MACHINE LEARNING FOR INFORMATION EXTRACTION

BY

DMITRY ZELENKO

M.S., University of Illinois at Urbana-Champaign, 1997
M.S., University of Illinois at Urbana-Champaign, 1998

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2003

Urbana, Illinois

Table of Contents

Chapter 1

Introduction

Information extraction is the process of converting a natural language text into a structured representation. The structured representation reflects concepts expressed by the text, and relationships that hold among the concepts. Examples of possible extracted concepts are *people*, *organizations*, *places*, *times* and *dates*, *money amounts* and *percentage changes* in financial publications, university *courses* and student *grades*. The extracted concepts can be related; examples of frequent relations include the *affiliation* relation, which specifies that a *person* is affiliated with an *organization*, a *location* relation between a *person*/*organization* and a *place*, and *social* relations between *people*.

Information extraction can be seen as a process of decoding the natural language text to recover the original (conceptual) representation that the underlying text aims to express. By recovering all concepts and relations represented by the text and refining the conceptual vocabulary to account for finer nuances of natural language, information extraction approaches, in the limit, the problem of *natural language understanding*. While understanding itself is an elusive goal, we can hypothesize that the mechanism for representing natural language in a structural form and using the representation in performing further inferences will play an crucial role in the ability of intelligent machines to pass an operational test of natural language understanding.

Less ambitious applications of information extraction abound. A structured representation of natural language documents can be readily stored in a relational database, and provide the basis for *question answering* applications [88]. It can also be integrated with other structured data, and used as part of *data mining* [52], the process of discovering new and interesting nuggets of information in a database. In many current applications, the extracted concepts and relations provide helpful text visualizations allowing users to quickly grasp essential concepts and relations of a natural language

document [70].

Information extraction applications of recent years were spurred by growth of the World Wide Web. Indeed, information extraction holds the promise of converting WWW into a huge *conceptual* database, rather than the existing database of web pages and links between them. Such a database would lead to a dramatic shift from the current *information retrieval* paradigm: keyword-based search [106].

History of information extraction is fairly recent. Starting from the late 80s, the DARPA-sponsored Message Understanding Conferences (MUC) [1, 2, 3, 4, 5] essentially created the field of information extraction. MUC was a response to increasing growth of online text that needed to be processed by human analysts. Early MUC attempted to define a set of events of interest that could be expressed in the text (e.g., vehicle launch). Each event was described by record template (e.g., what, when, where), and the goal of an information extraction system was to fill the template by extracting information from text. Later, MUC compartmentalized extraction problems and defined separate tasks for extraction of named entities, relations, coreference resolution, and others.

Experience of MUC proved that information extraction is very difficult, even for humans. For example, various aspects of the information extraction task exhibited the inter-annotator agreement of only 60-80 percent. Yet for many well-defined problems, such as named entity extraction, the best extraction systems currently achieve over 90% performance.

Early information extraction systems were manually engineered by computational linguists. The systems comprised a set of general linguistic patterns (e.g., finite state machines) and domain-dependent patterns that identified building blocks of underlying text and extracted the required information therefrom. Building information extraction systems manually is fairly arduous process that requires significant knowledge of both the language and the extraction domain. In the process, experience and skill of the knowledge engineer play a critical part. Over the 90s, manually engineered systems exhibited very good performance for the tasks of named entity extraction, relation extraction, and others [14]. The knowledge engineering paradigm is still the approach of choice for most information extraction systems that are being built at the present moment.

In the second half of the 90s, several adaptive (or learning) systems have been built for information extraction. The machine learning systems are based on annotated information extraction

data. The systems use the data to learn models that are employed to extract information from new texts. Most early learning systems for information extraction were heavily influenced by the speech processing community [55] and based on the like formalisms: variants of probabilistic modeling of the underlying text. Notable systems are Hidden Markov Model-based named entity extraction system [20] and Lexicalized Probabilistic Context Free Grammar-based parsing and relation extraction system [79]. In the late nineties, other modeling approaches were applied to information extraction including maximum entropy modeling [93] and inductive logic programming [25].

With the advent of Internet, adaptive information extraction techniques were used to extract data from web pages. Due to the semi-structured nature web pages (i.e., presence of html tags that in many cases delimit the information of interest), a community of *wrapper induction* [69, 84] appeared. Wrapper induction methods represent a variety of special purpose techniques for learning finite state machines that use formatting clues for information extraction. For relatively regular web pages, the wrapper induction methods require very few (less than 5) labeled web pages to produce wrappers (extraction models), with excellent performance. However, the techniques are not applicable to the general information extraction problem, where formatting regularities are absent.

Applications of machine learning approaches to the problem of information extraction and natural language processing, in general, pose both practical and theoretical challenges for machine learning.

From the practical perspective, the sheer amount of natural language data (tens and hundreds thousands of examples) requires development of very efficient algorithms for learning and inference processes. Furthermore, the feature vector representation of natural language data embeds the data into a very high dimensional spaces (tens and hundreds thousands on dimensions). Many classical learning algorithms are not applicable for spaces of such dimensionality. Another aspect of the feature vector representation is that despite the increased overall dimensionality, any particular example feature vector is very sparse, that is, it contains few non-zero coordinates. Most learning algorithms used in NLP applications exploit example sparsity to drastically improve their efficiency.

From the theoretical perspective, very high dimensionality of the data requires re-examination of the current generalization theory, in its applicability to NLP learning problems. In fact, it

is frequently the case with the NLP problems, that the dimensionality of the feature space far exceeds the number of training examples. Most classical learning theory results fail to hold in such circumstances (see Section 2.5). An alternative analysis of learning algorithm generalization ability was recently undertaken that provides for dimensionality independent bounds, but more work is still necessary to make the analysis applicable to NLP problems [17, 46]. We also note that nearly all theoretical analyses of learning are worst-case, i.e., they are required to hold for any distribution of the input data (the formal definition is found in Section 2.3). For NLP problems, the worst-case analysis is overly pessimistic due to presence of numerous constraints stemming from language regularities. An ability to incorporate the constraints into theoretical analyses would provide much more realistic generalization bounds [100, 47].

1.1 Overview

This section presents a brief summary of the following chapters.

In Chapter 2, we survey the field of machine learning. We introduce important statistical and computational concepts, as well as delineate a number of pertinent theoretical results and algorithms that are used in subsequent chapters.

We then leverage recent advances of machine learning to design and implement a novel information extraction system. The architecture of the system follows the common paradigm of building information extraction systems. Namely, we separate information extraction process in a sequences of tasks:

- Part of speech tagging.
- Entity extraction and coreference resolution.
- Relation extraction.

We formalize each of the tasks as a learning problem and apply learning algorithms to the problem. Below we briefly examine each of the constituent tasks and our novel contributions to their solution.

Chapter 3 addresses the problem of part of speech tagging. Part of speech tagging is the problem of identifying parts of speech of words. We formalize the task as a multiclass classification problem,

and apply the SNOW (Sparse Network of Winnows) learning system [96, 63] to learn the part of speech classifier. We perform a comprehensive experimental evaluation of the system and argue that SNOW architecture is appropriate for NLP applications. Chapter 3 reflects our joint work with Dan Roth, it previously appeared as [99].

In Chapter 4, we discuss the problem of entity extraction. The goal of entity extraction is to identify all entities mentioned in text, and classify them by types. Entity extraction is complemented by coreference resolution, which is the problem of determining whether different extracted entities correspond to the same real-world entities. We introduce a classification approach for entity extraction, and consider coreference resolution from the decoding perspective. That is, we design novel decoding algorithms that, given local coreference decisions, produce a global coherent interpretation of document entities. We experimentally evaluate algorithms for entity classification and coreference resolution using the evaluation methodology of the recent Automatic Content Extraction (ACE) program [10].

Chapter 5 addresses the problem of relation extraction. Relation extraction is the problem determining relations of interest that hold between extracted entities. We formalize relation extraction as a classification problem, and apply kernel methods to learn the relation classifiers. We design novel kernels that are defined in terms of shallow parses and give efficient algorithms for computing the kernels. We evaluate the kernel approach experimentally, with promising results. Chapter 5 is a joint work with Chinatsu Aone and Anthony Richardella, it was previously published as [111].

Finally, in Chapter 6 we combine the constituent pieces into one coherent extraction system.

Chapter 2

Machine Learning

Learning is the process of estimating unknown dependencies from observations. The concept of learning has been studied and formalized in different fields, viz., philosophy, psychology, cognitive science, statistics, pattern recognition, and computer science. In this section, we survey quantitative formalizations of the process of learning and show how they converge and give rise to the modern conceptual framework of statistical and computational learning theory.

Let X be a (measurable) set of possible observations. The underlying assumption in most learning models is that there exists a fixed unknown probability distribution P over X. The learning process receives a set S observations from X sampled independently according to P and seeks to estimate an unknown dependency from the finite sample S.

2.1 Classical Statistics

The classical (parametric) statistics [41] is concerned with estimating the form of the probability distribution P whose density function p is assumed to have an analytic parametric description $p(x) = p(x, w)$, where $w \in W$ is a vector of density parameters in some parameter space W. The number of parameters is usually assumed to be small, and the corresponding class of densities $\mathcal{P} = \{p(x, w) : w \in W\}$ is supposed to include the true underlying density. The classical statistical approach usually sidesteps the issue of selecting the class $\mathcal{P}$, and only deals with determining the values of the parameters w, given that the true density belongs to the class $\mathcal{P}$.

The most prevalent approach to parametric density estimation aims to maximize the probability of the finite sample $S = \{x_1, x_2, \ldots, x_m\}$, where x_i's are sampled independently from a fixed

unknown distribution P. Define the *likelihood function* $P(S|w)$ to be:

$$P(S|w) = \prod_{i=1}^{m} p(x_i, w) \tag{2.1}$$

We then seek $w^* \in W$ maximizing $P(S|w)$:

$$w^* = argmax_{w \in W} P(S|w) \tag{2.2}$$

The combination of (2.1) and (2.2) is termed the *maximum likelihood* approach for density estimation. It is common to replace the likelihood function $P(S|w)$ with the *log-likelihood function* $L_p(S|w)$ in order to make the optimization problem (2.2) more tractable[1]:

$$L_p(S|w) = \sum_{i=1}^{m} \log p(x_i, w) \tag{2.3}$$

Then, the optimization criterion (2.2) is obviously equivalent to

$$w^* = argmax_{w \in W} L_p(S|w) \tag{2.4}$$

2.2 Classical Pattern Recognition

The classical approach to pattern recognition leverages the conceptual framework of parametric density estimation and applies it to the problem of classification [38]. Namely, the observations in the sample S are augmented with class membership information: $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where $y_i \in Y, i = 1, 2, \ldots, m$, and $Y = \{c_1, c_2, \ldots, c_k\}$ is the set of k classes. There is a fixed unknown probability distribution $P(x, y)$ over the cross product $X \times Y = \{(x, y) \ : \ x \in X, y \in Y\}$. In the classical setting, the corresponding density function $p(x, y)$ is usually decomposed as $\sum_k p(c_k)p(x|c_k)$, and the class densities are then assumed to have simple parametric descriptions, i.e., $p(x|c_k) = p(x, w_k)$, where w_k is the parameter vector for the kth class density.

We define a misclassification *loss function* $l : Y \times Y \to \mathbb{R}$ that quantifies the misclassification

[1] Gradient computation for the log-likelihood function is often much less complex than that for the likelihood function.

error. For example, the most commonly used 0-1 loss function is defined as follows:

$$l_0(y, y') = \begin{cases} 1, & \text{if } y \neq y' \\ 0, & \text{if } y = y' \end{cases} \tag{2.5}$$

We define a classifier $c : X \to Y$ as a function mapping the set of observations X into a set of classes Y. Define the classifier *risk*(expected loss) with respect to the probability distribution P to be

$$R(c) = \int_{X \times Y} l(y, c(x)) dP(x, y) \tag{2.6}$$

If $p(x, y)$ is a density function corresponding the probability distribution P, then (2.6) can be written as

$$R(c) = \int_X \int_Y [l(y, c(x)) p(y|x) dy]\, p(x) dx \tag{2.7}$$

Since the density function $p(x)$ is never negative, the equation (2.7) implies that the risk $R(c)$ is minimized, if the integrand is minimized at each point x, that is, the optimal classifier c_b is chosen so that ([38])

$$c_b(x) = y', \text{where } y' = argmin_{y \in Y} \textstyle\sum_{y_i \in Y} l(y_i, y) p(y_i|x) \tag{2.8}$$

In particular, for the 0-1 loss function,

$$c_b(x) = y', \text{where } y' = argmax_{y \in Y} p(y|x) \tag{2.9}$$

The classifier (2.8) is termed the *Bayes optimal classifier*, and the corresponding risk $R(c_b)$ is termed the *Bayes risk*.

Given a finite sample S, the goal is to output a classifier c with the minimum expected loss with respect to the distribution $P(x, y)$. If we assume that the densities $p(x|c_1), p(x|c_2), \ldots, p(x|c_k)$ belong a common parametric family of probability distributions ($p(x|c_i) = p(x|w_i) \in \mathcal{P}$), then we can use the maximum likelihood approach for estimating $w_1, w_2, \ldots, w_k$ from the sample S. Furthermore, the prior class probabilities $p(c_i)$ can be estimated from the sample as well. Finally, the Bayes formula implies that

$$p(c_i|x) = \frac{p(x|c_i) p(c_i)}{p(x)} \tag{2.10}$$

Hence, $p(c_i|x) \simeq p(x|c_i)p(c_i)$, and the knowledge of class densities and prior probabilities uniquely determines the Bayes optimal classifier (2.8).

There are however several problems with the classical approach to classification. First, knowledge that the densities $p(x|c_i)$ belong to a certain family of parametric probability distribution is often unavailable, especially for high-dimensional data. Second, even if the exact parametric family is known, it may be very difficult to produce reliable estimates of parameters w_i, from a finite sample S.[2] Finally, it has been argued that the problem of density estimation from a finite sample is, in general, more difficult than the problem of learning a classifier c without resorting to modeling the underlying distribution [107]. These observations provide a motivation for recent developments in statistical learning theory.

2.3 Classification and Statistical Learning Theory

The statistical learning theory methodology suggests that a classification learning problem be solved directly via finding a classifier $c : X \to Y$ rather than modeling the class distributions explicitly [107, 108]. The classifier $c(w), w \in W$ is sought in a set $\mathcal{C} = \{c(w) : w \in W\}$, where W is some parameter space. The set $\mathcal{C}$ is termed the (classifier) *hypothesis space*.

Note that the Bayes optimal classifier may not, in general, belong to $\mathcal{C}$. Therefore, we seek to learn a classifier c^* that has the minimum risk with respect to $P(x, y)$ within $\mathcal{C}$:

$$c^* = c(w^*) = argmin_{c \in \mathcal{C}} R(c) \tag{2.11}$$

For a classifier $c \in \mathcal{C}$, we define the *error* of c with respect to the optimal classifier c^*:

$$error(c, c^*) = R(c) - R(c^*) \tag{2.12}$$

Let A be an algorithm that produces a classifier $c \in \mathcal{C}$ from the random sample S. We call such an algorithm a *learning algorithm*. From the practical standpoint, we need to ascertain that $error(c, c^*)$ will be small, with high probability, as the sample size m is increased. More precisely,

[2] The density estimation problem belongs, in general, to a class of ill-posed problems, which implies that presence of noise in the sample S can have a significant impact on estimation accuracy.

we seek to bound the number of training examples m (*sample complexity*) sufficient to make the error of the classifier c small, with high probability:

$$P^m\{error(c, c^*) \leq \epsilon\} > 1 - \delta \tag{2.13}$$

where P^m is the product distribution over the set of m labeled examples sampled independently according to a fixed unknown distribution P, ϵ is the required approximation of the classifier c with respect to the distribution P, and δ is the confidence of the approximation (with respect to the random sample S of length m).

A classifier satisfying (2.13) is termed *probably approximately correct* (PAC). The corresponding learning model is referred to as the *PAC model* of learning, and the learning algorithm A that produces a PAC classifier is termed a *PAC learning algorithm* for the hypothesis class $\mathcal{C}$ [105].

Note that the PAC learning model is *distribution-free*, that is, the condition (2.13) must hold for any fixed unknown distribution P. PAC bounds are therefore inherently worst-case and, in many cases, overly pessimistic, especially, for high-dimensional learning problems in natural language processing, where the class of underlying distributions is constrained.

Selection of the classifier $c(w)$ is based on the training sample S. The selection criteria that determine the classifier $c(w)$ from the sample S are termed an *inductive principle* [107]. A learning algorithm implements a particular inductive principle. We already introduced the maximum likelihood inductive principle for density estimation in Section 2.1. The maximum likelihood inductive principle states that we should select the parameters of a probability distribution that maximize the likelihood of the sample S. The maximum likelihood inductive principle is a specific instance of a general *empirical risk minimization* (ERM) inductive principle that forms a basis for statistical learning theory.

Define *empirical risk* $R(c)$ of a classifier $c(w)$ on a sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ as

$$R_{emp}(c) = \frac{1}{m} \sum_{i=1}^{m} l(y_i, c(x_i)) \tag{2.14}$$

The empirical risk minimization principle states that we should select a classifier $c^*_{emp}(w) \in \mathcal{C}$ that

minimizes the empirical risk (2.14):

$$c^*_{emp} = c(w^*_{emp}) = argmin_{c \in \mathcal{C}} R_{emp}(c) \tag{2.15}$$

The statistical learning theory addresses questions of relationship between the empirical risk R_{emp} and the true risk R. In particular, it studies convergence of the empirical risk R_{emp} to the true risk R. We will examine the convergence in Section 2.5.

2.4 Computational Learning Theory

The computational learning theory [105, 61] emphasizes the computational complexity of learning algorithms in terms of the required accuracy and confidence parameters ϵ and δ. Formally, a learning algorithm A is a polynomial *PAC learning algorithm* for a class $\mathcal{C}$, if for any distribution P, given a sample of $m = poly(\frac{1}{\epsilon}, \frac{1}{\delta})$, the algorithm A outputs a classifier c, so that

$$P^m\{error(c, c^*)\} > 1 - \delta \tag{2.16}$$

and the running time of A is polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ [60].

In many cases, there is a natural complexity parameter n associated with the domain of observations X (e.g., $X \subseteq \mathbb{R}^n$). In such cases, the learning algorithm A is also required to depend polynomially on n.

Unfortunately, for most interesting hypothesis spaces and the 0-1 loss function, the problem of learning is computationally hard. That is, existence of polynomial learning algorithms would violate widely accepted beliefs in the computational complexity theory (e.g., $NP \neq P$). Let us consider an important example of such hypothesis spaces.

Example 1 *Let $X \subseteq \mathbb{R}^n$, $Y = \{-1, 1\}$, and C_{lin} is the class of half-spaces (linear classifiers) in $\mathbb{R}^n$. For a half-space c described by a separating hyperplane $c_h(x) = w \cdot x + w_0$, $w = (w_1, w_2, \ldots, w_n) \in \mathbb{R}^n$:*

$$c(x) = sgn(c_h(x)) = sgn(w \cdot x + w_0) = \begin{cases} 1, & \text{if } w \cdot x + w_0 \geq 0 \\ -1, & \text{if } w \cdot x + w_0 < 0 \end{cases} \tag{2.17}$$

We will see in Section 2.5 that the sample complexity of learning half-spaces is linear in n and polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. However, the problem of finding the minimum empirical risk classifier in C_{lin}, for the 0-1 loss function, is computationally hard [12].

We can eschew the negative results of computational learning by restricting the class of distributions $\mathcal{P}$. For instance, the most common restriction is to assume that for any $P \in \mathcal{P}$, the optimal classifier $c^* \in \mathcal{C}$ has zero risk $R(c^*)$. Equivalently, the class of distributions $\mathcal{P}$ is functionally decomposed, i.e., $\forall P \in \mathcal{P}\ \exists c^* : X \to Y$, so that $P(x, y) = 0$, if $y \neq c^*(x)$. Most work in computational learning theory is done in this setting, and many interesting hypothesis classes are (polynomially) learnable, if $\mathcal{P}$ is functionally decomposed [61].

We can also avoid using the 0-1 loss function and replace it with a smoother upper bound that will reduce the computational complexity of the problem. A number of such smooth loss functions have been proposed, resulting in tractable learning algorithms (e.g, Support Vector Machine [33]). Moreover, the recent analysis [112] reveals that the classifiers obtained by minimizing the smooth loss functions can approximately[3] reach the error rate of the optimal classifier c^*.

2.5 Learning to Classify: Classical Results

We will now study the relationship between the empirical risk $R_{emp}(c)$ and the true risk $R(c)$, in the case of binary classifiers and the 0-1 loss function. We assume that $Y = \{-1, 1\}$. The presented results can be generalized to multi-class classifiers and arbitrary loss functions.

We will present bounds that describe the relationship between the true risk and empirical risk in terms of the number of examples and complexity of the hypothesis class $\mathcal{C}$. We now quantify complexity of a hypothesis class via a combinatorial concept that plays a very important role in the learning theory [109].

Definition 1 *Let $S = \{x_1, x_2, \ldots, x_m\}$, and $\mathcal{C}$ be a hypothesis class (of binary classifiers). Then we say that the sample S is* shattered *by $\mathcal{C}$, if $\forall (y_1, y_2, \ldots, y_m) \in \{-1, 1\}^m$, there is a $c \in \mathcal{C}$ such that $c(x_i) = y_i,\ i = 1, 2, \ldots, m$.*

[3] The distance function measuring the approximation error is determined by the underlying smooth loss function, see [112] for details.

In other words, a sample is shattered by $\mathcal{C}$, if classifiers from $\mathcal{C}$ can induce all possible classifications of the sample.

Definition 2 *The* Vapnik-Chevronenkis dimension *(VC dimension) [109] of a hypothesis class $\mathcal{C}$ ($VCD(\mathcal{C})$) is the size of largest sample shattered by $\mathcal{C}$.*

Example 2 *Let C_{lin} be a class of half-spaces in $\mathbb{R}^n$. Then, the VC dimension of the class of half spaces is equal to $n+1$. That is, there is a set of $n+1$ points that can shattered by half-spaces, but no set of $n+2$ points can be shattered [34].*

Using the concept of VC dimension, we can formulate the bounds for convergence of a classifier empirical risk to its true risk.

Theorem 1 ([108]) *Let $\mathcal{C}$ be a hypothesis space with VC dimension d. For any probability distribution P on $X \times \{-1, 1\}$ and any $c \in \mathcal{C}$, with probability $1-\delta$ over m examples sampled independently from P:*

$$R(c) \leq 2R_{emp}(c) + \frac{4}{m}\left(d \log \frac{2em}{d} + \log \frac{4}{\delta}\right) \tag{2.18}$$

provided that $d \leq m$.

For the case, when the empirical risk is zero, we have a tighter bound:

Theorem 2 ([21]) *Let $\mathcal{C}$ be a hypothesis space with VC dimension d. For any probability distribution P on $X \times \{-1, 1\}$ and any $c \in \mathcal{C}$ with zero empirical risk on m examples sampled independently from P, with probability $1-\delta$*

$$R(c) \leq \epsilon(m, \mathcal{C}, \delta) = \frac{2}{m}\left(d \log \frac{2em}{d} + \log \frac{2}{\delta}\right) \tag{2.19}$$

provided that $d \leq m$ and $m > \frac{2}{\epsilon}$.

Note that both Theorem 1 and Theorem 2 assume that the sample size is greater than the VC dimension of hypothesis class. For linear classifiers (half-spaces), which we will employ for the NLP problems, the VC dimension is roughly equal to the number of features present in data (see Example 2). For most NLP problems, the number of features exceeds the number of training examples; therefore, Theorem 1 and Theorem 2 are not applicable to the NLP domain.

