). Learning as search optimization: Approximate large margin methods for structured prediction. *Proceedings of the International Conference on Machine Learning (ICML).* Bonn, Germany. pp 169–176.

The development of the entity detection and tracking system described in Chapter 5 of this thesis was introduced (using an older version of the learning framework) at HLT/EMNLP 2005:

> Hal Daumé III and Daniel Marcu. (2005). A Large-Scale Exploration of Effective Global Features for a Joint Entity Detection and Tracking Model. *Proceedings of the Joint Conference on Human Language Technology Conference and Empirical Methods in Natural Language Processing (HLT/EMNLP).* Vancouver, Canada. pp 97–104.

The beginnings of the new version of the learning algorithm were introduced in a paper Daniel and I coauthored with John Langford that appeared at a workshop at NIPS:

> Hal Daumé III, John Langford and Daniel Marcu. (2005). Search-Based Structured Prediction as Classification. *Advances in Structured Learning for Text and Speech Processing, Workshop at the Conference on Neural Information Processing Systems (NIPS).* Whistler, Canada.

Finally, the SEARN algorithm, which built on both the above ICML and NIPS workshop papers, is submitted to NIPS 2006:

> Hal Daumé III, John Langford and Daniel Marcu. (2006, under review). Search-based Structured Prediction. *Proceedings of the Conference on Neural Information Processing Systems (NIPS).* Vancouver, Canada.