The negative results for the NLP domain notwithstanding, the bounds (2.18) and (2.19) do provide insight into the nature of generalization. They show that complexity of a hypothesis class is as important for generalization as the small error on the training set. Namely, embedding a hypothesis class $\mathcal{C}_1$ into another hypothesis class $\mathcal{C}_2$ such that $VCD(\mathcal{C}_1) < VCD(\mathcal{C}_2)$ will likely reduce the empirical risk, but also increase the right hand side of (2.19) and the second summand in (2.18). Therefore, we have a trade-off between the empirical risk and hypothesis class complexity. The trade-off suggests varying complexity of a hypothesis class via a hierarchy of hypothesis classes:

$$\mathcal{C}_1 \subset \mathcal{C}_2 \subset \cdots$$

$$VCD(\mathcal{C}_1) < VCD(\mathcal{C}_2) < \cdots$$

For each hypothesis class $\mathcal{C}_i$, we determine the classifier c_i with the least empirical risk $R_{emp}(c_i)$, and minimize the bounds (2.18) or(2.19) over all c_i's. Such an approach to classifier selection (*model selection*) is termed *structural risk minimization* [108]. Note that, with the existing bounds (2.18) and (2.19), structural risk minimization is hardly applicable for practical model selection, since the bounds are extremely loose, especially for NLP applications.

Thus, dependence of the convergence bounds on VC dimension and, hence, dimensionality of the feature spaces makes the bounds impractical for high-dimensional problems. We next introduce an alternative analysis of generalization that provides for dimensionality-independent bounds.

2.6 Learning to Classify: Margin-based Results

In this section, we restrict our attention to the class of linear classifiers C_{lin}. The results can be generalized to arbitrary classes of thresholded real-valued functions.

Definition 3 *For a classifier $c \in C_{lin}$ and a labeled example (x_i, y_i), where $y_i \in \{-1, 1\}$, define the* margin *ρ_i of $c(w)$ on (x_i, y_i) as*

$$\rho_i = y_i c_h(x_i) = y_i(w \cdot x_i + w_0) \tag{2.20}$$

We also term the distribution of ρ_i on a sample S the *margin distribution* of c with respect to S. The minimum margin in the margin distribution is termed the *margin* $\rho(c, S)$ of c with respect the training set S. If $\mathcal{C} \subseteq C_{lin}$, then the margin $\rho(\mathcal{C}, S)$ of $\mathcal{C}$ with respect to S is defined as

$$\rho(\mathcal{C}, S) = \max_{c \in \mathcal{C}} \rho(c, S) = \max_{c \in \mathcal{C}} \min_{(x_i, y_i) \in S} y_i c_h(x_i) \tag{2.21}$$

Note that if $\rho(\mathcal{C}, S)$ is positive, then there is a classifier $c \in \mathcal{C}$ that classifies the sample S correctly.

We will now present generalization bounds using $\rho(\mathcal{C}, S)$. Without loss of generality, we assume that for any $c = c(w, w_0) \in C_{lin}$, $w_0 = 0$ and $|||w||_2 = 1$.

The first bound for the case, when there is a $c \in \mathcal{C}$ with zero empirical risk, is given by the following theorem:

Theorem 3 ([17]) *Let C_{lin} be a class of linear classifiers and $\rho > 0$. Then, for any probability distribution P on $X \times \{-1, 1\}$, such that $\exists R \in \mathbb{R}, P\{x : ||x||_2 \geq R\} = 0$, with probability $1 - \delta$ over a random sample S of size m, for any classifier c with margin $\rho(c, S) \geq \rho$:*

$$R(c) \leq \epsilon(m, \mathcal{C}, \delta, \rho) = \frac{2}{m} \left(\frac{64R^2}{\rho^2} \log \frac{em\rho}{8R^2} \log \frac{32m}{\rho^2} + \log \frac{4}{\delta} \right) \tag{2.22}$$

provided that $m > \frac{2}{\epsilon}$ and $\frac{64R^2}{\rho^2} < m$.

Note that the bound (2.22) is dimension-independent. Therefore, the number of features in a learning problem does not directly affect generalization. This is the drastic difference from the bounds (2.18) and (2.19), where the number of features is factored in via the feature-dependent VC dimension. Another important implication of Theorem 3 is that the bound (2.22) is *data-dependent*. That is, we can state the bound only after ascertaining that the margin of a classifier on a particular sample S is greater than the fixed ρ.

For the case, when the training data is noisy, and their correct classification is impossible using the classifiers in C_{lin}, we need to introduce an additional concept to give margin-based generalization bounds.

Definition 4 *For a classifier $c \in C_{lin}$, a labeled example $(x_i, y_i)(y_i \in \{-1, 1\})$, and the target*

margin ρ*, define the* margin slack variable $\xi((x_i, y_i), c, \rho)$ *as*

$$\xi((x_i, y_i), c, \rho) = \xi_i = max(0, \rho - \rho_i) \tag{2.23}$$

The margin slack variable quantifies how far a classifier c is from having the margin ρ on a labeled example. For a sample $S = ((x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m))$, a classifier c, and the target margin ρ, the vector $\xi = \xi(S, c, \rho) = (\xi_1, \xi_2, \ldots, \xi_m)$ is termed the *margin slack vector* of S with respect to c and ρ.

We can now state a general margin-based generalization bound.

Theorem 4 ([17]) *Let* C_{lin} *be a class of linear classifiers and* $\rho > 0$*. Then, there is a constant* c*, such that for any probability distribution* P *on* $X \times \{-1, 1\}$*, such that* $\exists R \in \mathbb{R}, P\{x : ||x||_2 \geq R\} = 0$*, with probability* $1 - \delta$ *over a random sample* S *of size* m*, for any classifier* c*:*

$$R(c) \leq \frac{c}{m}\left(\frac{R^2 + ||\xi||_2^2}{\rho^2}\log^2 m + \log\frac{1}{\delta}\right) \tag{2.24}$$

where $\xi = \xi(S, c, \rho)$

Theorem 4 suggests that given a training sample S we will minimize the generalization bound by seeking a classifier that minimizes

$$\frac{R^2 + ||\xi||_2^2}{\rho^2} \tag{2.25}$$

Note that minimizing (2.25) does not necessarily lead to minimizing the number of misclassifications. As we noted in Section 2.4, minimizing the number of misclassifications is computationally intractable. The bound (2.24) will allow us to obtain a tractable version of learning linear classifiers by replacing the 0-1 loss function with a loss function based on (2.25).

2.7 Support Vector Machine

The Support Vector Machine [33] is an algorithm that minimizes the fraction (2.25). We first note the reciprocal relationship between the norm of the weight vector $||w||_2$ and the margin ρ. Let us remove the unit norm restriction from w and instead fix the corresponding (geometric) margin

$\frac{\rho}{||w||_2} = 1$. Then, (2.25) is equivalent to ([35])

$$\frac{R^2 + \frac{||\xi||^2}{||w||_2^2}}{\rho^2} = ||w||^2 R^2 + ||\xi||_2^2 \tag{2.26}$$

where

$$\begin{aligned} y_i c_h(x_i) = y_i(w \cdot x + w_0) &\geq 1 - \xi_i,\ i = 1, 2, \ldots, m \\ \xi_i &\geq 0,\ i = 1, 2, \ldots, m \end{aligned} \tag{2.27}$$

In practice, minimization of (2.26) is replaced with a more general criterion:

$$||w||^2 + C||\xi||_2^2 \rightarrow \min \tag{2.28}$$

where $C > 0$ is some constant that is determined via cross-validation. The optimization problem with the objective function (2.28) and the constraints (2.27) is termed the Support Vector Machine (SVM) [33].

The SVM optimization problem is a quadratic program. There are numerous quadratic programming packages available; however, off-the-shelf algorithms, in general, have complexity of $O(m^3)$. While the complexity is a big improvement compared with lack of tractable algorithms for minimizing the number of classification mistakes, it is still too demanding for many practical learning problems involving tens and hundreds of thousands of examples. We note that, in recent years, there were significant advances and algorithm design for the SVM optimization problem, and a number of state-of-the-art algorithms exhibit sub-quadratic complexity [57]. Yet the desire to scale algorithms to larger datasets led to rebirth of a class of online learning algorithms that do not require significant computational resources.

2.8 Online Linear Learning Algorithms

Online learning algorithms are a class of learning algorithms that process one (labeled) example at a time. At each point in time, an online learning algorithm maintains a current hypothesis

(classifier). Given a new example, the algorithm updates the current hypothesis.

Many online learning algorithms are *mistake-driven*[74]. Given a new example, a mistake-driven algorithm uses the current hypothesis to predict the example label. If the predicted label is different from the actual label, the algorithm updates its hypothesis using an appropriate update rule.

Let us denote $m_A(S)$ the number of mistakes that an online learning algorithm A makes on a (possibly infinite) sequence of examples S. Let $\mathcal{S}$ be a set of possible example sequences. Denote

$$m_A(\mathcal{S}) = \max_{S \in \mathcal{S}} m_A(S)$$

We say that an algorithm A is *mistake-bounded* on $\mathcal{S}$, if there is a $M \in \mathbb{R}$, such that for any sequence $S \in \mathcal{S}$ the number of mistakes $m_A(S)$ is bounded by M:

$$m_A(\mathcal{S}) \leq M$$

The resulting *mistake-bound model of learning* [74] is closely related to the PAC model of learning, for any mistake-bound learning algorithm can be converted into a PAC learning algorithm. Moreover, if the mistake bound M is polynomial in the natural complexity parameters of the data, then the corresponding PAC learning algorithm will be polynomial as well [74].

We now present several online learning algorithms that are extremely scalable and thus well-suited for NLP learning problems. All of the algorithms learn linear classifiers in $\mathbb{R}^n$.

2.8.1 Perceptron

Perceptron [94] is a classical learning algorithm whose design was inspired by workings of the neuron. The algorithm pseudocode in shown as Algorithm 1 (without loss of generality, we assume that $w_0 = 0$).

Algorithm 1 The Perceptron Learning Algorithm

```
w := (0, ..., 0)
for all (x_i, y_i) ∈ S do
  if y_i(w · x_i) < 0 then
    w := w + y_i x_i
  end if
end for
```

The following theorem quantifies the number of Perceptron mistakes on a sequence of examples S.

Theorem 5 ([87]) *Let S be an example sequence, such that $\exists \rho > 0 \ \exists R > 0, \ \rho(C_{lin}, S) \leq \rho$ and $\forall (x_i, y_i) \in S, \ ||x||_2 \leq R$. Then,*

$$m_{Perceptron}(S) \leq \frac{R^2}{\rho^2} \tag{2.29}$$

Note that the bound (2.29) is present in the generalization bound (2.22) highlighting the relationship between a good mistake bound and quality of generalization.

For the case, when the examples are not linearly separable, we have the following theorem:

Theorem 6 ([44]) *Let S be an example sequence with no duplicate examples, such that $\exists R > 0, \ \forall (x_i, y_i) \in S, \ ||x||_2 \leq R$. Let c be a linear classifier in $\mathbb{R}^n$. Then,*

$$m_{Perceptron}(S) \leq \frac{(R + ||\xi||_2)^2}{\rho^2} \tag{2.30}$$

where $\xi = \xi(S, c, \rho)$.

Note again the similarity of the bound (2.30) to the bound (2.24) that led to the design of the Support Vector Machine.

2.8.2 Winnow

Winnow [73] is an online linear learning algorithm with a multiplicative update rule. When a mistake is made on the example, rather than adding (subtracting) the example to (from) the weight vector, as Perceptron does, Winnow multiplies (divides) the corresponding weights by a learning parameter β.

For simplicity, we assume that X is restricted to $\{0,1\}^n$ and constrain the class of linear classifiers over $\{0,1\}^n$ to the class of positive linear classifiers $C_{lin}^+ = \{c(w) \in C_{lin} : w_0 = -1, \ w_i > 0, \ i = 1, \ldots, n\}$. Winnow is shown as Algorithm 2 below (x_{ij} denotes the jth coordinate of the ith example, and θ is the threshold parameter):

The following theorem quantifies the number of mistakes Winnow makes while learning positive linear classifiers:

Algorithm 2 The Winnow Learning Algorithm

$w := (1, \ldots, 1)$
for all $(x_i, y_i) \in S$ **do**
  if $y_i(w \cdot x_i - \theta) < 0$ **then**
    $w_j := w_j \beta^{y_i x_{ij}},\ j = 1, \ldots, n$
  end if
end for

Theorem 7 ([73]) *Let S be an example sequence, such that $\exists c(w) \in C^+_{lin}\ \exists \rho > 0,\ \rho(c, S) \leq \rho$. Then, if $\beta = 1 + \frac{\rho}{2}$ and $\theta = n$,*

$$m_{Winnow}(S) = O\left(||w||_1 \max\left(\frac{\log n}{\rho^2}, \frac{1}{\rho}\right)\right) \tag{2.31}$$

where $||w||_1 = \sum_{j=1}^{n} w_j$.

Note that the bound (2.31) is different from the previous bounds, for it depends on complexity of the classifier c, where the complexity is measured as L_1 norm of its weight vector. It is the remarkable property of Winnow that while dependence on the complexity classifier is linear, dependence on the dimensionality of the feature space is only logarithmic. The property is termed *feature efficiency* and the algorithm is said to be *feature-efficient*. It can be shown that a similar bound for Perceptron would involve linear dependence on dimensionality of the feature space [64], thus making Winnow more appropriate for the case when the target classifier is assumed to have a sparse representation in the feature space.

It is often the case in NLP applications that there exist good classifiers with sparse representations; therefore, Winnow is well-suited for NLP learning problems.

2.8.3 Sparse Network of Winnows (SNOW)

The SNOW (Sparse Network Of Winnows) [63, 96] is a learning architecture based on the Winnow learning algorithm. SNOW is a two-layer network of linear classifiers. Nodes in the first layer of the network represent the input features; target nodes (i.e., the correct values of the classifier) are represented by nodes in the second layer. Links from the first to the second layer have weights; each target node is thus defined as a (linear) function of the lower level nodes. The network is sparse in that a target node need not be connected to all nodes in the input layer. For example, it

is not connected to input nodes (features) that were never active with it in the same example, or it may decide, during training, to disconnect itself from some f the irrelevant input nodes, if they were not active often enough.

Learning in SNOW proceeds in an online fashion. Every example is treated autonomously by each target subnetworks. It is viewed as a positive example by a few of these and a negative example by the others. At prediction time, given an example, the information propagates through all the competing subnetworks; and the one which produces the highest activity gets to determine the prediction.

While the Winnow learning algorithm is used at each target node to learn its dependence on other nodes, the SNOW architecture overcomes Winnow restrictions. Namely, SNOW is naturally suited for multiclass learning and able to learn a class of general linear classifiers in a feature-efficient manner. The SNOW architecture also incorporates a number of such attractive properties as the ability to discard infrequent features and produce confidence for classification decisions.

2.9 Kernel Methods

We presented several linear learning algorithms that exhibit good generalization and low computational complexity. Yet, for many learning problems, the optimal decision boundaries cannot be expressed via linear classifiers. For such learning problems, we have to either enlarge the hypothesis space to include non-linear classifiers, or transform the feature space so that a linear classifier will be sufficient for producing a good decision boundary in the new feature space. Naturally, the feature transformation has to be non-linear and the linear classifier in the transformed feature space will correspond to non-linear classifier in the original feature space. Formally, we seek a transformation $\phi = (\phi_1, \phi_2, \ldots, \phi_N)$:

$$\phi(x) = \phi(x_1, \ldots, x_n) = (\phi_1(x), \ldots, \phi_N(x))$$

Let F be the transformed space, and $c : F \rightarrow \{-1, 1\}$ be a linear classifier in the transformed space. Then,

$$c(x) = w \cdot \phi(x) + w_0 = \sum_{j=1}^{N} w_j \phi_j(x) + w_0 \tag{2.32}$$

For many linear learning algorithms, the produced weight vectors have an alternate represen-

tation as a linear combination of the training examples:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i + \alpha_0 \tag{2.33}$$

where $\alpha_i \geq 0,\ i = 1, \ldots, n$. Similarly, for a transformed feature space:

$$w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i) + \alpha_0 \tag{2.34}$$

For example, observe that the Perceptron learning algorithm in Section 2.8.1 updates the weight vector by adding positive examples or subtracting negative examples [44]. Therefore, Perceptron produces a weight vector that has the representation (2.33), with $\alpha_0 = 0$. If learning is done in the transformed feature space, substituting (2.34) into (2.32) gives

$$c(x) = \sum_{i=1}^{m} \alpha_i y_i \sum_{j=1}^{N} \phi_j(x_i)\phi_j(x) = \sum_{i=1}^{m} \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle \tag{2.35}$$

where $\langle \cdot, \cdot \rangle$ is a dot product of two vectors.

The representation (2.35) is termed the *dual representation* of linear classifiers. Note that the ability to compute the dot product $\langle \phi(\cdot), \phi(\cdot) \rangle$ is sufficient to run the Perceptron algorithm in the transformed space, if we maintain the dual representation (2.34) of the weight vector. Moreover, if we can compute the function $k(x, x') = \langle \phi(x), \phi(x') \rangle$ directly from x, x', without creating the transformed examples $\phi(x)$ and $\phi(x')$ explicitly, we will still be able to conduct Perceptron learning.

The function $k(x, x') = \langle \phi(x), \phi(x') \rangle$ is termed a *kernel*. A kernel computation corresponds to a dot product computation in the transformed feature space [35]. Using the kernel function, we can rewrite (2.35) as

$$c(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$$

A learning algorithm that processes learning examples only via computing dot products between them is termed a *dual learning algorithm*. Perceptron is not the only learning algorithm that allows for a dual formulation. In fact, development of kernel methods was fueled by the fact that the dual optimization problem for the Support Vector Machine also leads to dot product-based formulation. The dual optimization problem for (2.27) and (2.28) is equivalent to the following optimization

problem [33, 108]

$$\begin{aligned} &\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,l=1}^{m} y_i y_l \alpha_i \alpha_l (\langle x_i, x_l \rangle + \frac{1}{C} \delta_{il}) \rightarrow \max \\ &\sum_{i=1}^{m} y_i \alpha_i = 0 \\ &\alpha_i \geq 0, \ i = 1, \ldots, m \end{aligned} \tag{2.36}$$

where $\delta_{il} = 1$, if $i = l$, and $\delta_{il} = 0$, if $i \neq l$. It follows from (2.36) that SVM can learn in a transformed space $\phi(X)$ by substituting an appropriate kernel function $k(x_i, x_l)$ for $\langle \phi(x_i), \phi(x_l) \rangle$.

In many cases, it may be possible to compute the dot product of certain features without enumerating all the features. For example, let $x \in \mathbb{R}^2$, and consider the following feature transformation $\phi(x) = \phi(x_1, x_2) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2)$. Then, observe that

$$\langle \phi(x), \phi(x') \rangle = (\langle x, x' \rangle + 1)^2$$

Therefore, we can compute the kernel $k(x, x')$ efficiently without explicit enumeration of the transformed features, which include products of the original features, in addition to the original features themselves. The result can be generalized to products of length d in $\mathbb{R}^n$ by defining the kernel $k(x, x') = (\langle x, x' \rangle + 1)^d$. In fact, kernel functions can correspond to infinite-dimensional feature spaces, as it is the case with the Gaussian kernel $k(x, x') = e^{-\frac{||x-x'||}{\sigma^2}}$ [108].

It is also possible to take a kernel function k, rather than features, as a starting point of a learning algorithm application. Indeed, any symmetric[4] and positive-semidefinite[5] function k (over examples) is necessarily a kernel [35]. Intuitively, a kernel can be treated as a similarity function, and many learning applications allow to design natural similarity function over learning examples. We pursue this approach in Chapter 5, where we design domain-specific similarity functions, and then prove that the functions are kernels.

[4] A binary function $k(x, x')$ is termed *symmetric* (over X), if $\forall x, x' \in X, \ k(x, x') = k(x', x)$

[5] A binary symmetric function $k(x, x')$ is termed *positive-semidefinite* (over X), if $\forall x_1, \ldots, x_n \in X$, the $n \times n$ matrix $(k(x_i, x_j))_{i,j=1}^n$ is positive-semidefinite (has only non-negative eigenvalues).

2.10 Probabilistic Modeling

We noted in Section 2.2 that the problem of estimating a probability distribution in a high-dimensional space from a finite sample is a very difficult task. However, if additional knowledge about the structure of the high-dimensional space is available, the knowledge can be leveraged to make the estimation process more tractable.

A most common type of knowledge available is a set of (conditional) independence assumptions that can be imposed on the domain features. For example, consider the classification problem introduced in Section 2.2. Let $p(x|y_i)$ be a probability distribution corresponding to the ith class. Let us impose the assumptions that the features $x_1, x_2, \ldots, x_n$ are independent given the class membership information y_i. Then, the class probability distribution can be decomposed as

$$p(x|y_i) = p(x_1|y_i)p(x_2|y_i)\cdots p(x_n|y_i)$$

Now we can estimate each probability distribution $p(x_j|y_i)$, $j = 1, 2, \ldots, n$ via maximum likelihood. We can then use the estimated probability distributions for classification as delineated in Section 2.2. Such an approach to classification is termed *Naive Bayes* [38], and it is the most prevalent algorithm for numerous problem in natural language processing, especially, text categorization.

We can portray graphically the conditional independence assumptions introduced above, as shown in Figure 2.1.

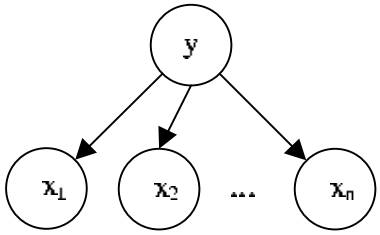

Figure 2.1: Naive Bayes Graphical Model

Figure 2.1 is an example of a *directed graphical model* [89]. A directed graphical model is a directed acyclic graph $G = (V, E)$, where V is the set of graph nodes and E is the set of graph edges. The graph nodes correspond to random variables (features). For each node v of the graph we define the parent set of nodes $parents(v)$ as $\{u \in V : (u, v) \in E\}$. Descendants of a node v are nodes

that are reachable from v in G. The graph edges enforce conditional independence assumptions. Namely, a node is independent of its non-descendants given its parents. The conditional probability distributions $p(v|parents(v))$ uniquely determine the joint probability distribution, for

$$p(V) = p(v_1, \ldots, v_n) = \prod_{i=1}^{n} p(v_i|parents(v_i))$$

Directed graphical models are often called *Bayesian networks* [89]. In recent years, Bayesian networks became an extremely popular tool for encoding probabilistic domain knowledge in complex domains.

Since a Bayesian network is uniquely determined by its conditional probability distributions, we can use the maximum likelihood approach to estimate the $p(v|parent(v))$ given a finite sample. In practice, the conditional probability distributions may be difficult to estimate due to lack of data, and various smoothing techniques are needed to produce robust estimates. Moreover, some of the network random variables may be hidden, i.e., not present in the sample. In that case, we will need to use the EM algorithm [36] to determine network parameters.

Bayesian networks are commonly used for *inference*, which is the process determining the probabilities or most probable values of hidden variable of a Bayesian network, given values for its visible variables. For example, the process of determining a class label y given an unlabeled example x is an inference. In general, the process of inference in Bayesian networks is computationally intractable [95]. In practice, the computational complexity of approximate algorithms for inference is linear in the network size [83], although the approximation error guarantees are not known.

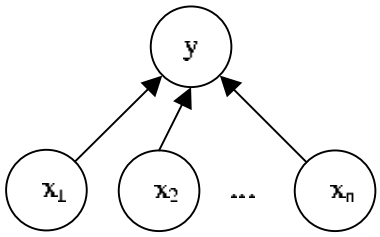

Figure 2.2: Conditional Model

Consider the graphical model in Figure 2.2. In contrast to the Naive Bayes model, the variables $x_1, x_2, \ldots, x_n$ are not required to be independent and the distribution $p(y|x_1, \ldots, x_n)$ cannot be

factored. Let us assume that the variable $y \in \{-1, 1\}$ and that the distribution $p(y|x_1, \ldots, x_n)$ has the following form:

$$p(y|x_1, \ldots, x_n) = \frac{1}{1 + e^{-yw^T x}} \tag{2.37}$$

where w is the vector of parameters. The resulted model is termed the *logistic regression* model. The special parametric form of the logistic regression model implies that its maximum likelihood estimation leads to a concave optimization problem with a unique maximum. Hence, numerous efficient optimization algorithm can be used to find the maximum likelihood estimate [80].

In natural language processing and, in particular, information extraction, Bayesian networks have been used since the early 90s to model sequential data.

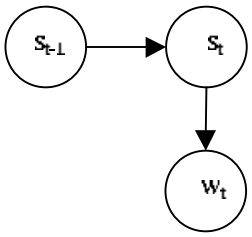

Figure 2.3: Hidden Markov Model

Consider the graphical model in Figure 2.3 that describes a random process. The random variable s_t is the state of random process at a time t, and s_{t-1} is the state of the random process at time $t-1$. At each time t, the random process emits an observation w_t with probability $p(w_t|s_t)$. The process also moves from the state s_{t-1} to the state s_t according to the conditional probability $p(s_t|s_{t-1})$. We assume that the distributions $p(w_t|s_t)$ and $p(s_t|s_{t-1})$ are stationary, i.e, independent of t. The random process is called a *Hidden Markov Model*.

In NLP applications, observations w_t usually correspond to visible elements of language (phonemes, words, etc.), while the states s_t correspond to the hidden high-level linguistic constructs (parts of speech, word classes, etc.) [27].

The Hidden Markov Model admits efficient estimation and inference algorithms. In a fully observable case (all s_t and w_t are visible), we can use maximum likelihood for estimation. If the states are hidden, the Baum-Welch algorithm [18] provides a fairly efficient (though not necessarily optimal) mechanism for estimating HMM parameters. The Viterbi decoding algorithm [42] is used

for determining the most probable state sequence given an observation sequence. Complexity of the Viterbi algorithm is $O(mn^2)$, where m is the sequence length, and n is the number of possible values for the random variable s_t.

HMMs belong to a class of generative models for natural language. Generative models impose a dependency structure that specifies that surface elements of natural language have been generated by hidden elements via the postulated dependency structure. Furthermore, generative models describe the joint distribution of both surface elements and hidden structural elements. During the estimation process, the parameters of the dependency structure are estimated. During the prediction process, a decoding (inference) algorithm is invoked to determine the most probable hidden elements that provide a high-level interpretation of the surface text. Other examples of generative models for NLP include *Probabilistic Context Free Grammars* (PCFGs) [27] and *Markov Random Fields* [30, 72].

Probabilistic Context Free Grammars are a probabilistic analog of context free grammars. More precisely, grammar productions $A \rightarrow B$ are augmented with probabilities specifying how likely a particular production to be invoked for a non-terminal symbol (A). PCFGs are more appropriate that HMMs for modeling non-local dependencies in text. They are the formalism of choice for natural language parsing. However, the estimation and inference algorithms for PCFGs are more complex than those of HMMs.

Markov Random Fields (MRF) [72] are undirected graphical models. An undirected graphical model is described by an undirected graph $G = (V, E)$. As it is the case with directed graphical models, nodes in the graph correspond to random variables. However, MRFs do not have the intuitive generative interpretation of directed graphical models. Instead of using conditional probability distribution, MRFs decompose the joint probability distribution over its random variables as the product of potential functions defined over the cliques of the graph G. Let $Cl(G)$ be the set of all cliques of the graph G, and let $\psi_{cl}(v)$ be a potential function of a clique $cl \in Cl$ that depends on the subset variables corresponding to the clique nodes. Then,

$$p(V) = p(v_1, v_2, \ldots, v_n) = \frac{1}{Z} \prod_{cl \in Cl} \psi_{cl}(v) \tag{2.38}$$

where $Z = \sum_{(v_1,\ldots,v_n)} \prod_{cl \in Cl} \psi_{cl}(v)$ is a normalizing constant.

The Hammersley-Clifford theorem [51] states that the expression (2.38) for a probability distribution is equivalent to the condition that dependencies in the MRF are local, that is, a variable is independent of its non-neighbors given its neighbors in G.

MRFs are an attractive formalism because its likelihood function (2.3) is necessarily convex, which guarantees absence of local maxima in the estimation process. Unfortunately, complexity of the estimation process is rather high. The inference mechanism for MRFs is essentially equivalent to that in Bayesian networks, and approximate algorithms exist for fast inference.

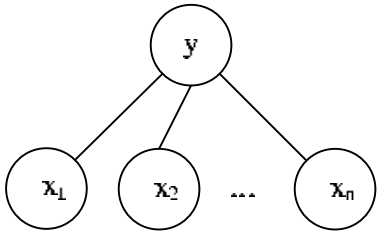

Figure 2.4: Maximum Entropy Graphical Model

Recently, *Maximum Entropy Models* [19], a special case of MRFs, were frequently used for NLP problems. The graphical structure of a typical Maximum Entropy Model is shown in Figure 2.4, where v_i's are (observed) random variables, and c is a (hidden) classification random variable. The Maximum Entropy model is a undirected Naive Bayes model. It is also the MRF, where the cliques are just edges connecting the observed nodes to the classification node. Therefore, Maximum Entropy models inherit absence of local maxima of the likelihood function and admit somewhat more efficient estimation algorithms.

Chapter 3

Part of Speech Tagging

Part of speech tagging is the problem of identifying parts of speech of words in a presented text. Since words are ambiguous in terms of their part of speech, the correct part of speech is usually identified from the context the word appears in. For example, in the sentence, "The can will rust", "the" is a definite determiner, "can" is a singular noun, "will" is a modal verb, and "rust" is a verb. On the other hand, in the sentence, "We can contest the will", "can" is a modal verb, and "will" is a singular noun. The examples showcase ambiguity inherent in making part of speech assignments. They also lead us to believe that determining parts of speech is an important prerequisite for high-level tasks in natural language processing. In particular, part of speech information plays a crucial role in information extraction systems.

3.1 Related Work

In recent years, a number of approaches have been tried for solving the problem. The most notable methods are based on Hidden Markov Models(HMM) [68, 102], Maximum Entropy models [92], and transformation based learning [24].

3.1.1 Probabilistic Modeling

In the part of speech HMM model, the states correspond to the part of speech tags and the observations are words. The dependency structure of the model is shown in Figure 2.3. In the model, a part of speech only depends on the previous part of speech. We can expand the context of part of speech dependencies by making the part of speech at the time t also depend on the part

of speech at the time $t-2$, whereby we obtain a trigram POS tagger shown in Figure 3.1.

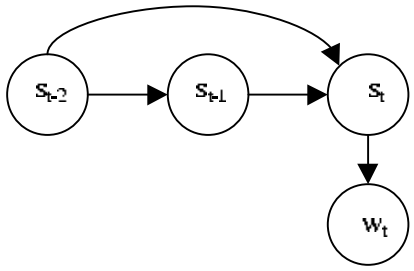

Figure 3.1: Trigram Graphical Model

Other dependency structures are possible; for instance, we can make the part speech at the time t depend on the part of speech at the time $t-1$ and and the word at the time t (Figure 3.2). The model is an example of *conditional models*. Conditional models do not model the joint probability distribution of hidden and visible variables, but instead model conditional probability distributions of hidden variables given visible variables. The conditional probability distribution $p(pos(t)|pos(t-1), w(t))$ can be estimated using such techniques as Maximum Entropy [92].

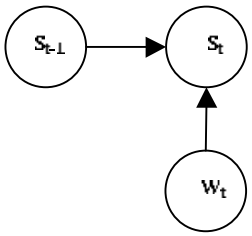

Figure 3.2: Conditional Graphical Model

3.1.2 Transformation-based Learning

Transformation based learning(TBL) [24] is a machine learning approach for rule learning. The learning procedure is a mistake-driven algorithm that produces a set of rules. The hypothesis of TBL is an ordered list of transformations. A *transformation* is a rule with an antecedent t and a consequent $c \in C$. The antecedent t is a condition on the input sentence. For example, a condition might be `the preceding word tag is` t. That is, applying the condition to a sentence s defines a feature $t(s)$. Phrased differently, the application of the condition to a given sentence s, checks

whether the corresponding feature is active in this sentence. The condition holds if and only if the feature is active in the sentence.

The TBL hypothesis is evaluated as follows: given a sentence s, an initial labeling is assigned to it. Then, each rule is applied, in order, to the sentence. If the condition of the rule applies, the current label is replaced by the label in the consequent. This process goes on until the last rule in the list is evaluated. The last labeling is the output of the hypothesis.

In its most general setting, the TBL hypothesis is not a classifier [24]. The reason is that, in general, the truth value of the condition of the ith rule may change while evaluating one of the preceding rules. For example, in part of speech tagging, labeling a word with a part of speech changes the conditions of the following word that depend on that part of speech(e.g., the preceding word tag is t).

TBL uses a manually-tagged corpus for learning the ordered list of transformations. The learning proceeds in stages, where on each stage a transformation is chosen to minimize the number of mislabeled words in the presented corpus. The transformation is then applied, and the process is repeated until no more mislabeling minimization can be achieved.

For example, in POS, the consequence of a transformation labels a word with a part of speech. [24] uses lexicon for initial annotation of the training corpus, where each word in the lexicon has a set POS tags seen for the word in the training corpus. Then a search in the space of transformations is conducted to determine a transformation that most reduces the number of wrong tags for the words in the corpus. The application of the transformation to the initially labeled corpus produces another labeling of the corpus with a smaller number of mistakes. Iterating this procedure leads to learning an ordered list of transformation which can be used as a POS tagger.

3.2 Problem Formalization

Let s be a input sentence. A sentence s is a sequence of words $w_1, w_2, \ldots, w_T$. We aim to assign to each word w_i its part of speech pos_i thereby producing a tagged sentence $(w_1, pos_1), \ldots, (w_n, pos_n)$. Let POS be the set of all parts of speech, and let $context(w_i)$ be the feature vector representation of the context of word w_i. The context features correspond to properties of words surrounding the word w_i. Let $Context$ be the set of all possible contexts. We will assign parts of speech to words

by using a classifier $c_{pos} : Context \rightarrow POS$.

Let $S = \{s_1, s_2, \ldots, s_m\}$ be the training corpus, where every word is labeled with its correct part of speech. We will use the training corpus to estimate the classifier c_{pos} by converting the corpus in a set of labeled examples $\{(context(w), pos(w)) : w \in S\}$, and learning a classifier from the examples.

At the evaluation stage, given a sentence s without POS information, we will generate unlabeled examples $\{(context(w)) : w \in s\}$, and the classifier c_{pos} will assign POS labels to examples.

Note that, at the training stage, the context features can include the *true* parts of speech of words surrounding the word w, for which an example is generated. Since this information is not available at the evaluation stage, we will use already predicted part of speech tags as well as the word *baseline* tags computed from the lexicon, to compute the corresponding context features. Hence, part of speech prediction will be done in the presence of *attribute noise*, and we seek a classifier that is robust with respect to the noise.

3.3 SNOW for Part of Speech Tagging

We address the POS problem from the standpoint of SNOW. That is, we represent a POS tagger as a network of linear classifiers and use Winnow for learning weights of the network. The SNOW approach has been successfully applied to other problems of natural language processing [49, 66, 96]. However, this problem offers additional challenges to the SNOW architecture and algorithms. First, we are trying to learn a multi-class predictor, where the number of classes is unusually large(about 50) for such learning problems. Second, evaluating hypothesis in testing is done in a presence of attribute noise.

We address the first problem by restricting the parts of speech a tag for a word is selected from. Second problem is alleviated by performing several labeling cycles on the testing corpus.

3.4 The Tagger Network

The tagger network consists of a collection of linear classifiers, each corresponds to a distinct part of speech. The 50 parts are taken from the WSJ corpus. The input nodes of the network correspond

to the features. The features are computed for a fixed word in a sentence. We use the following set of features (the features 1-8 are part of [24] features):

1. The preceding word is tagged pos_1.
2. The following word is tagged pos_1.
3. The word two before is tagged pos_1.
4. The word two after is tagged pos_1.
5. The preceding word is tagged pos_1 and the following word is tagged pos_2.
6. The preceding word is tagged pos_1 and the word two before is tagged pos_2.
7. The following word is tagged pos_1 and the word two after is tagged pos_2.
8. The current word is w.
9. The most probable part of speech for the current word is pos_1.

The most probable part of speech for a word is taken from a lexicon. The lexicon is a list of words with a set of possible POS tags associated with each word. The lexicon can be computed from available labeled corpus data, or it can represent the a-priori information about words in the language.

Training of the SNOW tagger network proceeds as follows. Each word in a sentence produces an example. Given a sentence, features are computed with respect to each word thereby producing a positive examples for the part of speech the word is labeled with, and the negative examples for the other parts of speech. The positive and negative examples are presented to the corresponding subnetworks, which update their weights according to Winnow.

In testing, this process is repeated, producing a test example for each word in the sentence. In this case, however, the POS tags of the neighboring words are not known and, therefore, the majority of the features cannot be evaluated. We discuss later various ways to handle this situation. The default one is to use the baseline tags - the most common POS for this word in the training lexicon.

Once an example is produced, it is then presented to the networks. Each of the subnetworks is evaluated and we select the one with the highest level of activation among the classifiers corresponding to the possible tags for the current word. After every prediction, the tag output by the SNOW tagger for a word is used for labeling the word in the test data. Therefore, the features of the following words will depend on the output tags of the preceding words.

3.5 Experimental Results

The data for all the experiments was extracted from the Penn Treebank WSJ corpus. The training and test corpus consist of 600000 and 150000, respectively. The first set of experiment uses only the SNOW system and evaluate its performance under various conditions. In the second set SNOW is compared with a naive Bayes algorithm and with Brill's TBL, all trained and tested on the same data. We also compare with *Baseline* which simply assigns each word in the test corpus its most common POS in the lexicon. Baseline performance on our test corpus is 94.1%.

A lexicon is computed from both the training and the test corpus. The lexicon has 81227 distinct words, with an average of 2.2 possible POS tags per word in the lexicon.

3.5.1 Investigating SNOW

We first explore the ability of the network to adapt to new data. While online algorithms are at a disadvantage - each example is processed only once before being discarded - they have the advantage of (in principle) being able to quickly adapt to new data. This is done within SNOW by allowing it to update its weights in test mode. That is, after prediction, the network receives a label for a word, and then uses the label for updating its weights.

In test mode, however, the *true* tag is not available to the system. Instead, we used as the feedback label the corresponding baseline tag taken from the lexicon. In this way, the algorithm never uses more information than is available to batch algorithms tested on the same data. The intuition is that, since the baseline itself for this task is fairly high, this information will allow the tagger to better tolerate new trends in the data and steer the predictors in the right direction. This is the default system that we call SNOW in the discussion that follows.

Another policy with on-line algorithms is to supply it with the *true* feedback, when it makes

a mistake in testing. This policy (termed adp-SNOW) is especially useful when the test data comes from a different source than the training data, and will allow the algorithm to adapt to the new context. For example, a language acquisition system with a tagger trained on a general corpus can quickly adapt to a specific domain, if allowed to use this policy, at least occasionally. What we found surprising is that in this case supplying the true feedback did not improve the performance of SNOW significantly. Both on-line methods though, perform significantly better than if we disallow on-line update, as we did for noadp-SNOW. The results, presented in Table 3.1, exhibit the advantage of using an on-line algorithm.

noadp-SNOW	SNOW	adp-SNOW
96.5	97.13	97.2

Table 3.1: **Effect of adaptation:** Performance of the tagger network with no adaptation(noadp-SNOW), baseline adaptation(SNOW), and true adaptation(adp-SNOW).

One difficulty in applying the SNOW approach to the POS problem is the problem of attribute noise alluded to before. Namely, the classifiers receive a noisy set of features as input due to the attribute dependence on (unknown) tags of neighboring words. We address this by studying quality of the classifier, when it is guaranteed to get (almost) correct input.

Table 3.2 summarizes the effects of this noise on the performance. Under SNOW we give the results under normal conditions, when the the features of the each example are determined based on the baseline tags. Under SNOW+cr we determine the features based on the *correct* tags, as read from the tagged corpus. One can see that this results in a significant improvement, indicating that the classifier learned by SNOW is almost perfect. In normal conditions, though, it is affected by the attribute noise. In attempt to reduce the effect of the noise under SNOW+cyc we use the results of the first testing cycle as the initial labeling for the next testing cycle. Indeed, from this standpoint, POS tagging can be viewed as a reduction in attribute noise of the testing data. Clearly, the input to the second cycle is less noisy (by more than 2%), resulting in a slight increase in performance. The first cycle decreased attribute noise by more than 2%. The following cycle, however, led to a minor improvement in performance(0.1%). Additional cycles did not produce any noticeable improvement. The tagger appears to have reached a local maximum(with respect to tagging accuracy).

Baseline	SNOW+cr	SNOW
94.1	98.8	97.13

Table 3.2: **Quality of classifier:** The SNOW tagger was tested with correct initial tags (SNOW+cr) and, as usual, with baseline based initial tags.

Next, we experimented with the sensitivity of SNOW to several options of labeling the training data.

- Usually both features and labels of the training examples are computed in terms of correct parts of speech for words in the training corpus. (this approach was used in the experiments described above).
- We call the labeling *Semi-supervised* when we only require the features of the training examples to be computed in terms of the most probable pos for words in the training corpus, but the labels still correspond to the correct parts of speech.
- The labeling is *Unsupervised* when both features and labels of the training examples are computed in terms of most probable POS of words in the training corpus.

Baseline	SNOW+uns	SNOW+ss	SNOW
94.1	94.3	97.13	97.13

Table 3.3: **Effect of supervision**. Performance of SNOW with *unsupervised* (SNOW+uns), *semi-supervised* (SNOW+ss) and normal mode of *supervised* training.

It is not surprising that the performance of the tagger learned in an *semi-supervised* fashion is the same as that of the one trained from the correct corpus. Intuitively, since in the test stage the input to the classifier uses the baseline classifier, in this case there is a better fit between the data supplied in training (with a correct feedback!) and the one used in testing.

The results of learning from all three kinds of training data and then applying the learned taggers to the test data, which are initially labeled with most probable part of speech, are shown in Table 3.3. We see that the tagger learned in the unsupervised fashion is slightly better than the baseline, and the results of semi-supervised and supervised taggers are basically the same.

3.5.2 Comparative Study

We compared performance of the SNOW tagger with one of the best POS taggers, based on Brill's TBL, and with a naive Bayes (e.g., [38] based tagger. We used the same training and test sets. The results are summarized in Table 3.4.

Baseline	NB	TBL	SNOW	adp-SNOW
94.1	96	97.15	97.13	97.2

Table 3.4: **Comparison of tagging performance.** Tagging accuracy is compared for Naive Bayes(NB), TBL, SNOW and adp-SNOW taggers.

In can be seen that the TBL tagger and SNOW perform essentially the same. However, given that SNOW is an online algorithm, we have tested it also in its (true feedback) adaptive mode, where it is shown to outperform them. It is interesting to note that a simple minded NB method also performs quite well.

Another important point of comparison is that the NB tagger and the SNOW taggers are trained with the features described in Section 3.4. TBL, on the other hand, uses a much larger set of features. Moreover, the learning and tagging mechanism in TBL relies on the inter-dependence between the produced labels and the features. However, [91] demonstrate that the inter-dependence impacts only 12% of the predictions. Since the classifier used in TBL without inter-dependence can be represented as a linear classifier[96], it is perhaps not surprising that SNOW performs as well as TBL. Also, the success of the adaptive SNOW taggers shows that we can alleviate the lack of the inter-dependence by adaptation to the testing corpus. It also highlights importance of relationship between a tagger and a corpus.

3.5.3 Alternative Performance Metrics

We propose additional measures of POS tagging performance. First, many words can have only one part of speech in the language. If this information is available to a tagging program, then tagging performance for these words should be 100%. That is, the words are deterministic with respect to their parts of speech. We believe that it often misleading to reflect tagging accuracy for deterministic words in performance measures. Therefore, In Table 3.5 we present the results only for ambiguous words which, we feel, is a better performance measure. For example, out of

150000 words from the testing corpus that we used, about 65000 were deterministic. Naturally, excluding the deterministic words from the performance measure led to a decrease in the values of the measure. The results of tagging accuracy for non-deterministic words are shown in Table 3.5.

Baseline	NB	TBL	SNOW	adp-SNOW
90.1	93	95	95	95.2

Table 3.5: **Performance for ambiguous words.**

Sometimes we may be interested in determining POS *classes* of words rather than simply parts of speech. For example, some natural language applications may require identifying that a word is a noun without specifying the exact noun tag for the word(singular, plural, proper, etc.). In this case, we want to measure performance with respect to POS classes. That is, if the predicted part of speech for a word is in the same class with the correct tag for the word, then the prediction is termed correct.

Out of 50 POS tags we created 12 POS classes: punctuation marks, determiners, preposition and conjunctions, existential "there", foreign words, cardinal numbers and list markers, adjectives, modals, verbs, adverbs, particles, pronouns, nouns, possessive endings, interjections. The performance results for the classes are shown in Table 3.5.3.

Baseline	NB	TBL	SNOW	adp-SNOW
96.2	97	97.95	97.95	98

Table 3.6: **Performance for POS classes.** The 50 parts of speech are separated into 12 non-overlapping classes. Tagging accuracy is then computed with respect to predicting the right class rather than the correct part of speech.

3.5.4 Discussion

We applied SNOW to the problem of part of speech tagging, with excellent results. Our investigation indicates that SNOW architecture is well-suited for natural language processing applications due to its ability to effectively exploit data sparsity as well as efficient online processing of examples. We also observed that performance of SNOW for part of speech tagging is comparable to that of Transformation-based Learning, the state-of-the-art method, which is much more computationally intensive.

The computational complexity of SNOW during both training and evaluation is linear in the number of classes. The linear dependence may make the architecture difficult to scale to larger problems with hundreds and thousands of classes (e.g., word prediction). We note the recent work of [40], which introduces *sequential model* in conjunction with SNOW, that addresses the problem of linear dependence and allows to effectively reduce the complexity of both training and evaluation for multi-class problems.

Chapter 4

Entity Extraction and Coreference Resolution

Entity extraction is the process of recognizing entities in text. A well-defined set of possible entities is determined by a particular information extraction application. For example, entities may include *people*, *organizations*, and *locations*. Entities can also be expressed as names (`John Smith`), noun phrases (`chief scientist of WorldCom Corp.`), or pronouns (`he`).

4.1 Entity Extraction Overview and Related Work

We will distinguish the entity extraction sub-tasks corresponding to name, nominal (noun phrase), and pronoun entity extraction. Such differentiation is useful from the algorithmic perspective since, historically, different algorithms have been developed for extraction of names, nominals, and pronouns. We will now present each of the entity extraction subtasks in more detail.

4.1.1 Named Entity Extraction

Named Entity Extraction is the earliest widely studied information extraction task. Several Message Understanding Conferences included named entity extraction as an evaluation task and resulted in significant advances in named entity identification [4, 5].

According to the Conferences, a named entity extraction system seeks to identify all proper names and quantities that represent people, organizations, locations, dates, times, money amounts and percentages. For example, the sentence `Arthur Rudolph, the developer of the giant Saturn 5 rocket that launched a crew of American astronauts on the first manned`

`flight to the moon in 1969, died Friday after he lapsed into a coma at his home in Hamburg, Germany` has the *person* entity `Arthur Rudolph`, the three *location* entities, `the moon`, `Hamburg`, `Germany`, and the two *date* entities `1969`, `Friday`. The set of named entities is domain-dependent, and different information extraction applications may define a totally different set of named entities.

Early systems for named entity extraction were manually built. They comprised a collection of cascaded finite-state machines that extracted entities based on hand-crafted patterns ([14, 54, 78]). Such manually built systems, though sometimes achieving excellent performance, are difficult and expensive to maintain. They also suffer from lack of portability since a new type of text or a new domain usually requires significant modification of the hand-crafted patterns.

The adaptive approaches to named entity extraction were pioneered by an influential HMM-based extraction system [20]. The system represents a generative model for text, where the states of HMM correspond to particular entity types, or absence thereof. The system graphical model is shown in Figure 4.1.

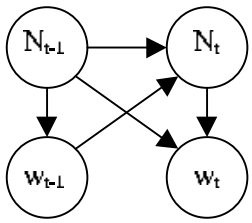

Figure 4.1: Named Entity Extraction Graphical Model

The conditional probability distributions $p(N_t|N_{t-1}, w_{t-1})$ and $p(w_t|N_t, N_{t-1})$ are very difficult to estimate due to data sparsity. In order to counter data sparsity, multi-level back-off techniques are used for estimation [59, 65]. Back-off techniques interpolate the conditional probabilities, for which not enough training data is available, with less specific conditional probabilities. For example,

$$\tilde{p}(N_t|N_{t-1}, w_{t-1}) = \lambda p(N_t|N_{t-1}, w_{t-1}) + (1 - \lambda)p(N_t|N_{t-1})$$

where λ depends on the number of occurrences of (N_{t-1}, w_{t-1}) in the training data. The back-off process can be applied recursively to $p(N_t|N_{t-1})$ to obtain even more robust estimates.

Other graphical models are possible for named entity extraction. Recently, conditional models have been applied to named entity extraction, with superior results [93, 23]. Conditional models can avoid heuristics of back-off estimation by plugging in any learning learning algorithm that produces a classifier that outputs conditional probabilities. This will lead to a more robust estimation process as well as the ability to augment conditional probability distributions with additional features (e.g., case-sensitivity), which are not easy to incorporate in the back-off estimation process.

4.1.2 Nominal Entity Extraction

Nominal entities are noun phrases denoting the entities of interest. For example, in the sentence, `Arthur Rudolph, the developer of the giant Saturn 5 rocket that launched a crew of American astronauts on the first manned flight to the moon in 1969, died Friday after he lapsed into a coma at his home in Hamburg, Germany`, the noun phrase `the developer of the giant Saturn 5 rocket that launched a crew of American astronauts on the first manned flight to the moon in 1969` denotes a nominal *person* entity.

The task of extracting nominal entities *per se* was not part of the MUC effort. The noun phrases were to be extracted, according to MUC, only if they were attached to (denoted) a named entity. The recent Automatic Content Extraction program [10] extended the extraction task to include nominal entities as well. For example, in the sentence `The jury deliberated for three hours`, the noun phrase `the jury` is to be extracted and treated as a nominal *person* entity, according to the ACE guidelines.[1]

In contrast to the named entity extraction problem, where recognition and classification of entities are considered and modeled simultaneously, it is beneficial for nominal entity extraction to explicitly separate the tasks of noun phrase recognition and their subsequent classification into one of the pre-defined entity types. This separation is a result of significant body of research on the general problem of noun phrase identification (chunking). While the task of extracting nominal entities is only beginning to be studied extensively, the problem of noun phrase chunking has been the object of significant research in recent years [7]. We will briefly review prevalent approaches to noun phrase chunking.

[1] See [10] for details.

Noun phrase chunking is usually considered as part of the general chunking (or shallow parsing) paradigm that seeks to assign a partial syntactic structure to a sentence. In contrast to full parsing, shallow parsing does not provide a complete interpretation of a sentence, but instead identifies only a limited number of phrases. It has been argued [8] that for many natural language processing applications, a solution to the much more complex problem of full parsing is not necessary and can be superseded with that of shallow parsing, which is, in general, an easier and more tractable task. The types of phrases (chunks) identified by shallow parsers include noun phrases, verb phrases, prepositional phrases, and others. Sometimes shallow parsers identify subject/verb/object structure and predicate/argument structure as well.

In the original shallow parsing paper [8], Abney presented a manual approach to chunking by constructing a set of finite-state machines based on words and their part of speech information. Yet the difficulties associated with building shallow parsers manually as well as availability of training data (e.g., the fully parsed Wall Street Journal corpus), led to significant developments in application of machine learning to the shallow parsing problem. In particular, the recent chunking competition under the aegis of the Computational Natural Language Learning (CoNLL) conference [6] showcased a variety of learning algorithms and their performance on the the problem.

From the learning perspective, the chunking problem is formalized as a tagging problem: each word is assigned a tag denoting the position of the word in a phrase, if any. The tag may reflect that the word begins a phrase (B), the word is located inside a phrase (I), or that the word is outside of a phrase (O). Such a tagging approach is sometimes termed BIO modeling. A learning algorithm applied to the tag representation of the shallow parsing problem seeks to learn a classifier that predicts, for each word, its tag given the word context that may include surrounding words, part of speech tags, as well as the chunking tags.

Many learning approaches exhibit excellent performance on the shallow parsing problem. We note the applications of SNOW [82], regularized Winnow [104], and Support Vector Machines [67] that deliver state of the art performance for the problem.

Since the technology of noun phrase identification is readily available, it is natural to separate extraction of nominal entities into two stages. The first stage utilizes the general shallow parsing component to output a set of noun phrases, while the second phase performs an additional classifi-

cation step that determines whether an extracted noun phrase represents an entity of interest. We do adopt the two stage approach in our work, but remark that straightforward one-stage extraction of only the target nominal entities, as defined by the extraction task, may be a viable alternative and deserves to be studied in the future.

4.1.3 Pronominal Entity Extraction

Pronominal entities are pronouns that denote entities of interest. For pronouns, the recognition problem is trivial, for the class of pronouns is closed, and they can be unambiguously identified in text.[2] However, the pronoun classification problem by the entity type is difficult. For example, some pronouns are *pleonastic*, that is, they do not refer to anything: `It is cold.`

For many pronouns, it is difficult to determine the type of the entity a pronoun denotes based on its local context alone: `The negotiations between the two countries ended successfully. They both confirmed plans for further cooperation`. The pronoun `they` in the previous sentence refers to `the two countries`, and this information can only be ascertained by taking into account the larger context that comprises more than a single sentence.

The difficulty of classifying pronouns on their own suggests that their classification need not be considered as a independent problem, but instead treated in conjunction with the task of coreference resolution. Indeed, we pursue the combinational approach in Section 4.5.1 and show that it improves extraction performance.

4.2 Coreference Resolution Overview

Coreference resolution is the problem of determining whether different extracted entities correspond to the same real-world entities.

In the literature, the problem of anaphora resolution is often studied [81], which is closely related to the coreference resolution problem. *Anaphora* is a phenomenon of referring to an already mentioned entity in a document. The reference is then called an *anaphor* and the referred entity is termed an *antecedent*. Anaphora resolution problem is usually restricted to nominal and pronominal

[2] There are exceptions, of course. For instance, we sometimes need to disambiguate the pronoun "I" from the roman numeral "I".

anaphors, thereby ignoring the problem of named entity coreference, which is extremely important for information extraction. Additionally, since anaphora addresses (literally) only backward references, the infrequent phenomenon of forward references (termed *cataphora*) is not covered by anaphora resolution. In our presentation, the term "coreference resolution" implies resolution of named, nominal, and pronominal entities that subsumes both backward and forward references.

The information extraction perspective on coreference resolution imposes a limited scope on the set of entities to be resolved. Indeed, we are not interested in resolving all coreferences in a document, but only those involving entities to be extracted as part of a specific extraction task. Thus, we can safely ignore coreference resolution of noun phrases if they are deemed irrelevant to the extraction task at hand.

Let us define the coreference relation $coref$ on a set of document entities. We say that the relation $coref(x, y)$ holds if and only if the entities x and y are coreferent. It is clear that $coref$ is an equivalence relation (it is obviously reflexive, symmetric, and transitive); hence, it induces a set of equivalence classes of document entities. The goal of coreference resolution is to produce a set of equivalence classes that have one-to-one correspondence to the real-world entities mentioned in the document. In other words, no two extracted entities denoting different real-world entities may be part of the same equivalence class, and no two extracted entities denoting the same real world entity may belong to different equivalence classes.

It is frequently helpful to compartmentalize the relation $coref(x, y)$ and, hence, the coreference resolution task into three different subtasks corresponding to different kinds of entities involved. More precisely, if x or y is a pronominal entity, then we obtain a pronoun resolution problem. Otherwise, if x or y is a nominal entity, then we have a noun phrase resolution problem. Finally, if both x and y are named entities, then it is a name resolution problem.

An information extraction system needs to address all three aspects of the coreference resolution problem. Yet different modeling and algorithmic choices may be appropriate for name, noun phrase, and pronoun resolution. In fact, there is hardly a paper on coreference resolution that addresses all of the three problems at the same time. Most existing studies concentrate on either pronoun or noun phrase coreference resolution. We now survey the related work in the field.

Most early work on coreference and anaphora resolution dealt with pronoun coreference [71, 62].

The early approaches identified a set of pronouns in a document, and, for each pronoun, sought to determine the best antecedent. Different definitions of "best" led to different carefully designed and complex rules that were sometimes based on existing discourse theories [103].

The area of pronoun and noun phrase coreference resolution was greatly revitalized since mid-1990s by application of learning approaches to the problem. We note, amongst many, the work of [13, 77, 86, 85].

We consider the following components in adaptive approaches to coreference resolution.

- Coreference examples and their feature representation.
- Coreference examples generation process.
- Learning algorithms for coreference classifiers.
- Decoding algorithm that combines predictions of coreference classifiers into a coherent discourse interpretation.

Let us examine the components in more detail.

A coreference example represents a pair of entities. For noun phrase and pronominal coreference, the pair is usually comprised of a nominal or pronominal anaphor and its candidate antecedent. A coreference example is a feature-based representation of a pair of entities that is designed to make manifest the properties of the anaphor and its candidate antecedent that are most helpful in making the decision whether the anaphor indeed refers to the antecedent in question. For instance, the popular features are shown below.

- Distance feature (number of sentences between an anaphor and an antecedent).
- Number agreement feature (true if and only if an anaphor and an antecedent are both singular or both plural).
- Gender agreement features (true if an anaphor and an antecedent have the same gender; false, they have different gender; unknown, if gender cannot be determined).
- Semantic class agreement (true if an anaphor and an antecedent belong to the same semantic class in a certain semantic taxonomy (e.g., Wordnet)).

Most learning-based systems for coreference resolution employed larger hand-crafted feature sets [86]. A coreference example has a binary label reflecting whether the entities that constitute the example are indeed coreferent or not.

There are two common approaches to generating coreference examples. First, since an example is generated for a pair of entities, every feasible anaphor/antecedent pair can be used to produce coreference examples. Such a generation process, however, leads to an extremely large set of examples even for small corpora annotated with coreference information since the number of examples generated per document is quadratic in the number of document entities. Moreover, many examples generated in this fashion correspond to entities residing far from one another in the original documents, and therefore, at least for nominal and pronominal coreference resolution, they do not carry much information of relevance to coreference decisions.

An alternative approach to generation of coreference examples, proposed by [86], ameliorates for the drawbacks of the all-pairs approach by focusing on examples that are most useful from the coreference perspective. The example generation algorithm proceeds from a fixed anaphora backward (in text), and generates a negative example for each candidate antecedent until an antecedent coreferent with the anaphor is encountered. A positive example is generated for the antecedent, and the process of generating examples, for the fixed anaphor, stops. Much fewer examples are generated in this manner, and positional information is naturally incorporated in the examples. In Section 4.4, we describe an extension of this example generation process that we adopt in our experiments.

Few learning algorithms have been experimentally evaluated on the coreference resolution problem. Most published studies employed a decision tree algorithm [13, 85], with the notable exception of a unified probabilistic modeling approach of [28].

In the absence of training data, we note application of clustering for coreference of noun phrases [26]. Namely, the noun phrase attributes are used to define a distance function that is used within a heuristic clustering algorithm to produce a clustering of noun phrases that aims to correspond to the coreference partition of the corresponding noun phrase entities.

Finally, the coreference decoding procedure that combines the predictions of coreference classifiers into a single coherent interpretation did not yet receive much attention in the literature. The

most prevalent decoding approach is to classify an ordered list of candidate antecedents for a fixed anaphora and select either the first one that is predicted to be the true antecedent by the classifier [86], or the most confident one [85]. Our major contribution is the introduction of an alternative decoding framework that yields more coherent interpretation of coreference classifier predictions.

4.3 Entity Classification

We approach the problem of entity extraction from the classification perspective. We take an existing generic shallow parsing system as a starting point of our investigation. Examples of such systems abound [9, 11]. In our experiments presented in Section 4.6, we employ the manually built shallow parsing component of [15].

The goal of our entity extraction approach is to transform a generic name and noun phrase output by a shallow parsing system into task-specific extracted entities. More precisely, we consider each name and noun phrase that are part of a produced shallow parse and re-classify them into the vocabulary of a particular extraction task.

The classification approach to entity extraction is modest yet immensely practical. The task of adapting the general shallow representation of text to specific needs provides a very cost-efficient way of reusing general linguistic mechanisms for new tasks and domains.

Let us consider the sentence, `Police have arrested a man carrying a live hand grenade at Gatwick Airport and evacuated one terminal`, and restrict the extraction task to that of *people*, *organizations*, and *locations*. In the sentence, there is a single named entity, `Gatwick Airport` and several noun phrases: `Police`, `a hand grenade`, `a man`, `a man carrying a live hand grenade`, and `one terminal`. According to our extraction task, the named entity `Gatwick airport` and the noun phrase `Police` are classified as *organizations*, the (embedded) noun phrases `a man` and `a man carrying a live hand grenade` are classified as *people*, the noun phrase `one terminal` is classified as *location*, and the noun phrase `hand grenade` is not part of the extraction vocabulary so no entity type is assigned to it.

During the training stage of the entity classification, an example is generated for every generic name and noun phrase produced by the shallow parser. An example is labeled with the type of the extracted entity in the task-specific extraction vocabulary. Naturally, many extracted generic

entities will not correspond to any of the task-specific entities; therefore, many of the generated examples will have no labels.

For instance, for the nominal entity `a man carrying a live hand grenade`, we will use the features of the noun phrase (e.g, its head `man`) as well as the context around the noun phrase (e.g., the preceding verb `arrested`) to generate the nominal entity classification example.

We will use the training examples to learn a multiclass classifier Cl in the following manner. We decompose the multiclass learning task into a set of binary learning tasks, where each binary classifier Cl_t corresponds to a type t of extracted entities. Therefore, a generated example becomes a positive example for Cl_t, if it is labeled with t, and a negative example for Cl_t, otherwise. In Section 4.6, we experiment with a number of learning algorithms for the task.

In the spirit of the SNOW system, we require each of the learned classifiers Cl_t to output a score $score_t(con)$ given the context con of an entity to be classified. We deem that the classifier Cl_t predicts the label t if and only if the corresponding score $score_t(con)$ is positive.

For some learning algorithms and the corresponding classifiers, the scores $score_t(con)$ can be converted into conditional probabilities $p_{Cl}(t|con)$ denoting the probability of the type t given the context con.

During the evaluation stage, we generate an example for each of the generic names and noun phrases produced by a generic shallow parsing systems and apply all of the binary classifiers to the example, thereby obtaining predictions for each possible entity. If none of the binary type classifiers predicted a label for the example, we assert that the corresponding generic entity is not to be extracted. If one or more binary classifiers predicted labels for the generic entity, we assign to the entity the type corresponding to the prediction with the largest score.

Thus, the entity classification system outputs a set of entities for a document, with each entity assigned a predicted entity type. The following task of coreference resolution is aimed at aggregating the extracted entities that correspond to single real-world entity.

4.4 Coreference Resolution

The goal of the coreference resolution module is to partition the extracted entities of the same type into a set of equivalence classes. Each equivalence class will correspond to a single real-world entity

that aggregates a set of extracted entities in a document.

We will describe two approaches to coreference resolution. For each approach, we delineate the example generation and decoding algorithms. The particular feature representations and algorithms employed will be described as part of the experiments in Section 4.6. We will also present two theoretical reformulations of the coreference resolution task that reduce it to problems of continuous optimization.

4.4.1 Preliminaries

Let $E = e_1, e_2, \ldots, e_n$ a set of extracted entities. In the remainder of this section, we assume that the entities are of the same type since coreference resolution processing steps of entities of different types can be conducted independently of one another. Note that in Section 4.5.1, where entity classification is interleaved with the coreference resolution process, we relax this assumption.

At the training stage, the entities are partitioned in a set of equivalence classes: $E = E_1 \cup E_2 \cup \cdots E_k$, where $E_i \cap E_j = \emptyset,\ i \neq j$. At the evaluation stage, we seek to produce such a partition.

Let c be a classifier that given a pair of entities (e_i, e_j) determines whether they are coreferent or not. Formally, $c : E \times E \rightarrow \{-1, 1\}$ where the label of 1 corresponds to entity coreference and the label of -1 corresponds to absence thereof.

At the training stage, we use the equivalence class information to learn the coreference classifier for a pair of entities. At the evaluation stage, we apply the classifier to produce the coreference decisions.

Application of the coreference classifier is not straightforward, since the equivalence class constraint leads to dependence between individual classifier predictions. For example, if the entity pairs (e_1, e_2), (e_2, e_3) are classified as coreferent, and the pair (e_1, e_3) is classified as non-coreferent, then we have a contradiction, because transitivity of the equivalence class relation implies that the pair (e_1, e_3) is also coreferent. We need a way to resolve the contradictions in a principled manner.

We call an algorithm that combines predictions of individual coreference classifiers a *coreference decoding algorithm*. Additionally, a particular decoding algorithm needs to be coupled with a corresponding *example generation algorithm*. In the following discussion, we introduce two coreference decoding algorithms and their corresponding example generation algorithms and evaluate them

experimentally.

We will introduce additional notation to streamline our presentation. Let R be a binary relation over a set X. We denote by R^* the transitive closure of the binary relation R.

We incorporate additional knowledge in the coreference resolution process by imposing constraints on the set of possible anaphor/antecedent pairs. Note that, for the sake of brevity, we use the words "anaphor" and "antecedent" for both anaphora and cataphora phenomena. The constraints are specified by the following relation $IsCandidateAntecedent$:

$$isCandidateAntecedent(anaphor, antecedent) =$$
$$= \begin{cases} 1, & \text{if } anaphor \text{ is pronominal, and } anaphor \text{ follows antecedent} \\ 1, & \text{if } anaphor \text{ is pronominal, and } antecedent \text{ is not pronominal} \\ 1, & \text{if } anaphor \text{ is nominal, and } antecedent \text{ is not pronominal, and it precedes } anaphor \\ 1, & \text{if } anaphor \text{ is nominal, and } antecedent \text{ is a name} \\ 1, & \text{if } anaphor \text{ is a name, and } antecedent \text{ is a name that precedes } anaphor \\ 0, & \text{otherwise} \end{cases}$$

The constraints restrict the list of possible antecedents for different classes of anaphora by incorporating coreference knowledge. The knowledge specifies that pronominal anaphora never refer forward to other pronouns, that nominal anaphora refer to preceding nominals or names, and names refer only to preceding names.

4.4.2 Sequential Transitive Coreference Decoding

The sequential transitive decoding algorithm is a standard decoding algorithm used for coreference resolution [86]. We describe the example generation and decoding components of the algorithm.

The example generation algorithm is shown as Algorithm 3. For each entity e_i in the document, the algorithm proceeds backward in the document and generates a negative example for each candidate antecedent e_j non-coreferent with e_i until the first candidate antecedent e_k coreferent with e_i is encountered. The algorithm then generates a single positive example for the entity e_k. If there are no preceding entities coreferent with e_i, the algorithm proceeds forward in the document and generates examples in the same manner.

In order to reduce the number of examples generated, the algorithm utilizes the fact that antecedent and anaphor are usually close in text, and generates examples for only D preceding and D following antecedents for a fixed anaphora, where D is an input parameter to the algorithm.

Algorithm 3 The Sequential Coreference Example Generation Algorithm

$(e_1, e_2, \ldots, e_n)$ is the list of entities ordered by their location in the document
$coref$ is the true coreference relation
D is the maximum distance between an anaphor and an antecedent
for all $i = 1, 2, \ldots, n$ **do**
  $anaphor = e_i$
  $j = i - 1$
  $d = \max(1, i - D)$
  while $j \geq d$ and ((not $isCandidateAntecedent(anaphor, e_j)$) or $coref(e_i, e_j) = -1$) **do**
    if $isCandidateAntecedent(anaphor, e_j)$ **then**
      generate negative coreference example for the pair (e_i, e_j)
    end if
    $j = j - 1$
  end while
  if $j \geq d$ **then**
    generate positive coreference example for the pair (e_i, e_j)
  else
    $j = i + 1$
    $d = \min(n, i + D)$
    while $j \leq d$ and ((not $isCandidateAntecedent(anaphor, e_j)$) or $coref(e_i, e_j) = -1$) **do**
      if $isCandidateAntecedent(anaphor, e_j)$ **then**
        generate negative coreference example for the pair (e_i, e_j)
      end if
      $j = j + 1$
    end while
    if $j \leq d$ **then**
      generate positive coreference example for the pair (e_i, e_j)
    end if
  end if
end for

Let us consider the following sentence, `A woman who ran down her`$_1$ `cheating husband with her`$_2$ `Mercedes after catching him with his mistress was convicted of murder Thursday in a real-life Texas soap opera in which she claimed it was all a tragic accident.` From this sentence, we will generate coreference examples for the following anaphor/antecedent pairs for *person* entities, in the order of generation (+ or − sign indicates whether the example is positive or negative).

- (("her$_1$","a woman"), +)
- (("her$_2$","her$_1$"), +)
- (("her cheating husband","a woman"), −)
- (("him","her$_2$"), −)
- (("him","her cheating husband"), +)
- (("his","him"), +)
- (("his mistress","her cheating husband"), −)
- (("his mistress","a woman"), −)
- (("she","his mistress"), −)
- (("she","his"), −)
- (("she","him"), −)
- (("she","her$_2$"), +)

The sequential transitive decoding algorithm is shown as Algorithm 4. The algorithm processes entities sequentially in order of their appearance in the document. The algorithm establishes a coreference relation between an entity e_i and the closest preceding entity classified as coreferent with e_i by the learned coreference classifier. If no such preceding entity is found, the algorithm establishes a coreference relation between the entity e_i and the closest following entity classified as coreferent with e_i. After a single pass through the document, the equivalence classes are constructed via the transitive closure of the established coreference relations.

The following theorem analyzes the computational complexity of the sequential transitive decoding algorithm.

Theorem 8 *The running time of the sequential transitive decoding algorithm is $O(nDq)$, where q is the complexity of evaluating the coreference classifier c.*

Algorithm 4 The Sequential Transitive Decoding Algorithm

$(e_1, e_2, \ldots, e_n)$ is the list of entities ordered by their location in the document
c is the coreference classifier
D is the maximum distance between an anaphor and an antecedent
for all $i = 1, 2, \ldots, n$ **do**
  $j = i - 1$
  $d = \max(1, i - D)$
  while $j \geq d$ and ((not $isCandidateAntecedent(anaphor, e_j)$) or $coref(e_i, e_j) = -1$) **do**
    $j = j - 1$
  end while
  if $j > 0$ **then**
    $coref(e_i, e_j) = 1$
  else
    $j = i + 1$
    $d = \min(n, i + D)$
    while $j \leq d$ and ((not $isCandidateAntecedent(anaphor, e_j)$) or $c(e_i, e_j) = -1$) **do**
      $j = j + 1$
    end while
    if $j \leq d$ **then**
      $coref(e_i, e_j) = 1$
    end if
  end if
end for
Output $coref^*$

Proof: For each entity, the algorithm evaluates the coreference classifier at most $2D$ times. The complexity bound follows. □

Note that the parameter D represents the natural trade-off in the decoding algorithm between the decoding accuracy and the decoding time.

The application of the sequential transitive decoding algorithm to the sentence, `A woman who ran down her`$_1$ `cheating husband with her`$_2$ `Mercedes after catching him with his mistress was convicted of murder Thursday in a real-life Texas soap opera in which she claimed it was all a tragic accident`, with the correct coreference classifier, will mirror the presented process of example generation and lead to the following entity equivalence classes:

- "a woman", "her$_1$", "her$_2$", "she"
- "her cheating husband", "him", "his"
- "his mistress"

In Section 4.6, we experimentally evaluate the sequential transitive decoding algorithm for coreference resolution.

4.4.3 Loss-based Coreference Decoding

We introduce a formal framework for coreference decoding based on the classification framework introduced in Chapter 2. We formalize the coreference decoding problem in the framework, analyze its hardness, and propose an approximate algorithm for decoding problem solution.

Let A be a learning algorithm for learning a coreference classifier c mapping a pair of entities (e_1, e_2) to $\{-1, 1\}$. Let l be a loss function that is being minimized by A.

Let $E_1, E_2, \ldots, E_k$ be a partition of E. Define the variable e_{ij} that indicates whether two entities belong to the same equivalence class, as follows:

$$e_{ij} = \begin{cases} 1, & \text{if } \exists l \in \{1, \ldots, k\},\ e_i \in E_l \text{ and } e_j \in E_l \\ -1, & \text{otherwise} \end{cases}$$

Let $\mathcal{E} = \{e_{ij}\}$ be an equivalence class partition of the entities E. Then, the partition induces the following loss with respect to the classifier c:

$$l(c, \mathcal{E}) = \sum_{i,j} l(e_{ij}, c(e_i, e_j)) \tag{4.1}$$

During training, the algorithm A learns the classifier c that minimizes (4.1), given the partition of E. During evaluation, the decoding algorithm searches for the partition $\mathcal{E}^*$ that minimizes the partition loss (4.1), given the classifier c.

$$\mathcal{E}^* = argmin_{\mathcal{E}} l(c, \mathcal{E})$$

4.4.4 Problem Hardness

For hardness considerations, we restrict our attention to the 0-1 loss function l. It is helpful to re-interpret the problem (4.6) in graph-theoretic terms. Let c be a coreference classifier output by a learning algorithm minimizing the 0-1 loss function. Consider the complete weighted graph

$G = (V, E)$ whose node v_i corresponds to the entity e_i and the edge (v_i, v_j) has the weight -1 or +1. We seek to partition the resulted graph in a set of connected components, and we are penalized for each positive edge lying between the partition components and for each negative edge lying within a component. Note that in contrast to the standard formulation of graph-theoretic partition problems, the number of partition is not fixed. In addition, the edge weights are allowed to be both positive and negative.

We also observe that, if the coreference classifier c is perfect, the coreference equivalence classes can be easily constructed by removing all negative edges from the graph and equating each resulted connected component with an equivalence class.

It turns out that the 0-1 loss function formulation of the coreference decoding problem is an instance of a recently analyzed *correlation clustering* problem [16]. The analysis implies that the problem is NP-hard. Therefore, we have to resort to heuristics to induce the equivalence class partition efficiently.

4.4.5 Semi-separable Loss Functions

Let us introduce an additional restriction on the loss function.

Definition 5 *Let c be a classifier that produces a score of $c_h(x)$ on an example x. A loss function $l(y, c_h(x))$ is termed* semi-separable*, if there is a function $f : \mathbb{R} \to \mathbb{R}$ such that*

$$l(y, c_h(x)) = \max(0, -yf(x)) \tag{4.2}$$

Some natural loss functions are semi-separable. For example, the 0-1 loss function is semi-separable with f being sgn function.

We now show that for semi-separable loss functions, the partition optimization problem with loss (4.1) can be replaced with a more manageable equivalent optimization problem.

Theorem 9 *Let $\mathcal{E} = \{e_{ij}\}$ and $\mathcal{E}' = \{e'_{ij}\}$ be two partitions of E. Then*

$$\sum_{i,j} \max(0, -e_{ij} f(e_i, e_j)) \leq \sum_{i,j} \max(0, -e'_{ij} f(e_i, e_j)) \tag{4.3}$$

if and only if

$$\sum_{i,j} e_{ij} f(e_i, e_j) \geq \sum_{i,j} e'_{ij} f(e_i, e_j) \tag{4.4}$$

Proof: Denote

$$I_a = \{(i,j) : e_{ij} = e'_{ij}\} \qquad I_d = \{(i,j) : e_{ij} = -e'_{ij}\}$$
$$I_a^+ = \{(i,j) \in I_a : -e_{ij} f(e_i, e_j) \geq 0\} \quad I_a^- = \{(i,j) \in I_a : -e_{ij} f(e_i, e_j) < 0\}$$
$$I_d^+ = \{(i,j) \in I_d : -e_{ij} f(e_i, e_j) \geq 0\} \quad I_d^- \{(i,j) \in I_d : -e_{ij} f(e_i, e_j) < 0\}$$

I_a and I_d are the sets of index pair where partitions e_{ij} and e'_{ij} agree and disagree, respectively. The superscripted subsets of I_a and I_d determine the subsets where the $e_{ij} f(e_i, e_j)$ is positive or negative. Let us denote $f_{ij} = e_{ij} f(e_i, e_j)$ and $f'_{ij} = e'_{ij} f(e_i, e_j)$. Then, (4.4) can be written as

$$\sum_{(i,j) \in I_a} f_{ij} + \sum_{(i,j) \in I_d} f_{ij} \geq \sum_{(i,j) \in I_a} f'_{ij} + \sum_{(i,j) \in I_d} f'_{ij}$$

Since $f_{ij} = f'_{ij}$ for $(i,j) \in I_a$ and $f_{ij} = -f'_{ij}$ for $(i,j) \in I_d$, we obtain that (4.4) is equivalent to

$$\sum_{(i,j) \in I_d} f_{ij} \geq 0$$

Similarly, (4.3) can be written as

$$-\sum_{(i,j) \in I_d^+} f_{ij} \leq -\sum_{(i,j) \in I_d^-} f'_{ij}$$

whence

$$0 \leq \sum_{(i,j) \in I_d^-} f_{ij} + \sum_{(i,j) \in I_d^+} f_{ij} = \sum_{(i,j) \in I_d} f_{ij}$$

□

Corollary 1 *Let l be a semi-separable loss function. Then,*

$$\min_{\mathcal{E}} \sum_{i,j} l(e_{ij} f(e_i, e_j)) = \max_{\mathcal{E}} \sum_{i,j} e_{ij} f(e_i, e_j) \tag{4.5}$$

Hence, for semi-separable loss functions, the optimization objective can be written as follows.

$$\sum_{i,j} e_{ij} w_{ij} \rightarrow \max \tag{4.6}$$

where $w_{ij} = f(e_i, e_j)$.

Now observe that minimizing the functional (4.6), with respect to $\mathcal{E}$, is equivalent to finding a minimum cut of the graph G, i.e., partition of the graph vertex set V into disjoint sets $V_1, V_2, \ldots, V_k$, so that the sum of weights edges between leading from V_i to V_j, $i \neq j$, is minimized. Indeed, let $W = \sum_{i,j} w_{ij}$, $W^+ = \sum_{i,j,e_{ij}=1} w_{ij}$, and $W^- = \sum_{i,j,e_{ij}=-1} w_{ij}$. Note that W^- is the value of the graph cut and $W^+ = W - W^-$. Then

$$\max_{\mathcal{E}} \sum_{ij} e_{ij} w_{ij} = \max_{\mathcal{E}} (W^+ - W^-) = \max_{\mathcal{E}} (W - 2W^-) = \min_{\mathcal{E}} (W^-)$$

Note that in contrast to the standard (weighted) mincut formulation, the number of partitions k is not fixed. Additionally, the weights w_{ij} are not necessarily positive. For example, if we fix $k = 2$, then, for positive weights, there are numerous efficient mincut algorithms applicable to the problem [29]. On the other hand, for a fixed $k \geq 3$ and/or negative weights w_{ij}, $i \neq j$, the optimization problem is NP-hard [45]. In our case, for a variable k, the assumption of all positive (negative) weights leads to the trivial one-class (n classes) solution.

Greedy Decoding

The greedy decoding algorithm incrementally optimizes (4.1). The example generation component and the decoding component of the greedy algorithm for semi-separable loss functions are shown as Algorithm 5 and Algorithm 6, respectively.

The example generation algorithm for greedy decoding generates coreference examples for every eligible pair of entities in the document. The algorithm may lead to a very large number of examples

generated. We reduce the number of examples by restricting possible the distance between an anaphor and an antecedent.

Algorithm 5 The Example Generation Algorithm for Greedy Decoding

$(e_1, e_2, \ldots, e_n)$ is the list of entities ordered by their location in the document
$coref$ is the true coreference relation
D is the maximum distance between an anaphor and an antecedent
$I = \{\{1\}, \{2\}, \ldots, \{n\}\}$
for all $(e_i, e_j),\ \{i\} \in I, \{j\} \in I, |i - j| \leq D$ **do**
  if $isCandidateAntecedent(e_i, e_j)$ **then**
    generate an example for (e_i, e_j) with label $coref(e_i, e_j)$
  end if
end for

Let us consider the example of example generation for the sentence presented in section 4.4.2. In addition to all examples shown therein, the Algorithm 5 will also generate the following examples (with D being large);

- (("her$_1$","her cheating husband"), $-$)
- (("her$_1$","his mistress"), $-$)
- (("her$_2$","a woman"), $+$)
- (("her$_2$","her cheating husband"), $-$)
- (("her cheating husband","his mistress"), $-$)
- (("him","her$_1$"), $-$)
- (("him","a woman"), $-$)
- (("him","his mistress"), $-$)
- (("his","her cheating husband"), $+$)
- (("his","her$_2$"), $+$)
- (("his","her$_1$"), $+$)
- (("his","a woman"), $+$)

Algorithm 6 The Greedy Decoding Algorithm

$(e_1, e_2, \ldots, e_n)$ is the list of entities ordered by their location in the document
c is the true coreference classifier
D is the maximum distance between an anaphor and an antecedent
$I = \{\{1\}, \{2\}, \ldots, \{n\}\}$
for all $(e_i, e_j),\ \{i\} \in I, \{j\} \in I$ **do**
  if $|i - j| \leq D$ **then**
    if $isCandidateAntecedent(e_i, e_j)$ **then**
      $w_{\{i\},\{j\}} = c_h(e_i, e_j)$
    end if
  else
    $w_{\{i\},\{j\}} = 0$
  end if
end for
Sort $\{w_{\{i\},\{j\}}\}$
$w_{max} = w_{\mathbf{i}^*,\mathbf{j}^*} = \max_{\mathbf{i} \in I, \mathbf{j} \in I \setminus \mathbf{i}} w_{\mathbf{i},\mathbf{j}}$
while $|I| > 1$ *and* $w_{max} > 0$ **do**
  $I = I \setminus \{\mathbf{i}, \mathbf{j}\}$
  for all $\mathbf{k} \in I$ **do**
    $w_{\mathbf{k},\mathbf{i}^* \cup \mathbf{j}^*} = w_{\mathbf{k},\mathbf{i}^*} + w_{\mathbf{k},\mathbf{j}^*}$
  end for
  $I = I \cup \{\mathbf{i}^* \cup \mathbf{j}^*\}$
  $w_{max} = w_{\mathbf{i}^*,\mathbf{j}^*} = \max_{\mathbf{i} \in I, \mathbf{j} \in I \setminus \mathbf{i}} w_{\mathbf{i},\mathbf{j}}$
end while

- (("his","his mistress"), $-$)
- (("she","her$_1$"), $+$)
- (("she","a woman"), $+$)

It is clear that a lot more examples are generated in this manner compared with the sequential decoding example generation examples. The examples enjoy significant redundancy, which shall help coreference classifiers to better focus on correct coreference decisions during the decoding process.

The greedy algorithm for maximizing the weight is shown as Algorithm 6. In the algorithm, we use bold $\mathbf{i}$ to denote sets of indices. The greedy algorithm seeks to exploit the redundancy prevalent among the coreference examples and make coreference decisions based on multiple coreference predictions.

The greedy decoding algorithm initially puts each extracted entity into a separate equivalence class and then iteratively merges the equivalence classes, while the merge improves the cumulative

loss function (4.1). During each iteration, the pair of classes is selected in a greedy fashion to optimize the loss weight improvement achieved by the merge. Note that the algorithm iteratively updates the weights between the already merged equivalence classes.

The greedy decoding may help prevent mistakes made by the sequential decoding algorithm. In our coreference example, the sequential decoding algorithm may incorrectly corefer the pronoun `she` with the noun phrase `his mistress`, since this decision will be considered in isolation from other coreference decisions. The greedy algorithm may, however, corefer the pronoun `she` with other pronouns $\texttt{her}_2$ and $\texttt{her}_1$, and only then attempt to resolve the three pronouns together to one of the two noun phrases (`a woman` and `his mistress`) and by aggregating the coreference information may be less likely to make the resolution mistake.

The following theorem analyzes the computational complexity of the greedy decoding algorithm for coreference resolution.

Theorem 10 *The computational complexity of the greedy decoding algorithm is* $O(nD(q+\log(nD))$, *where* q *is the time complexity of the evaluating the coreference classifier* c.

Proof: The algorithm computes the coreference classifier for every pair of entities lying at most D entities from each other in $O(nDq)$ time. The computed coreference weights are then sorted in $O(nD\log(nD))$. The process of merging entities, in the worst case, requires $n-1$ iterations, where each iteration takes $O(D\log(nD))$ time, since an entity has non zero coreference weights with only $O(D)$ other entities, and the complexity of an insertion in the sorted list is $O(\log(nD))$. Hence, the complexity bound follows. □

4.4.6 Continuous Optimization Approach

In this section, we exhibit two reductions of the coreference resolution problem to continuous optimization problems. We will show that the optimal solutions to the continuous problems are necessary the optimal coreference partitions. The reductions are interesting reformulations of the coreference resolution problem, mostly from the theoretical perspective.

Pairwise Reduction

Let us make a variable change $x_{ij} = \frac{1}{2}(e_{ij} + 1)$ mapping the set $\{-1, 1\}$ to the set $\{0, 1\}$. Hence, $e_{ij} = 2(x_{ij} - 1)$, and the objective function in (4.6) becomes $\sum_{ij} w_{ij} e_{ij} = 2 \sum_{ij} w_{ij} x_{ij} - W$. Hence, the optimization problem (4.6) is equivalent to:

$$\begin{aligned} &\textstyle\sum_{i,j} w_{ij} x_{ij} \rightarrow \max \\ &x_{ii} = 1, \ \forall i = 1, \ldots, n \\ &x_{ij} = x_{ji}, \ \forall i, j = 1, \ldots, n \\ &x_{ij} = 1 \wedge x_{jl} = 1 \Rightarrow x_{il} = 1, \forall i, j, l = 1, \ldots, n \\ &x_{ij} \in \{0, 1\} \end{aligned} \tag{4.7}$$

Consider the matrix $X = (x_{ij})$. Note that if $x_{ij} = 0$, then the ith column x_i and the jth column x_j are orthogonal, i.e, $x_i \cdot x_j = 0$, and if $x_{ij} = 1$, then $x_i = x_j$ and $x_i \cdot x_j = k_i = \sum_{l=1}^{n} x_{il}$. Hence, any x_i is an eigenvector of the matrix X, with the eigenvalue of k_i, since

$$X \cdot x_i = k_i x_i$$

Whereby we get that

$$X \cdot X = K \cdot X \tag{4.8}$$

where $K = diag(k_1, \ldots, k_n)$ is a diagonal matrix, whose diagonal elements are eigenvalues of X. Note that for a partition with k equivalence classes, the matrix X is of rank k, whose only k non-zero eigenvalues are the cardinalities of the k equivalence classes.

We now show that the equation (4.8) provides necessary and sufficient conditions for the matrix X to represent a transitive relation. Since necessity follows from the above discussion, we only need to prove sufficiency. In fact, we will prove a stronger statement below.

Theorem 11 *Let $x_{ij} \in [0, 1]$, $x_{ii} = 1$, $x_{ij} = x_{ji}$, $i, j = 1, \ldots, n$, and X satisfies (4.8). Then, $x_{ij} \in \{0, 1\}$, and the matrix X represents an equivalence relation*

Proof: To prove that $x_{ij} \in \{0, 1\}$, observe that it follows from (4.8) that

$$\sum_{l=1}^{n} x_{il}^2 = x_{ii} \sum_{l=1}^{n} x_{il} = \sum_{l=1}^{n} x_{il}$$

$$\sum_{l=1}^{n} x_{il}(x_{il} - 1) = 0$$

Whence $x_{il} \in \{0, 1\}, \ \forall i, l$. Now note that (4.8) also implies that

$$\sum_{s} x_{is} x_{js} = x_{ij} \sum_{s} x_{is}, \forall i, j \tag{4.9}$$

Suppose that the relation represented by X is not transitive, hence, there exist indices i_1, j_1, s_1, such that $x_{i_1 s_1} = 1$, $x_{s_1 j_1} = 1$, and $x_{i_1 j_1} = 0$. But then, the right side in (4.9), for $i = i_1, j = j_1$, is equal to zero, and the left side is equal to $x_{i_1 s_1} x_{s_1 j_1} + \ldots > 0$. Contradiction. Thus, $x_{ij} = 1$, and the relation is transitive. □

Hence, the coreference resolution problem is equivalent to the following quadratic programming problem:

$$\begin{aligned}
&\textstyle\sum_{i,j} w_{ij} x_{ij} \rightarrow \max \\
&X \cdot X = K \cdot X \\
&K = diag(\textstyle\sum_j x_{1j}, \ldots, \sum_j x_{nj}) \\
&x_{ij} = x_{ji}, x_{ii} = 1, \forall i, j \\
&x_{ij} \in [0, 1]
\end{aligned} \tag{4.10}$$

We next present an alternative reduction that leads to a more manageable quadratic programming problem, with the number of constraints linear in n.

Equivalence Class Reduction

Let $\mathcal{E} = \{E_1, E_2, \ldots, E_k\}$ be a partition of entities E. Note that $k \leq n$, where $n = |E|$. We augment the partition with a set of empty equivalence classes $E_{k+1}, \ldots, E_n$, thereby fixing the number of equivalence classes to be n, for any partition.

Let us now introduce the variables y_{ij} denoting that the entity e_i belongs to the equivalence

class E_j:

$$y_{ij} = \begin{cases} 1, & \text{if } e_i \in E_j \\ 0, & \text{otherwise} \end{cases}$$

The constraints imposed on the variables y_{ij} only require that each entity be assigned to exactly one equivalence class:

$$\sum_{j=1}^{n} y_{ij} = 1, \ i = 1, \ldots, n \tag{4.11}$$

Now observe that the variables x_{ij} have the following relationship to the variables y_{ij}:

$$x_{ij} = \sum_{k=1}^{n} y_{ik} y_{jk} = y_i^T y_j$$

where $y_i = (y_{i1}, \ldots, y_{in})$.

Therefore, the objective function of (4.7) can be written as

$$\sum_{ij} w_{ij} y_i^T y_j$$

Thus, we obtain the alternative formulation of the coreference resolution problem.

$$\begin{aligned} \sum_{ij} w_{ij} y_i^T y_j &\rightarrow \max \\ \sum_{j=1}^{n} y_{ij} = 1, \ i &= 1, \ldots, n \\ y_{ij} &\in \{0, 1\} \end{aligned} \tag{4.12}$$

We will now show that (4.12) can be reduced to an equivalent continuous quadratic programming problem. We present the reduction in two steps. First, we show we can eliminate the constraints (4.11). Then, we prove that an optimal solution to the resulting unconstrained program in the interval $[0, 1]$ is a 0-1 vector.

Lemma 1 *Let* $W = 2 \sum_{ij} |w_{ij}| + 1$. *Then, the optimization problem (4.12) is equivalent to the*

following unconstrained problem.

$$\sum_{ij} w_{ij} y_i^T y_j + W \sum_{i=1}^{n} \left(\sum_{j=1}^{n} y_{ij} - \sum_{j_1 \neq j_2} y_{ij_1} y_{ij_2} \right) \to \max \tag{4.13}$$

Proof: The lemma follow from the observation that $W \sum_{i=1}^{n} (\sum_{j=1}^{n} y_{ij} - \sum_{j_1 \neq j_2} y_{ij_1} y_{ij_2}) = W$ if $\sum_{j=1}^{K} y_{ij} = 1$, and $W \sum_{i=1}^{n} (\sum_{j=1}^{n} y_{ij} - \sum_{j_1 \neq j_2} y_{ij_1} y_{ij_2}) \leq 0$, otherwise. □

Observe that incorporating the constraints into the objective function (4.4.6) did not change the form of the objective function (for brevity, we introduce the single-indexed variables z to denote double-indexed variables y and let $N = n^2$):

$$f(z_1, \ldots, z_N) = \sum_{i \neq j} w_{ij} z_i z_j + \sum_i z_i \to \max \tag{4.14}$$

Theorem 12 ([22]) *An optimal solution to the problem (4.14) on the interval* $[0, 1]$ *is a 0-1 vector.*

Proof: Let $\zeta = (\zeta_1, \ldots, \zeta_N) \in [0, 1]^n$ be an optimal solution of (4.14). We will construct a 0-1 vector $(z_1^*, \ldots, z_N^*)$ such that $f(z^*) \geq f(\zeta)$. Denote

$$df_i(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_N) = f(z_1, \ldots, z_{i-1}, 1, z_{i+1}, \ldots, z_N) - f(z_1, \ldots, z_{i-1}, 0, z_{i+1}, \ldots, z_N)$$

$$r_i(z) = f(z) - df_i(z) z_i$$

Note that both $df_i(z)$ and $r_i(z)$ do not depend on z_i.

Let us now replace ζ with $\zeta^{(1)}$ such that $\zeta_1^{(1)} = 1$ if $df_1(\zeta) \leq 0$ and $\zeta_1^{(1)} = 0$ otherwise. Also, $\zeta_i^{(1)} = \zeta_i$ for $i \neq 1$. Then,

$$f(\zeta^{(1)}) - f(\zeta) = r_1(\zeta^{(1)}) - r_1(\zeta)) + (df_1(\zeta^{(1)}) - df_1(\zeta))(\zeta_1^{(1)} - \zeta_1) = -df_1(\zeta)(\zeta_1^{(1)} - \zeta_1) \geq 0$$

By applying the procedure recursively to $\zeta^{(1)}, \zeta^{(2)}, \ldots$ we obtain the 0-1 vector $z^* = \zeta^{(N)}$, and

$$f(z^*) = f(\zeta^{(N)}) \geq f(\zeta^{(N-1)}) \geq \cdots \geq f(\zeta^{(1)}) \geq f(\zeta)$$

Hence, either $f(\zeta) = f(z^*)$ or ζ is not an optimal solution of (4.14). $\square$

While the presented reductions provide elegant theoretical formulations of coreference resolution, from a mathematical programming perspective, we note that finding a local maximum of a quadratic program with n^2 variables takes, in general, $O(n^6)$ [76]. The complexity bound makes the approach impractical. Moreover, the local maximum is not necessarily a global maximum, for, as we indicated in Section 4.4.4, the global optimization of cumulative loss function for coreference resolution is, in general, an NP-hard problem.

4.5 Combining Entity Classification and Coreference Resolution

In the above presentation, we considered entity classification and coreference resolution as two independent tasks. There are examples, however, when coreference resolution and entity classification can hardly be considered in isolation, and significant benefit in the accuracy of tasks can be achieved by exploiting the synergy between them.

Let us consider the following sentences. `The negotiations between the two countries ended successfully. They both confirmed plans for further cooperation`. The pronoun `they` in the second sentence is a pronominal entity with type `location` referring to `the two countries`. Yet the immediate context of `the` makes correct type classification by itself a very difficult task. On the other hand, the decision of co-referring `they` with `the two countries` is a fairly simple one. Thus, in this case, it is appropriate to delay the entity type classification task until after coreference resolution is done. In fact, correct coreference information unambiguously determines the entity type of `they`.

The observation leads us combining entity type classification with coreference resolution and introducing the idea of delaying type classification to the stage of coreference resolution. We therefore remove the assumption that all entities have the same type at the coreference resolution stage. Instead, we assume that an entity type classifier Cl outputs a conditional probability distribution of types $p_{Cl}(t_i|con_i)$ given the syntactic context con_i of the entity e_i.

We also assume that that possible types for different entities are pairwise conditionally inde-

pendent, given the local context information for entities:

$$p(t_i, t_j | con_i, con_j) = p_{Cl}(t_i | con_i) p_{Cl}(t_j | con_j)$$

The loss of coreferring two entities e_i and e_j depends on the types that they are assigned. Let $e_i(t)$ denote the entity e_i instantiated with type t and c be a coreference classifier, as before. Note, the coreference classifier can be applied to the pair of entities $e_i(t_i)$ and $e_j(t_j)$ only if $t_i = t_j$ since only entities with the same type can corefer. Then,

$$l(e_{ij}, t_i, t_j) = \begin{cases} p_{Cl}(t_i | con_i) p_{Cl}(t_j | con_j) l(e_{ij}, c(e_i(t_i), e_j(t_j))), & \text{if } t_i = t_j \\ \infty, & \text{otherwise} \end{cases}$$

Let $\mathcal{T}$ be a type assignment to all entities $e_1, \ldots, e_n$. The cumulative partition and type assignment loss, with respect to the coreference classifier c and type classifier Cl, can be written as follows:

$$l(c, Cl, \mathcal{E}, \mathcal{T}) = \sum_{t_i = t_j} p_{Cl}(t_i | con_i) p_{Cl}(t_j | con_j) l(e_{ij}, c(e_i(t_i), e_j(t_j))) \tag{4.15}$$

Thus, the goal of the decoding algorithm is to search for the partition $\mathcal{E}^*$ and the type assignment $\mathcal{T}^*$ that minimize the partition and type assignment loss (4.15), given the coreference classifier c and the type classifier Cl:

$$(\mathcal{E}^*, \mathcal{T}^*) = argmin_{(\mathcal{E}, \mathcal{T})} l(c, Cl, \mathcal{E}, \mathcal{T})$$

4.5.1 Greedy Coreference Decoding and Entity Classification Algorithm

For semi-separable loss functions, let us denote

$$w_{ij}(t_i, t_j) = \begin{cases} p_{Cl}(t_i | con_i) p_{Cl}(t_j | con_j) c_h(e_i(t_i), e_j(t_j)), & \text{if } t_i = t_j \\ -\infty, & \text{otherwise} \end{cases}$$

We note that the analogue of Theorem 9 can be proved for the combined setting as well, which implies that the objective function for the combined type and coreference decoding has the following

form:

$$\sum_{t_i=t_j} w_{ij}(t_i,t_j)e_{ij} \rightarrow_{\mathcal{E},\mathcal{T}} \max \tag{4.16}$$

We now adapt the greedy decoding algorithm for coreference resolution presented in Section 4.4.5 to the combined coreference and type classification setting.

The modified greedy decoding algorithm for semi-separable loss functions involving type classification is shown as Algorithm 7. The algorithm computes, for every entity, the probability distribution of types conditional upon the entity context. For every eligible pair of entities and every type, the algorithm computes the weight of merging the entities and assigning them the types. Initially, each entity is assigned to a separate equivalence class. Then, the greedy merging process is interleaved with assignment of types. We use the notation $type(e_i) = t$ to assign the type t to the entity e_i. Once two equivalence classes are selected for a merge, the merge also fixes the types of entities in the equivalence classes. The merging process stops when no merge can further improve the objective function (4.15).

The following theorem analyzes the computational complexity of the greedy decoding algorithm for coreference resolution and type classification.

Theorem 13 *The computational complexity of the Algorithm 7 is* $O(n(r + D|T|\log(nD|T|) + Dq))$, *where* q *and* r *are the time complexities of the evaluating the coreference classifier* c *and the entity type classifier* Cl, *respectively, and* $|T|$ *is the number of possible entity types.*

Proof: The time complexity of evaluating the type classifier on all entities is $O(nr)$. The time complexity of evaluating the coreference classifiers on all eligible pairs of entities is $O(nDq)$. The number of produced typed weights is $O(nD|T|)$. Consequently, as we observed in the proof of Theorem 10, the sorting and merging process takes $O(nD|T|\log(nD|T|))$ time. Hence, the time complexity follows. □

4.6 Experiments

In this section, we evaluate the proposed decoding algorithms experimentally. For the evaluation, we use coreference-annotated data prepared as part of the of the Automatic Content Extraction

Algorithm 7 The Greedy Decoding Algorithm for Coreference Resolution and Type Classification

$(e_1, e_2, \ldots, e_n)$ is the list of entities ordered by their location in the document
c is the true coreference classifier
Cl is the entity type classifier
T is the set of entity types
D is the maximum distance between an anaphor and an antecedent
$I = \{\{1\}, \{2\}, \ldots, \{n\}\}$
for all $(e_i, e_j),\ \{i\} \in I, \{j\} \in I$ **do**
  if $|i - j| \leq D$ **then**
    for all $t \in T$ **do**
      if $isCandidateAntecedent(e_i, e_j)$ **then**
        $w_{\{i\},\{j\}}(t) = p_C l(t|con_i) p_C l(t|con_j) c_h(e_i(t), e_j(t))$
      end if
    end for
  else
    for all $t \in T$ **do**
      $w_{\{i\},\{j\}}(t) = 0$
    end for
  end if
end for
Sort $\{w_{\mathbf{i},\mathbf{j}}(t)\}$
$w_{max} = w_{\mathbf{i}^*,\mathbf{j}^*}(t^*) = \max_{\mathbf{i}\in I,\mathbf{j}\in I\setminus\mathbf{i},t\in T} w_{\mathbf{i},\mathbf{j}}(t)$
while $|I| > 1$ *and* $w_{max} > 0$ **do**
  if $\mathbf{i}^* = \{i^*\}$ **then**
    $type(e_{i^*}) = t^*$
  end if
  if $\mathbf{j}^* = \{j^*\}$ **then**
    $type(e_{j^*}) = t^*$
  end if
  $I = I \setminus \{\mathbf{i}, \mathbf{j}\}$
  for all $\mathbf{k} \in I$ **do**
    $w_{\mathbf{k},\mathbf{i}^*\cup\mathbf{j}^*}(t^*) = w_{\mathbf{k},\mathbf{i}^*}(t^*) + w_{\mathbf{k},\mathbf{j}^*}(t^*)$
    for all $t \in T \setminus \{t^*\}$ **do**
      $w_{\mathbf{k},\mathbf{i}^*\cup\mathbf{j}^*}(t) = -\infty$
    end for
  end for
  $I = I \cup \{\mathbf{i}^* \cup \mathbf{j}^*\}$
  $w_{max} = w_{\mathbf{i}^*,\mathbf{j}^*}(t^*) = \max_{\mathbf{i}\in I,\mathbf{j}\in I\setminus\mathbf{i},t\in T} w_{\mathbf{i},\mathbf{j}}(t)$
end while

(ACE) program [10].

4.6.1 Automatic Content Extraction (ACE) Program

The goal of the ACE program is to develop automatic content extraction technology to support processing of language data [10]. We will focus on one task of the ACE program: detection of entities.

The entities to be detected as part of the ACE program are categorized in the following types.

- *Person.* *Person* entities are restricted to denote humans. *Person* entities also include fictional characters (`Santa Claus`) and groups of people, unless the groups satisfy the criteria of *organization* entities.

- *Organization.* *Organizations* are restricted to denote groups of people exhibiting a formal associational structure. They include business units, government agencies, sports teams, music bands, etc.

- *GPE* (Geopolitical Entity). *GPE*s denote politically defined geographic regions. GPEs do not distinguish between a geographic territory, the territory government, or its people. For example, consider the GPE uses of France in the following sentences. `France enjoy a temperate climate` (the geographic territory). `France signed a treaty with the United States` (the government). `France elected a new president` (the people).

- *Facility.* *F*acilities denote permanent man-made structures such as buildings, factories, stadiums, prisons, museums, and space stations, barns, parking garages and airplane hangars, streets, highways, airports, ports, train stations, bridges, and tunnels.

- *Locations.* *Locations* are restricted to geographic entities lacking political connotations, such as land masses, bodies of water, and geological formations. These include, for example, the solar system, Mars, the continents, the Mideast, the Hudson River, Mt. Everest, and Death Valley.

We note that ACE uses a slightly different terminology in describing particular entities (text fragments) as "entity mentions", and using the word "entities" to denote the actual equivalence

classes of "entity mentions" corresponding to real-world entities.

The equivalence classes of entities are also classified whether they are named, nominal, or pronominal. An entity equivalence class is termed named, if it includes a named entity. If an equivalence class includes only pronouns, then it is called pronominal. Otherwise, the equivalence class is nominal. This classification is used during the ACE evaluation process, which we will consider next.

4.6.2 ACE Evaluation Methodology

The goal of the ACE entity detection task is to detect entities in text and compile them into equivalence classes. The entity detection performance is measured on the level of equivalence classes. There are two types of entity detection errors: *misses* and *false alarms*. A *miss* happens when an evaluated system fails to output a true equivalence class. A *false alarm* happens, when a spurious equivalence class is output.

In order to determine miss and false alarm errors, the output equivalence classes must be associated with (mapped to) the true equivalence classes. The mapping is accomplished by mapping an output equivalence to a true equivalence that has the maximum overlap, in terms of entities, with the output equivalence class.[3] The additional constraints are imposed on the mapping that require that each true equivalence class be associated with at most one output equivalence class, and each output equivalence class be associated with at most one true equivalence class. Additionally, for two equivalence classes to be associated they both have to have the same entity type. A *miss* error occurs when a true equivalence class cannot be mapped to an output equivalence class, and a *false alarm* error occurs when an output equivalence class cannot be mapped to a true equivalence class.

The ACE entity evaluation function uses the miss and false alarm statistics of a system to compute an entity evaluation measure as follows. The miss and false alarms statistics are categorized by the entity type and the entity level (named, nominal, pronominal). Let $N_{miss}(type, level)$, $N_{fa}(type, level)$ denote the number of missed and false alarm equivalence classes, respectively, of a fixed $type$ and a fixed $level$. Let $N(type, level)$ be the total number of true equivalence classes for a $type$ and a $level$. Let $C_{miss}(type, level)$ and $C_{fa}(type, level)$ be the costs of missing and incorrectly

[3] See [10] for a detailed description of the mapping algorithm

$C_{miss,fa}(type, level)$	Name	Nominal	Pronominal
Person	1	0.2	0.04
Organization	0.5	0.1	0.02
GPE	0.25	0.05	0.01
Location	0.1	0.02	0.004
Facility	0.05	0.01	0.002

Table 4.1: Miss and False Alarm Cost

	`Training Data`	`Testing Data`
# newswire articles	130	29
# newspaper articles	76	17
# broadcast news articles	216	51

Table 4.2: Number of training and testing documents.

predicting an entity for a fixed *type* and a fixed *level*. The cost functions used in ACE evaluation are presented in Table 4.1. The ACE entity evaluation measure V_{EDT} (called *ACE value*) is computed as

$$V_{EDT} = 1 - \frac{\sum_{type} \sum_{level} (N_{miss}(type, level) C_{miss}(type, level) + N_{fa}(type, level) C_{fa}(type, level))}{\sum_{type} \sum_{level} C_{miss}(type, level) N(type, level)}$$

The ACE entity value of 1 represents a perfect system, while the value of 0 represents a system that outputs nothing. Note that it is quite possible for the value to be negative, if the system outputs too many spurious entities.

4.6.3 Training and Testing Data

The training data for our experiments comprised a collection of newswire, newspaper, and broadcast news articles from the first 6 months of 1998. The testing data were selected from the same sources and covered the last 3 months of 1998. The parameters of the training and testing set are shown in Table 4.2. Each of the documents in the training and testing data was annotated with both type, level , and coreference information.

We used the data to conduct a number of experiments evaluating the quality of learning and decoding algorithms for type classification. The experiments are presented in the following sections.

4.6.4 Learning Algorithms

We evaluated two learning algorithms in our experiments: Support Vector Machine and Logistic Regression. Both algorithms learn linear classifier in feature spaces.

We used the standard configuration of the SVM^{light} [58] implementation in our experiments, with the regularization parameter $C = 1$.

We have implemented logistic regression via an application of the conjugate gradient optimization procedure [80] to minimizing the logistic loss function on the training data. The sparse structure of examples makes the implementation fairly efficient and comparable, in terms of the running time, to the SVM^{light} implementation.

4.6.5 Entity Classification Evaluation

We first measure the performance of entity type classifiers. We formalized the entity type classification problem in Section 4.3. Recall that we generate an example for every name, noun phrase, and pronoun output by the shallow parser. The example is labeled with the true entity type, if any, of the corresponding chunk output by the shallow parser.

The examples features describe the properties of the chunk context. The context covers the chunk under consideration (*current*), the preceding chunk (*previous*), the following chunk (*next*), and the parent chunk (*parent*). The parent chunk is a noun phrase, for candidate entities within a noun phrase, and a sentence, otherwise. For each of the chunks, the shallow parser also produces a set of attributes. The attributes are the text of a chunk (*text*), the head of a chunk (the stemmed head of noun or verb phrase), the part of speech tag of a chunk (*pos*), and a generic entity type of a chunk (*type*) (a `person/organization/location` type assigned by the shallow parser to mostly name chunks). We will use the notation *chunk.attribute* to refer to an attribute of a chunk (e.g., *current.text*). The entity type examples include the following features (where X is a variable instantiated with the value of the corresponding attribute).

- $current.text = X$, $parent.text = X$, $previous.text = X$, $next.text = X$.
- For noun and verb phrase chunks: $current.head = X$, $parent.head = X$, $previous.head = X$, $next.head = X$.

	Training Data			Testing Data		
	Named	Nominal	Pronominal	Name	Nominal	Pronominal
# `Person` examples	5288	5548	5996	1195	1362	1476
# `Organization` examples	2997	1743	809	604	444	309
# `GPE` examples	5006	1178	458	989	280	99
# `Location` examples	264	301	14	44	62	2
# `Facility` examples	226	477	19	33	105	7
# Total examples	14451	10170	8837	3182	2604	2314

Table 4.3: Entity type classification examples

- $current.pos = X$, $parent.pos = X$, $previous.pos = X$, $next.pos = X$.
- For chunks assigned generic entity types: $current.type = X$, $parent.type = X$, $previous.type = X$, $next.type = X$.

We separated the entity type classification problem into three learning subproblems for classifying named, nominal, and pronominal entities. We trained three distinct classifier for the three subproblems.

We used the training and testing documents to generate training and testing examples for entity type classification. The examples statistics are presented in Table 4.3.

We use the standard classification evaluation methodology that was proposed originally for information retrieval [106]. For each entity type, we quantify its classification performance in terms of *F-measure* [106].

We say that an example is positive, for an entity type, if it has been labeled with the entity type. Otherwise, the example is negative. *Precision* is the ratio of the number of correctly classified positive examples to the number of predicted positive examples. *Recall* is the ratio of the number of correctly predicted positive examples to the number of true positive examples. *F-measure* (Fm) combines precision and recall as follows:

$$Fm = \frac{2 * precision * recall}{(precision + recall)}$$

We also micro-averaged F-measure over all types, where the averaging weights are proportional to the number of examples labeled with a corresponding type.

	Name		Nominal		Pronominal		All	
	SVM	LR	SVM	LR	SVM	LR	SVM	LR
`Person`	91.6	91.4	87.4	87.1	89	89.1	89.2	89.1
`Organization`	85.2	86	71.3	72.6	30	29.4	68.1	68.7
`GPE`	94.1	93.9	78.2	77.9	17	16.8	85.2	85
`Location`	55.7	54.3	69.5	70.2	0	0	62.6	62.5
`Facility`	51.2	49.3	65.6	64.5	0	0	59.2	57.9
Total	90.1	90.1	81.5	81.6	75.2	75.2	83.1	83.1

Table 4.4: Entity type classification performance

The combined entity type performance is shown in Table 4.4.[4] The results that indicate that both SVM and logistic regression exhibit the same accuracy in entity type classification.

4.6.6 Coreference Resolution Evaluation

We described in Section 4.4 the example generation and inference algorithms for coreference resolution. In the description, we implied that there is a single coreference classifier that, given an anaphor and a candidate antecedent, predicts whether the anaphor and the antecedent are coreferent.

In the experiments, we relax the assumption of the single classifier. Instead, we split the coreference resolution classifier into several distinct classifiers depending on an anaphor. The split is a result of the fact that different features are appropriate for different kinds of an anaphor. For example, while the distance between an anaphor and and antecedent (in terms of words, sentences, paragraphs) might be extremely useful for pronominal anaphors, it is not a valuable feature for name coreference resolution. Also, different coreference usage patterns may be prevalent for different kinds of anaphor. For instance, we may expect that the pronouns "I" and "it" behave differently with respect to coreference phenomena. Hence, different models may be appropriate for different kinds of pronominal anaphors.

The Table 4.5 shows the categorization of anaphors and their corresponding coreference classifiers. Note that the coreference resolution and example generation algorithms presented above stay the same, with the exception that multiple distinct datasets will be generated during the example generation process based on different kinds of anaphors, and different classifiers will be learned and invoked in the decoding algorithms depending on the kinds of anaphors.

[4] We report the performance in percentage points.

Anaphor	Coreference Classifier
Name	c_{name}
Nominal	$c_{nominal}$
Pronominal: `it`,`its`	c_{it}
Pronominal: `you`, `your`	c_{you}
Pronominal: `they`, `their`, `them`	c_{they}
Pronominal: `we`, `I`, `my`, `us`,`our`	c_{first}
Pronominal: `he`, `she`, `his`, `him`, `her`	c_{third}

Table 4.5: Coreference classifiers

Anaphor/Antecedent	Attributes Used
Name	*text* (the name text) *gender* (the name gender, if any) *last name* (the last name of the name, if any) *first name* (the first name of the name, if any)
Nominal	*head* (the noun phrase head) *gender* (the noun phrase gender, if any) *plural* (whether the noun phrase is plural or singular)
Pronominal	*text* (the pronoun text) *gender* (the pronoun gender, if any) *plural* (whether the pronoun is plural or singular) *personal pronoun type* (first/second/third person pronoun) *possessive* (whether the pronoun is possessive or not)

Table 4.6: Attributes used in coreference resolution

The Table 4.6 lists attributes employed to generate coreference example features for all types of anaphors. Below we list features for each of the coreference classifiers defined in terms of the attributes of the anaphors and antecedents.

The c_{name} classifier employs the following features.

- Every conjunction $A = X \wedge B = Y$, where A is an anaphor attribute, B is an antecedent attribute, and X and Y are the corresponding values of the attributes in the given anaphor and the antecedent.

- For every common attribute A of an anaphor and an antecedent, the value of the proposition, $anaphor.A = antecedent.A$ that reflects the same attributes have the same value in both the given anaphor and the antecedent.

The $c_{nominal}$ classifier and all pronominal classifiers employed the following features.

- Every conjunction $A = X \wedge B = Y$, where A is an anaphor attribute, B is an antecedent attribute, and X and Y are the corresponding values of the attributes in the given anaphor and the antecedent.
- For every common attribute A of an anaphor and an antecedent, the value of the proposition, $anaphor.A = antecedent.A$ that reflects that the same attributes have the same value in both the given anaphor and the antecedent.
- The number of words between the anaphor and the antecedent.
- The number of sentences between the anaphor and the antecedent.
- The number of paragraph between the anaphor and the antecedent.
- The number of candidate antecedent between the anaphor and the antecedents.

Note that the last four distance features are either positive or negative depending whether the antecedent is before or after the anaphor. We also discretized all of the distance features, that is, we converted them into boolean features, where a boolean feature corresponds to a particular discretization bin. We used the entropy-based discretization procedure, with a stopping criterion based on the minimum description length [37].

Coreference Evaluation Results

We adopt the ACE evaluation methodology presented in Section 4.6.2 to evaluate the coreference resolution performance.

We apply both the sequential decoding algorithm and the greedy decoding algorithm in conjunction the Support Vector Machine and Logistic Regression learning algorithms. In both algorithms, we set the parameter $D = 10$ that quantifies the maximum number of candidate antecedents between an anaphor and an antecedent.

Note that the loss functions employed by both SVM and logistic regression are not semi-separable. In order to make the greedy decoding algorithm applicable, we have approximated the loss functions with the following semi-separable loss function:

$$l(y, c_h(x)) = \max(0, -yc_h(x))$$

	Sequential Decoding		Greedy Decoding	
	SVM	LR	SVM	LR
Person	70.8	71.0	70.8	71.5
Organization	53.9	54.5	55.7	56.0
GPE	85.0	85.8	85.4	85.7
Location	17.9	12.7	16.6	12.7
Facility	36.0	32.6	36.3	34.2
Total	67.9	68.3	68.4	69.0

Table 4.7: ACE Values for Different Decoding Algorithms

where $c_h = w \cdot x$ is the score of a linear coreference classifier c.

The Table 4.7 presents the ACE values for the coreference decoding algorithms combined with the corresponding type classification algorithms. It is worth noting that human-level performance for the task is circa 80%[32]. Therefore, the best configuration of the system achieves more than 85% of the human-level performance.

We note that logistic regression exhibited overall better coreference performance than SVM, and that greedy coreference decoding algorithm provides a slight leverage compared to the sequential decoding algorithm at the expense of extra computational complexity.

4.6.7 Evaluation of Combining Type Classification with Coreference Resolution

We next determine incorporation of type classification in the coreference resolution process does improve the extraction performance.

In order to incorporate the type classifier scores into the coreference resolution process, we first need to convert them into conditional probabilities, which we accomplish via the logistic function:

$$p(t|x) = \frac{1}{1 + e^{-Cl_t(x)}} \tag{4.17}$$

For logistic regression, the formula (4.17) represents the conditional probability of the type t given the example x (see Section 2.10). For SVM, the formula is only a heuristic approximation of the conditional probability.

The ACE values in the combined setting are shown in Table 4.8. The performance results that

	SVM	LR
Person	70.9	72.5
Organization	55.9	55.4
GPE	85.8	84.0
Location	13.6	20.5
Facility	35.7	30.4
Total	68.6	69.3

Table 4.8: ACE Values for Entity Classification and Coreference Combination

indicate that the combined setting leads to an improvement in extraction accuracy.

4.6.8 Discussion

The experimental results indicate that, somewhat surprisingly, logistic regression is slightly superior to SVM in application to coreference resolution. Comparison of decoding algorithms shows that the greedy decoding algorithm that exploits redundancy of multiple coreference decisions does provide an advantage over the sequential transitive decoding algorithm. The sequential decoding procedure was however 3-4 times than faster the the greedy algorithm, in our experiments. Finally, combination of coreference resolution with entity type classification leads to correction of entity type mistakes, which translates into better extraction performance.

In general, our approach to entity extraction and coreference resolution is part of the paradigm of *inference with classifiers* [98]. The paradigm stipulates that complex learning problems be divided into modular classification learning sub-problems. The best-of-the-breed learning algorithms are then employed for solving the self-contained learning problems. Yet an application of the learned classifiers in tandem requires an appropriate inference procedure than exploit classifiers' inter-dependencies. We showcased several instances of such inference procedures for entity extraction and coreference resolution. We believe that the inference with classifiers approach is a viable direction in addressing large-scale learning problems involving multiple inter-dependent classification decisions, and its further investigation is warranted.

Chapter 5

Relation Extraction

Relation Extraction is the problem determining relations of interest that hold between extracted entities. For example, an extracted person-organization pair of entities may belong to an *affiliation* relation, specifying that a person is a member of an organization. Two people, on the other hand, may be involved in such social relations as *friend*, *relative*, *associate*, and others.

Relation extraction is an important step towards semantic interpretation of underlying text. Indeed, the set of entities present in text and relations that connect them provides a powerful vocabulary for text understanding.

5.1 Related Work

The problem of relation extraction from natural language texts was previously addressed by Message Understanding Conferences (MUC) [5]. A number of systems were developed that relied on parsing and manual pattern development for identifying the relations of interest (see, for example, [14]). An adaptive system [79], presented under the aegis of MUC, used lexicalized probabilistic context-free grammars augmented with semantic information to produce a semantic parse of text for detecting `organization-location` relations.

Recently, Hidden Markov Models have been used for extracting relations from semi-structured records ("paper title", "author", and "affiliation" extraction from article headers) [43]. HMMs are mostly appropriate for modeling *local* and *flat* problems. Relation extraction from natural language often involves modeling long range dependencies, for which HMM methodology is not directly applicable.

5.2 Problem Formalization

Recall that the entity extraction system is built on top of a generic shallow parsing system. Therefore, after entity processing we obtain a shallow parse augmented with entity type information. We use the shallow parse as input to relation extraction system. We do not utilize coreference information in relation extraction.

Let us consider the sentence, "John Smith is the chief scientist of WorldCom Corp.". The shallow parsing system produces the representation of the sentence shown in Figure 5.1.

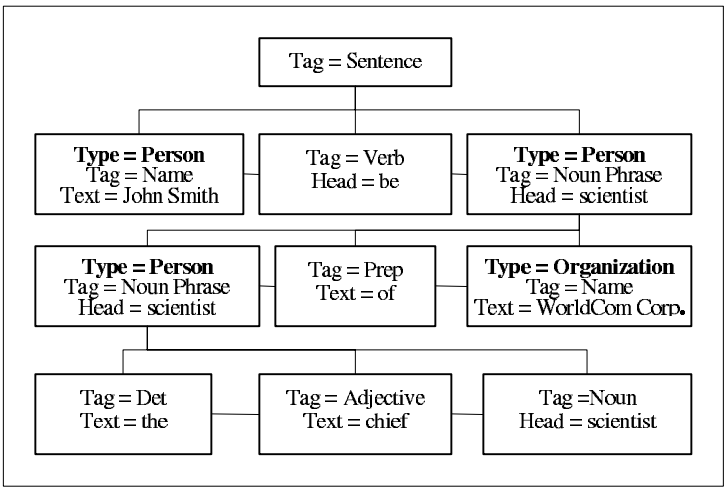

Figure 5.1: The shallow parse representation of the sentence "John Smith is the chief scientist of WorldCom Corp.". The tags are assigned by the generic shallow parsing system, while the (entity) types are assigned by the entity extraction system. The tags "Det" and "Prep" denote "Determiner" and "Preposition", respectively.

The sentence is represented a shallow parse tree. In contrast to common parse trees, the type of a parent node does *not* determine the structure of its children nodes. Instead of providing the full interpretation of the sentence, shallow parsing only identifies its key elements. Therefore, shallow parsing is fairly robust, and is able to generate structured representations even for ungrammatical sentences.

We next convert the shallow parse tree into examples for the `person-affiliation` relation. This type of relation holds between a `person` and an `organization`. There are three nodes in the shallow parse tree in Figure 5.1 referring to people.[1] There is one `organization` node in the

[1]Note that since coreference information is not available, we no not know whether the nodes refer to the same person.

tree. We create an example for the `person-affiliation` relation by taking a `person` node and an `organization` node in the shallow parse tree and assigning attributes to the nodes specifying the role that a node plays in the `person-affiliation` relation. The person and organization under consideration will receive the *member* and *affiliation* roles, respectively. The rest of the nodes will receive *none* roles reflecting that they do not participate in the relation. We then attach a label to the example by asking the question whether the node with the role of *member* and the node with the role of *affiliation* are indeed (semantically) affiliated, according to the sentence. For the above sentence, we will then generate three positive examples, shown in Figure 5.2.

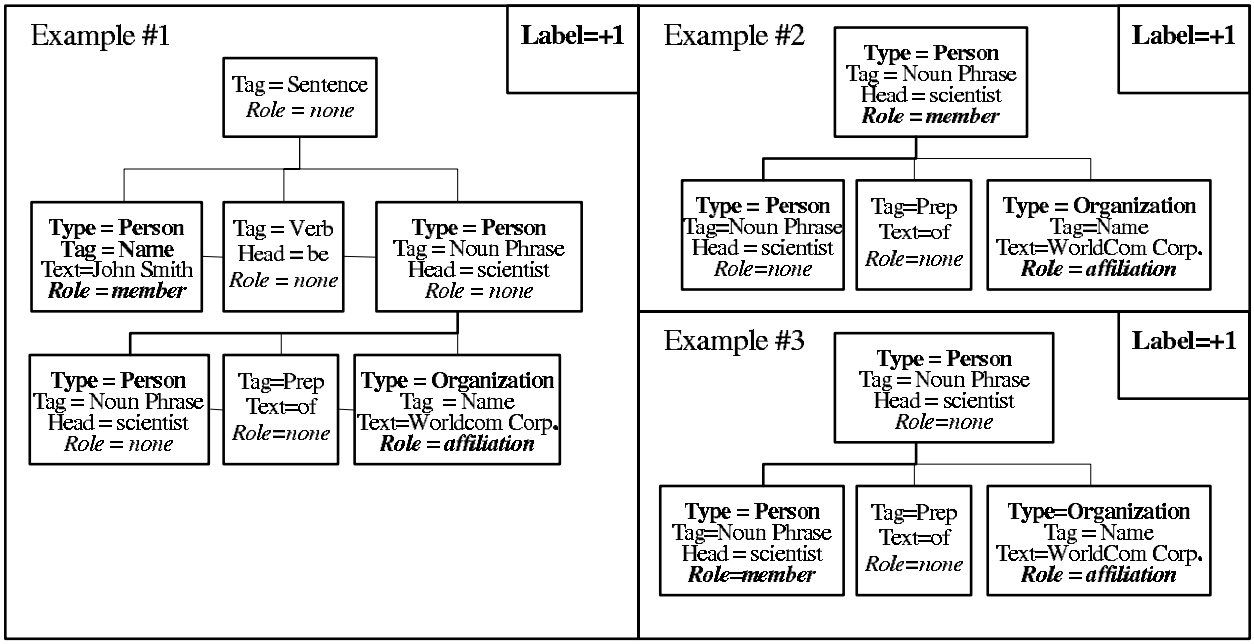

Figure 5.2: The three `person-affiliation` examples generated from the shallow parse in Figure 5.1. The "Label=+1" means that the examples do express the relation.

Note that in generating the examples between the `person` and `organization` entities, we eliminated the nodes that did not belong to the least common subtree enclosing the entities, thereby removing irrelevant subtrees.

To summarize, a relation example is shallow parse, in which nodes are augmented with the role attribute, and each node of the shallow parse belongs to the least common subtree comprising the relation entities under consideration.

We now formalize the notion of relation example. We first define the notion of the example node.

Definition 6 *A node p is a set of attributes $\{a_1, a_2, \ldots\}$. Each node may have a different number of attributes. The attributes are named.*

We use $p.a$ to denote the value of attribute with the name a in the node p, e.g., $p.Type = Person$ and $p.Role = member$. If a node P does not have an attribute a, we say that $P.a = \Lambda$.

Definition 7 *An (unlabeled) relation example is defined inductively as follows:*

- *Let p be a node, then the pair $P = (p, [])$ is a relation example, where by $[]$ we denote an empty sequence.*
- *Let p be a node, and $[P_1, P_2, \ldots, P_l]$ be a sequence of relation examples. Then, the pair $P = (p, [P_1, P_2, \ldots, P_l])$ is a relation example.*

We say that p is the parent of $P_1, P_2, \ldots, P_l$, and P_i's are the children of p. We denote by $P.p$ the first element of the example pair, by $P.c$ the second element of the example pair, and use the shorthand $P.a$ to refer to $P.p.a$, and $P[i]$ to denote P_i. If unambiguous, we also use $P.a_i$ to denote the child P_i of P such that $P_i.Text = a_i$ ($P_i.Head = a_i$, for noun and verb phrases).

A labeled relation example is unlabeled relation example augmented with a label $l \in \{-1, +1\}$. An example is positive, if $l = +1$, and negative, otherwise.

We now define kernels on relation examples that represent similarity of two shallow parse trees.

5.3 Kernels for Relation Extraction

Kernels on parse trees were previously defined by [31]. The kernels enumerated (implicitly) all subtrees of two parse trees, and used the number of common subtrees, weighted appropriately, as the measure of similarity between two parse trees. Since we are operating with shallow parse trees, and the focus of our problem is relation extraction rather than parsing, we use a different definition of kernels.

The nodes of the shallow parse trees have attributes, and we need to use the attributes in the kernel definition. We define a primitive kernel function on the nodes in terms of nodes' attributes, and then extend it on relation examples.

We first define a matching function $t(\cdot,\cdot) \in \{0,1\}$ and a similarity function $k(\cdot,\cdot)$ on nodes. The matching function defined on nodes determines whether the nodes are matchable or not. Many matching functions are possible.

In the case of relation extraction, we can define nodes as matchable only if their types and roles match. If nodes do not have types, then their tags should match as well. Thus,

$$t(P_1.p, P_2.p) = \begin{cases} 1, & \text{if } P_1.Type = P_2.Type \neq \Lambda \text{ and } P_1.Role = P_2.Role \\ 1, & \text{if } P_1.Type = P_2.Type = \Lambda \text{ and } P_1.Tag = P_2.Tag \\ 0, & \text{otherwise} \end{cases}$$

The similarity function on nodes is computed in terms of the nodes' attributes.

$$k(P_1.p, P_2.p) = \begin{cases} 1, & \text{if } P_1.Head = P_2.Head \text{ or } P_1.Text = P_2.Text \\ 0, & \text{otherwise} \end{cases}$$

Then, for two relation examples P_1, P_2, we define the similarity function $K(P_1, P_2)$ in terms of similarity function of the parent nodes and the similarity function K_c of the children. Formally,

$$K(P_1, P_2) = \begin{cases} 0, \text{if } t(P_1.p, P_2, p) = 0 \\ k(P_1.p, P_2.p) + K_c(P_1.c, P_2.c), \quad \text{otherwise} \end{cases} \tag{5.1}$$

Different definitions of the similarity function K_c on children give rise to different K's. We now give a general definition of K_c in terms of similarities of children subsequences. We first introduce some helpful notation (similar to [75]).

We denote by $\mathbf{i}$ a sequence $i_1 \leq i_2 \leq \ldots \leq i_n$ of indices, and we say that $i \in \mathbf{i}$, if i is one of the sequence indices. We also use $d(\mathbf{i})$ for $i_n - i_1 + 1$, and $l(\mathbf{i})$ for length of the sequence $\mathbf{i}$. For a relation example P, we denote by $P[\mathbf{i}]$ the sequence of children $[P[i_1], \ldots, P[i_n]]$.

For a similarity function K, we use $K(P_1[\mathbf{i}], P_2[\mathbf{j}])$ to denote $\sum_{s=1,\ldots,l(\mathbf{i})} K(P_1[i_s], P_2[j_s])$. Then, we define the similarity function K_c as follows

$$K_c(P_1.c, P_2.c) = \sum_{\mathbf{i},\mathbf{j}, l(\mathbf{i})=l(\mathbf{j})} \lambda^{d(\mathbf{i})} \lambda^{d(\mathbf{j})} K(P_1[\mathbf{i}], P_2[\mathbf{j}]) \prod_{s=1,\ldots,l(\mathbf{i})} t(P_1[i_s].p, P_2[j_s].p) \tag{5.2}$$

The formula (5.2) enumerates all subsequences of relation example children with matching parents, accumulates the similarity for each subsequence by adding the corresponding child examples' similarities, and decreases the similarity by the factor of $\lambda^{d(\mathbf{i})}\lambda^{d(\mathbf{i})}$, $0 < \lambda < 1$, reflecting how spread out the subsequences within children sequences. Finally, the similarity of two children sequences is the sum all matching subsequences similarities.

The following theorem states that the formulas (5.1) and (5.2) define a kernel, under mild assumptions.

Theorem 14 *Let $k(\cdot,\cdot)$ and $t(\cdot,\cdot)$ be kernels over nodes. Then, K as defined by (5.1) and (5.2) is a kernel over relation examples.*

To prove the theorem, we need the following lemmas.

Lemma 2 ([53]) *Let K be a kernel on a set $U \times U$ and for all finite non-empty $A, B \subseteq U$ define $\bar{K}(A,B) = \sum_{x\in A, y\in B} K(x,y)$. Then $\bar{K}$ is the kernel on the product of the set of all finite, nonempty subsets of U with itself.*

Lemma 3 ([35]) *If K_1 is a kernel over a set X, and K_2 is a kernel over a set Y, then $K_1 \bigoplus K_2((x,x'),(y,y')) = K(x,x') + K(y,y')$ is a kernel over a set $X \times Y$. The kernel $K_1 \bigoplus K_2$ is called the* direct sum *of kernels K_1 and K_2.*

Corollary 2 *If $K_1, \ldots, K_n$ are kernels over the corresponding sets $X_1, \ldots, X_n$, and then the direct sum $K_1 \bigoplus \cdots \bigoplus K_n((x_1,x_1'),\ldots,(x_n,x_n')) = \sum_{i=1,\ldots,n} K(x_i,x_i')$ is a kernel over the set $X_i \times \cdots \times X_n$.*

Lemma 4 ([35]) *If K_1 is a kernel over a set X, and K_2 is a kernel over a set Y, then $K_1 \bigotimes K_2((x,x'),(y,y')) = K(x,x')K(y,y')$ is a kernel over a set $X \times Y$. The kernel $K_1 \bigotimes K_2$ is called the* tensor product *of kernels K_1 and K_2.*

Corollary 3 *If $K_1, \ldots, K_n$ are kernels over the corresponding sets $X_1, \ldots, X_n$, and then the tensor product $K_1 \bigotimes \cdots \bigotimes K_n((x_1,x_1'),\ldots,(x_n,x_n')) = \prod_{i=1,\ldots,n} K(x_i,x_i')$ is a kernel over the set $X_i \times \cdots \times X_n$.*

Proof of Theorem 14: For two relation examples P_1 and P_2, of which at least one has no children, $K(P_1, P_2) = k(P_1.p, P_2.p)t(P_1.p, P_2.p)$. Therefore, K is a kernel as a product of two kernels[35].

For two relation examples P_1 and P_2 with non-empty children lists, we first extend each children subsequence to be of length $M = max(l(P_1.c), l(P_2.c))$ by appending to it a sequence of "empty" children, thereby embedding the space of all subsequences in the space of subsequences of length M. We also extend the $t(\cdot,\cdot)$ and $k(\cdot,\cdot)$ to empty nodes by making empty nodes match only with empty nodes, and putting $k(x, y) = 0$, if x or y is empty. We also let $d(\mathbf{i})$ denote $i_n - i_1 + 1$, where i_n is the last "non-empty" index of the sequence $\mathbf{i}$.

We then observe that $K_c(P_1.c, P_2.c)$ can be written as

$$K_c(P_1.c, P_2.c) = \sum_{\mathbf{i},\mathbf{j}} \lambda^{d(\mathbf{i})}\lambda^{d(\mathbf{j})} K(P_1[\mathbf{i}], P_2[\mathbf{j}]) \prod_{s=1,\ldots,M} t(P_1[i_s].p, P_2[j_s].p)$$

$K(P_1[\mathbf{i}], P_2[\mathbf{j}])$ is a direct sum of kernels defined over individual children, hence, it is a kernel over subsequences children by Corollary 2. Similarly, $\prod_{s=1,\ldots,l(\mathbf{i})} t(P_1[i_s].p, P_2[j_s].p)$ is a tensor product of kernels, hence, it is a kernel over subsequences of children by Corollary 3. Since the set of kernels is closed with respect to product and scalar multiplication,
$\lambda^{d(\mathbf{i})}\lambda^{d(\mathbf{j})} K(P_1[\mathbf{i}], P_2[\mathbf{j}]) \prod_{s=1,\ldots,M} t(P_1[i_s].p, P_2[j_s].p)$ is a kernel over subsequences of children. Application of Lemma 2 to this kernel, where U is the set of subsequences of children entails that K_c is a kernel over two children sequences represented as sets of their subsequences.

Finally, since a sum and a product of kernels is also a kernel,

$$K(P_1, P_2) = t(P_1.p, P_2.p)k(P_1.p, P_2.p) + t(P_1.p, P_2.p)K_c(P_1.c, P_2.c)$$

is a kernel over relation examples. □

We first consider a special case of K_c, where the subsequences $\mathbf{i}$ and $\mathbf{j}$ are assumed to be *contiguous* and give a very efficient algorithm for computing K_c. In Section 5.3.2, we address a more general case, when the subsequences are allowed to be sparse (non-contiguous).

5.3.1 Contiguous Subtree Kernels

For contiguous subtree kernels, the similarity function K_c enumerates only contiguous children subsequences, that is, for a subsequence $\mathbf{i}$ in (5.2), $i_{s+1} = i_s + 1$ and $d(\mathbf{i}) = l(\mathbf{i})$. Since then $d(\mathbf{i}) = d(\mathbf{j})$ as well, we slightly abuse notation in this section by making λ stand for λ^2 in formula (5.2). Hence, (5.2) becomes

$$K_c(P_1.c, P_2.c) = \sum_{\mathbf{i},\mathbf{j}, l(\mathbf{i})=l(\mathbf{j})} \lambda^{l(\mathbf{i})} K(P_1[\mathbf{i}], P_2[\mathbf{j}]) \prod_{s=1,\ldots,l(\mathbf{i})} t(P_1[i_s].p, P_2[j_s].p) \tag{5.3}$$

Let us consider a relation example corresponding to the sentence "James Brown was a scientist at the University of Illinois". The example is shown in Figure 5.3. We compare the example with the relation example #1 in Figure 5.2.

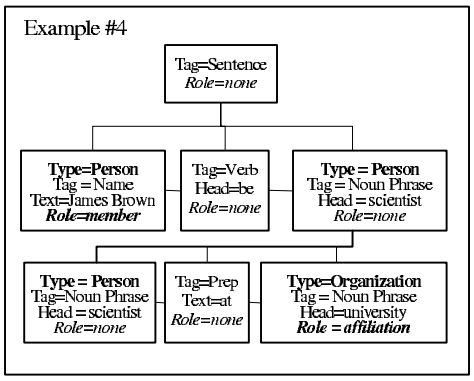

Figure 5.3: A relation example for the sentence "James Brown was a scientist at the University of Illinois".

According to the definitions (5.1) and (5.3), for the examples P_1 (relation example #1) and P_2 (relation example #4), the kernel function is computed as follows (Assume that the matching and similarity functions are those defined in Section 5.3. Also assume that $\lambda = 0.5$).

$$\begin{aligned}
K(P_1,P_2) &= k(P_1.p,\ P_2.p) \\
&\quad + K_c([P_1.JohnSmith,\ P_1.be,\ P_1.scientist],[P_2.JamesBrown,\ P_2.be,\ P_2.scientist]) \\
&= 0.5(k(P_1.JohnSmith,\ P_2.JamesBrown)+k(P_1.be,\ P_2.be)+K(P_1.scientist,\ P_2.scientist)) \\
&\quad + 0.5^2(k(P_1.JohnSmith,\ P_2.JamesBrown)+2k(P_1.be,\ P_2.be)+K(P_1.scientist,\ P_2.scientist)) \\
&\quad + 0.5^3(k(P_1.JohnSmith,\ P_2.JamesBrown)+k(P_1.be,\ P_2.be)+K(P_1.scientist,\ P_2.scientist))
\end{aligned}$$

$$= \quad 1.125+0.875K(P_1.scientist,\ P_2.scientist)$$

$$= \quad \ldots$$

$$= \quad 1.125+0.875 \cdot 1.875$$

$$= \quad 2.765625$$

The core of the kernel computation resides in the formula (5.3). The formula enumerates all contiguous subsequences of two children sequences. We now give a fast algorithm for computing K_c between P_1 and P_2, which, given kernel values for children, runs in time $O(mn)$, where m and n is the number of children of P_1 and P_2, respectively.

Let $C(i,j)$ be the K_c computed for suffixes of children sequences of P_1 and P_2, where every subsequence starts with indices i and j, respectively. That is,

$$C(i,j) = \sum_{\mathbf{i},\mathbf{j},i_1=i,j_1=j,l(\mathbf{i})=l(\mathbf{j})} \lambda^{l(\mathbf{i})} K(P_1[\mathbf{i}], P_2[\mathbf{j}]) \prod_{s=1,\ldots,l(\mathbf{i})} t(P_1[i_s].p, P_2[j_s].p)$$

Let $L(i,j)$ be the length of the longest sequence matching states in the children of P_1 and P_2 starting with indices i and j, respectively. Formally,

$$L(i,j) = \max\{l : \prod_{s=0,\ldots,l} t(P_1[i+s].p, P_2[j+s].p) = 1\}$$

Then, the following recurrences hold:

$$L(i,j) = \begin{cases} 0, \text{ if } t(P_1[i].p, P_2[j], p) = 0 \\ L(i+1, j+1) + 1, \text{ otherwise} \end{cases} \tag{5.4}$$

$$C(i,j) = \begin{cases} 0, \text{if } t(P_1[i].p, P_2[j], p) = 0 \\ \frac{\lambda(1-\lambda^{L(i,j)})}{1-\lambda} K(P_1[i], P_2[j]) + \lambda C(i+1, j+1), \text{ otherwise} \end{cases} \tag{5.5}$$

The boundary conditions are:

$$L(m+1, n+1) = 0$$

$$C(m+1, n+1) = 0$$

The recurrence (5.5) follows from the observation that, if $P_1[i]$ and $P_2[j]$ match, then every matching pair (c_1, c_2) of sequences that participated in computation of $C(i+1, j+1)$ will be extended to the matching pair $([P_1[i], c_1], [P_2[j], c_2])$, and

$$\begin{aligned} C(i,j) &= \lambda K(P_1[i], P_2[j]) + \sum_{(c_1,c_2)} \lambda^{l(c_1)+1}(K(P_1[i], P_2[j]) + K(c_1, c_2)) \\ &= \sum_{s=1,\ldots,L(i,j)} \lambda^s K(P_1[i], P_2[j]) + \lambda \sum_{(c_1,c_2)} \lambda^{l(c_1)} K(c_1, c_2)) \\ &= \frac{\lambda(1-\lambda^{L(i,j)})}{1-\lambda} K(P_1[i], P_2[j]) + \lambda C(i+1, j+1) \end{aligned}$$

Now we can easily compute $K_c(P_1.c, P_2.c)$ from $C(i,j)$.

$$K_c(P_1.c, P_2.c) = \sum_{i,j} C(i,j) \tag{5.6}$$

The time and space complexity of K_c computation is $O(mn)$, given kernel values for children. Hence, for two relation examples the complexity of computing $K(P_1, P_2)$ is the sum of computing K_c for the matching internal nodes (assuming that complexity of $t(\cdot,\cdot)$ and $k(\cdot,\cdot)$ is constant).

5.3.2 Sparse Subtree Kernels

For sparse subtree kernels, we use the general definition of similarity between children sequences as expressed by (5.2).

Let us consider a example corresponding to the sentence "John White, a well-known scientist at the University of Illinois, led the discussion.". The example is shown in Figure 5.4. We compare the example with the relation example #1 in Figure 5.2.

According to the definitions (5.1) and (5.3), for the examples P_1 (relation example #1) and P_2 (relation example #5), the kernel function is computed as follows (Assume that the matching and similarity functions are those defined in Section 5.3. Also assume that $\lambda = 0.5$).

$$\begin{aligned} K(P_1,P_2) &= k(P_1.p,\ P_2.p) + \\ &\quad + K_c([P_1.JohnSmith,\ P_1.be,\ P_1.scientist],[P_2.JohnWhite,\ P_2.comma,\ P_2.scientist,\ P_2.lead,\ P_2.discussion]) \end{aligned}$$

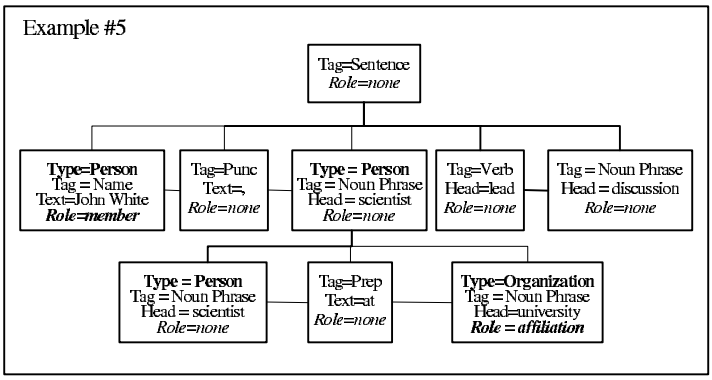

Figure 5.4: A relation example for the sentence "James Brown, a well-known scientist at the University of Illinois, led the discussion.". The tag "Punc" denotes "Punctuation".

$$
\begin{aligned}
= \;& 0.5^2(k(P_1.JohnSmith,\ P_2.JohnWhite)+k(P_1.be,\ P_2.lead)+K(P_1.scientist,\ P_2.scientist)) \\
& +0.5^2 0.5^4 k([P_1.JohnSmith,\ P_1.be],\ [P_2.JohnWhite,\ P_2.lead])+ \\
& +0.5^3 0.5^3([P_1.JohnSmith,\ P_1.scientist],\ [P_2.JohnWhite,\ P_2.scientist])+ \\
& +(0.5^2+0.5^6)K(P_1.scientist,\ P_2.scientist) \\
= \;& 0.265625 \cdot K(P_1.scientist,\ P_2.scientist) \\
= \;& \ldots \\
= \;& 0.265625 \cdot 2.078125 \\
= \;& 0.552
\end{aligned}
$$

As in the previous section, we give an efficient algorithm for computing K_c between P_1 and P_2. The algorithm runs in time $O(mn^3)$, given kernel values for children, where m and n ($m \geq n$) is the number of children of P_1 and P_2, respectively.

Let $K_{c,q}(i,j)$ be K_c computed using subsequences of length q in prefixes of children sequences of P_1 and P_2 ending with indices i and j.

$$K_{c,q}(i,j) = \sum_{\mathbf{i} \subseteq \{1,\ldots,i\}} \sum_{\substack{\mathbf{j} \subseteq \{1,\ldots,j\} \\ l(\mathbf{i})=l(\mathbf{j})=q}} \lambda^{d(\mathbf{i})} \lambda^{d(\mathbf{j})} K(P_1[\mathbf{i}], P_2[\mathbf{j}]) T(\mathbf{i},\mathbf{j})$$

where

$$T(\mathbf{i},\mathbf{j}) = \prod_{s=1,\ldots,l(\mathbf{i})} t(P_1[i_s].p, P_2[j_s].p)$$

Let $C_q(i,j,a)$ be the K_c computed using subsequences of length q in prefixes of children sequences of P_1 and P_2 ending with indices i and j, respectively, with a number $a \in R$ added to each result of kernel children computation, and setting a damping factor for a sequence $\mathbf{i(j)}$ to be $\lambda^{i-i_1+1}(\lambda^{i-i_1+1})$. Formally,

$$C_q(i,j,a) = \sum_{\mathbf{i} \subseteq \{1,\ldots,i\}} \sum_{\substack{\mathbf{j} \subseteq \{1,\ldots,j\} \\ l(\mathbf{i})=l(\mathbf{j})=q}} [\lambda^{i-i_1+j-j_1+2}(K(P_1[\mathbf{i}], P_2[\mathbf{j}]) + a)T(\mathbf{i},\mathbf{j})]$$

Then the following recurrences hold:

$$\begin{aligned}
C_0(i,j,a) &= a \\
C_q(i,j,a) &= \lambda C_q(i,j-1,a) + \\
&\quad \sum_{s=1,\ldots,i} [t(P_1[s].p, P_2[j].p)\lambda^{i-s+2} \cdot C_{q-1}(s-1, j-1, a + K(P_1[s], P_2[j]))] \\
K_{c,q}(i,j) &= \lambda K_{c,q}(i,j-1) + \\
&\quad \sum_{s=1,\ldots,i} [t(P_1[s].p, P_2[j].p)\lambda^2 \cdot C_{q-1}(s-1, j-1, K(P_1[s], P_2[j]))] \\
K_c &= \sum_{q=1,\ldots,\min(m,n)} K_{c,q}(m,n)
\end{aligned}$$

The above recurrences do not allow, however, for an efficient algorithm in computing K_c due to presence of *real-valued* parameter a.

In order to obtain an efficient dynamic programming algorithm, we rewrite $C_q(i,j,a)$ as follows:

$$C_q(i,j,a) = aC_q(i,j) + \sum_{r=1,\ldots,q} C_{q,r}(i,j)$$

where

$$C_q(i,j) = \sum_{\mathbf{i} \subseteq \{1,\ldots,i\}} \sum_{\substack{\mathbf{j} \subseteq \{1,\ldots,j\} \\ l(\mathbf{i})=l(\mathbf{j})=q}} \lambda^{d(\mathbf{i})}\lambda^{d(\mathbf{j})}T(\mathbf{i},\mathbf{j})$$

and

$$C_{q,r}(i,j) = \begin{cases} \sum_{\substack{i_1=1,\ldots,i \\ j_1=1,\ldots,j}} \left[t(P_1[i_1].p, P_2[j_1].p)\lambda^{i-i_1+j-j_1+2} C_{q-1,r}(i_1-1, j_1-1)\right], \text{ if } q \neq r \\ \sum_{\substack{i_1=1,\ldots,i \\ j_1=1,\ldots,j}} \left[t(P_1[i_1].p, P_2[j_1].p)\lambda^{i-i_1+j-j_1+2} K(P_1[i_1], P_2[j_1]) C_{q-1}(i_1-1, j_1-1)\right], \text{ if } q = r \end{cases}$$

Observe that $C_q(i,j)$ computes the subsequence kernel of [75] (with matching nodes) for prefixes of P_1 and P_2. Hence, we can use the result of [75] to give $O(qij)$ for $C_q(i,j)$ computation. Denote

$$C'_q(i,j) = \sum_{s=1,\ldots,i} t(P_1[s].p, P_2[j].p)\lambda^{i-s+2} C_{q-1}(s-1, j-1)$$

Then

$$C_q(i,j) = \lambda C_q(i, j-1) + C'_q(i,j)$$

and

$$C'_q(i,j) = t(P_1[i], P_2[j])\lambda^2 C_{q-1}(i-1, j-1) + \lambda C'_q(i, j-1)$$

Using the same trick for $C_{q,r}(i,j)$, we get

$$C_{q,r}(i,j) = \lambda C_{q,r}(i, j-1) + C'_{q,r}(i,j)$$

where

$$C'_{q,r}(i,j) = \begin{cases} \lambda C'_{q,r}(i, j-1) + t(P_1[i], P_2[j])\lambda^2 C_{q-1,r}(i-1, j-1), \text{ if } q \neq r \\ \lambda C'_{q,r}(i, j-1) + t(P_1[i], P_2[j])\lambda^2 K(P_1[i], P_2[j]) C_{q-1}(i-1, j-1), \text{ o.w.} \end{cases}$$

That completes our list of recurrences of computing $K_q(i,j,a)$. The boundary conditions are

$$\begin{aligned} K_{c,q}(i,j) &= 0, \text{ if } q > \min(i,j) \\ C_q(i,j) &= 0, \text{ if } q > \min(i,j) \\ C_0(i,j) &= 1, \\ C'_q(i,j) &= 0, \text{ if } q > \min(i,j) \end{aligned}$$

$$C_{q,r}(i,j) = 0, \text{ if } q > \min(i,j) \text{ or } q < r$$
$$C'_{q,r}(i,j) = 0, \text{ if } q > \min(i,j) \text{ or } q < r$$

5.4 Experiments

In this section, we apply kernel methods to extracting two types of relations from text: `person-affiliation` and `organization-location`.

A `person` and an `organization` are part of the `person-affiliation` relation, if the `person` is a member of or employed by `organization`. A company founder, for example, is defined not to be affiliated with the company (unless, it is stated that (s)he also happens to be a company employee).

A `organization` and a `location` are part of the `organization-location` relation, if the `organization`'s headquarters is at the `location`. Hence, if a single division of a company is located in a particular city, the company is not necessarily located in the city.

The nuances in the above relation definitions make the extraction problem more difficult, but they also allow to make fine-grained distinctions between relationships that connect entities in text.

5.4.1 Experimental Methodology

The (text) corpus for our experiments comprises 200 news articles from different news agencies and publications (Associated Press, Wall Street Journal, Washington Post, Los Angeles Times, Philadelphia Inquirer).

We used the existing shallow parsing system to generate the shallow parses for the news articles. We generated relation examples from the shallow parses for both relations, as described in Section 5.2. That is, for the `person-affiliation` relation, a relation example was generated for every (`person`,`organization`) pair of entities that appeared in the same sentence. The relation example was labeled as positive, if the `person` was affiliated the `organization`, and it was labeled as negative otherwise. We again note that no coreference information was used in generating examples. The resulting examples' statistics are shown in Table 5.1.

For each relation, we randomly split the set of examples into a training set (60% of the examples) and a testing set (40% of the examples). We obtained the models by running learning algorithms on

	person-affiliation	org-location
#positive	1262	506
#negative	2262	1409
#total	3524	1915

Table 5.1: Number of examples for relations.

the training set, testing the models on the test set, and computing performance measures. In order to get stable performance estimates, we averaged performance results over 10 random train/test splits. For each of the algorithms, we also computed the learning curves by gradually increasing the number of examples in the training set and observing performance change on the test set. The learning curves were also averaged over 10 random train/test splits.

For extraction problems, the system performance is usually reflected using the performance measures of information retrieval: precision, recall, and F-measure [106]. Precision is the ratio of the number of correctly predicted positive examples to the number predicted positive examples. Recall is the ratio of the number of correctly predicted positive examples to the number of true positive examples. F-measure (Fm) combines precision and recall as follows:

$$Fm = \frac{2 * precision * recall}{(precision + recall)}$$

We report precision, recall, and F-measure for each experiment. We also present F-measure learning curves for each learning curve experiment.

In the experiments below, we present the performance of kernel-based algorithms for relation extraction in conjunction with that of feature-based algorithms. Note that the set of features used by the feature-based learning algorithms (presented in Appendix 6.3) is not the same as the set of implicit features employed by kernel-based learning algorithms. The features used correspond to small subtrees of the shallow parse representations of relation examples, while the kernel formulation can take advantage of subtrees of any size. Therefore, in comparing the performance of kernel-based and feature-based methods, we seek to evaluate how much advantage a kernel formulation can give us with respect to a less expressive feature formulation.

We now describe the experimental setup of the algorithms used in evaluation.

5.4.2 Kernel Methods Configuration

We evaluated two kernel learning algorithms: Support Vector Machine (SVM) [33] and Voted Perceptron [44]. For SVM, we used the SVM^{Light} [56] implementation of the algorithm, with custom kernels incorporated therein. We implemented the Voted Perceptron algorithm as described in [44].

We implemented both contiguous and sparse subtree kernels and incorporated them in the kernel learning algorithms. For both kernels, λ was set to 0.5. The only domain specific information in the two kernels was encapsulated by the matching $t(\cdot,\cdot)$ and similarity $k(\cdot,\cdot)$ functions on nodes, as defined in Section 5.3.

We should emphasize that the above definitions of t and k are the *only* domain-specific information that the kernel methods use. Certainly, the kernel design is somewhat influenced by the problem of relation extraction, but the kernels can be used for other (not necessarily text-related) problems as well, if the functions t and k are defined differently.

We also normalized the computed kernels before their use within the algorithms. The normalization corresponds to the standard unit norm normalization of examples in the feature space corresponding to the kernel space [35]:

$$K(P_1, P_2) = \frac{K(P_1, P_2)}{\sqrt{K(P_1, P_1)K(P_2, P_2)}}$$

For both SVM^{Light} and Voted Perceptron, we used their standard configurations (e.g., we did not optimize the value of C that interpolates the training error and regularization cost for SVM, via cross-validation). For Voted Perceptron, we performed two passes over the training set.

5.4.3 Linear Methods Configuration

We evaluated three feature-based algorithms for learning linear discriminant functions: Naive-Bayes, Winnow, and SVM.

We designed features for the relation extraction problem. The features are conjunctions of conditions defined over relation example nodes. The features are listed in Appendix 6.3.

Again, we use the standard configuration for both algorithms: for Naive Bayes we employed

	Recall	Precision	F-measure
Naive Bayes	75.59	91.88	82.93
Winnow	80.87	88.42	84.46
SVM (feature-based)	76.21	91.67	83.22
Voted Perceptron (contiguous)	79.58	89.74	84.34
SVM (contiguous)	79.78	89.9	84.52
Voted Perceptron (sparse)	81.62	90.05	85.61
SVM (sparse)	82.73	91.32	**86.8**

Table 5.2: `Person-affiliation` performance (in percentage points)

	Recall	Precision	F-measure
Naive Bayes	71.94	90.40	80.04
Winnow	75.14	85.02	79.71
SVM (feature-based)	70.32	88.18	78.17
Voted Perceptron (contiguous)	64.43	92.85	76.02
SVM (contiguous)	71.43	92.03	80.39
Voted Perceptron (sparse)	71	91.9	80.05
SVM (sparse)	76.33	91.78	**83.3**

Table 5.3: `Organization-location` performance (in percentage points)

add-one smoothing [55]; for Winnow, learning rate (promotion parameter) was set to 1.1 and the number of training set passes to 2. For SVM, we used the linear kernel and set the regularization parameter (C) to 1.

5.4.4 Experimental Results

The performance results for `person-affiliation` and `organization-location` are shown in Table 5.2 and Table 5.3, respectively.

The results indicate that kernel methods exhibit good performance in relation extraction. The results should be taken with caution though, for kernel-based and feature-based learning algorithms employ very different representations of examples. Indeed, kernel methods work (implicitly) in far richer feature spaces than those used by feature-based algorithms. It is therefore not surprising that their performance is generally superior. If such rich feature spaces could be explicitly generated and employed within feature-based algorithms, the performance picture could be different.

In order to answer the question whether such rich feature spaces could be generated *efficiently*, we attempted to generate the features corresponding to sparse subtree kernels. The features rep-

resent all possible relation example subtrees, in which subsets of nodes are instantiated with the values of their Text/Head attributes. The generation process led to a combinatorial explosion in the number of features that prevented successful completion of the experiments.[2] Therefore, the *equivalent* feature-based representation of sparse subtree kernels is not feasible, at least with limited computational resources. It is though an open issue whether a less expressive yet more tractable feature representation would be sufficient to bridge the gap between the two approaches.

One practical problem in applying kernel methods to NLP is their speed. Training kernel classifiers, especially with custom kernels, takes an order of magnitude more time compared to the training time for feature classifiers. Therefore, kernel methods are more difficult to tune. Moreover, we found out that tuning of kernel algorithms is a complicated process by itself, for slight modification in the kernel implementation and/or parameters may lead to major changes in performance.

Kernel classifiers are also much slower compared to feature classifiers.[3] Indeed, an application of a kernel classifier requires evaluation of numerous kernels whose computational complexity may be too high for practical purposes. Many low level problems in natural language processing involve very large corpora with tens and hundreds of thousands of examples. Even if kernel classifiers only depend on a small subset of the examples (for instance, support vectors of SVM), the need to evaluate thousands of complex kernels during the classifier application may render kernel methods inappropriate for various practical settings. Therefore, there is a pressing need to develop algorithms that combine the advantages of kernel methods with practical constraints that require efficient application of the classifiers learned.

Sparse vs. Contiguous: Learning Curves

The Figure 5.5 depicts F-measure learning curves for for kernel-based algorithms algorithms with different kernels.

The learning curves indicate that the sparse subtree kernel is far superior to the contiguous subtree kernel. From the enumeration standpoint, the subtree kernels implicitly enumerate the *exponential* number of children subsequences of a given parent, while the contiguous kernels essen-

[2] The desktop computer used for the experiments (1.2GHz Pentium III, 512Mb) ran out of memory.
[3] In our experiments, features classifiers were nearly 10 times faster than kernel classifiers.

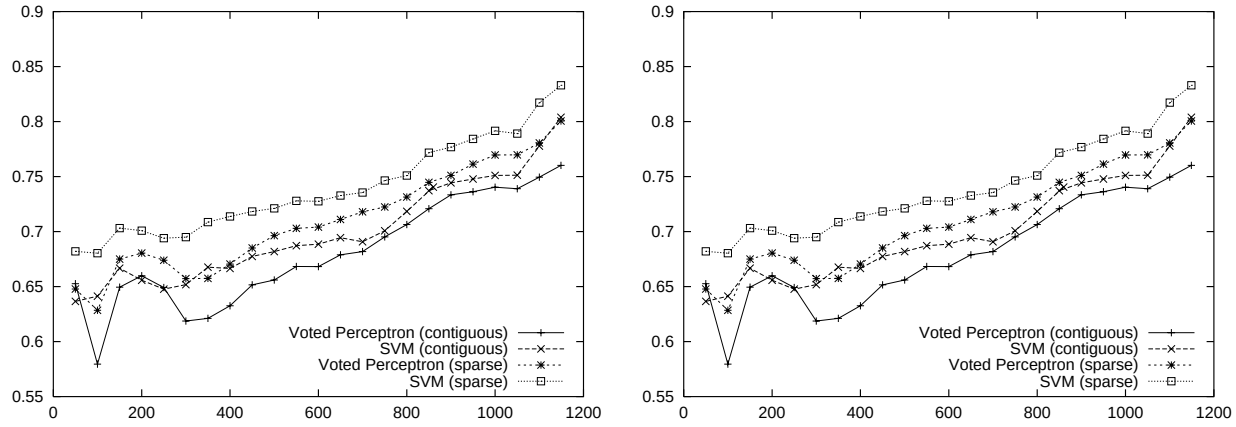

Figure 5.5: Learning curve (of F-measure) for the `person-affiliation` relation (on the left) and `org-location` relation (on the right), comparing kernel-based learning algorithms with different kernels.

tially operate with n-grams, whose number is just quadratic in a children sequence length. This exponential gap between the sparse and contiguous kernels leads to a significant performance improvement. The result is promising, for it showcases that it is possible to (implicitly) consider an exponential number of features while paying just a low polynomial price, with a significant performance boost.

5.5 Discussion

Kernel-based methods are an elegant approach for learning in rich structural domains. Our results show that, for relation extraction, the methods perform very well, while allowing for minimal ingestion of problem knowledge.

Our work follows recent applications of kernel methods to natural language parsing [31] and text categorization [75]. The common theme in all the papers is that objects' structure can be leveraged in a fairly efficient and statistically sound way.

The NLP domain is precisely the domain where the structural descriptions of objects (words, phrases, sentences) can be exploited. While the prevalent n-grams approach to language modeling imposes statistical and computational constraints, kernel-based language modeling may help eschew the constraints. We hypothesize that the methods will require much fewer examples in achieving the state of the art performance for a range of NLP problems than approaches based on probabilistic

modeling.

Design of kernels for structural domains is a very rich research area. An interesting direction to pursue would be to use extensive work on *distances* defined on structural objects [101] in kernel design. The distance-based methods have already found widespread application in bioinformatics [39], and can be fruitfully extended to work in the NLP domain as well. [110] presents sufficient conditions for a Pair Hidden Markov Model (which is a probabilistic version of edit distance) to constitute a kernel. More generally, the work of [48] makes it possible to use any distance measure to embed objects (and define a dot product) in a pseudo-euclidean space. Incidentally, SVM can be adapted for the pseudo-euclidean representations [50, 90], hence, applicable in structural domains, where natural distance functions exist.

5.6 Relation Extraction and Coreference Resolution

In order for the developed algorithms for relation extraction to be deployed within an information extraction system, the decisions output by relation classifiers have to be combined with other information. In particular, coreference information is crucial for determining whether two extracted relations correspond to the same relation between two real word entities. For example, let the relation `person-affiliation`("He","WorldCom") be extracted from the sentence "He works for WorldCom", and the relation `person-affiliation`("John Smith","the company") be extracted from the sentence "John Smith is the chief scientist of the company". If there is coreference information coreferring "He" with "John Smith" and coreferring "the company" with "WorldCom", then the two extracted relations can be *merged* thereby producing a more informative relation between the two names: `person-affiliation`("John Smith","WorldCom").

In general, exploring the interplay between coreference resolution and relation extraction is a viable direction for further research in information extraction.

Chapter 6

Putting It All Together: a Trainable Information Extraction System

In the preceding chapters, we presented a number of algorithms for part of speech tagging, entity extraction, coreference resolution, and relation extraction. This chapter describes how the algorithms and components fit together in the design of a complete information extraction system.

6.1 Training Information Extraction System

The training process of the information extraction (IE) system is shown in Figure 6.1. The process assumes availability of documents with the following labeling information necessary for different extraction subtasks: part of speech tagging, entity information, coreference information, and relation information.

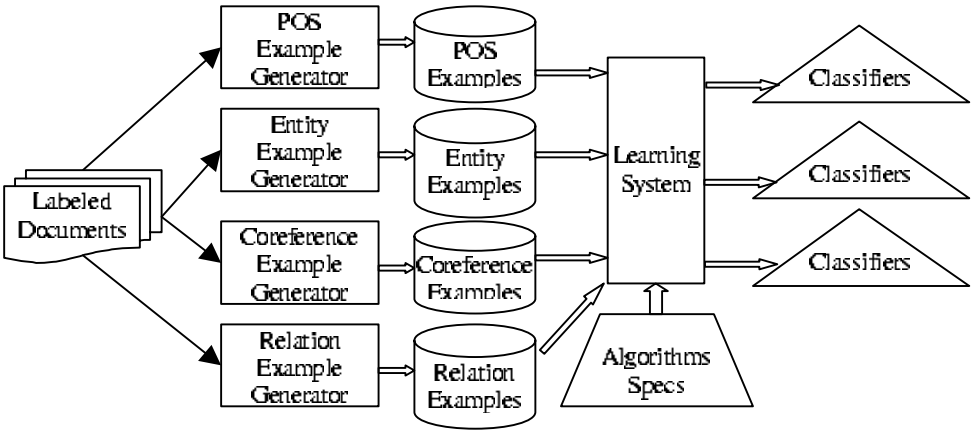

Figure 6.1: Training Information Extraction System

The labeled documents are employed within the example generation modules to generate examples for extraction subtasks. For each of the extraction subtasks, the relevant example generator focuses on the corresponding portion of labeled documents and uses the document and label information to compute relevant features and compile them into examples, as we described in the preceding chapters. Thus, the output of the example generation modules is a collection of labeled datasets, where each dataset contains a set of labeled examples for a particular extraction subtask.

Each of the extraction subtasks corresponds to a particular classifier. We derive the classifiers by applying learning algorithms to the generated datasets. Different learning algorithms may be applied to different datasets, thereby leading to different types of classifiers used for different extraction subtasks. Indeed, the algorithm specification is defined declaratively outside of the system. The declaration allows for easy manipulation of algorithm parameters and greatly facilitates learning experiments.

The learned classifiers are saved externally to be loaded during the extraction system application.

6.2 Applying Information Extraction System

The information extraction process is shown in Figure 6.2.

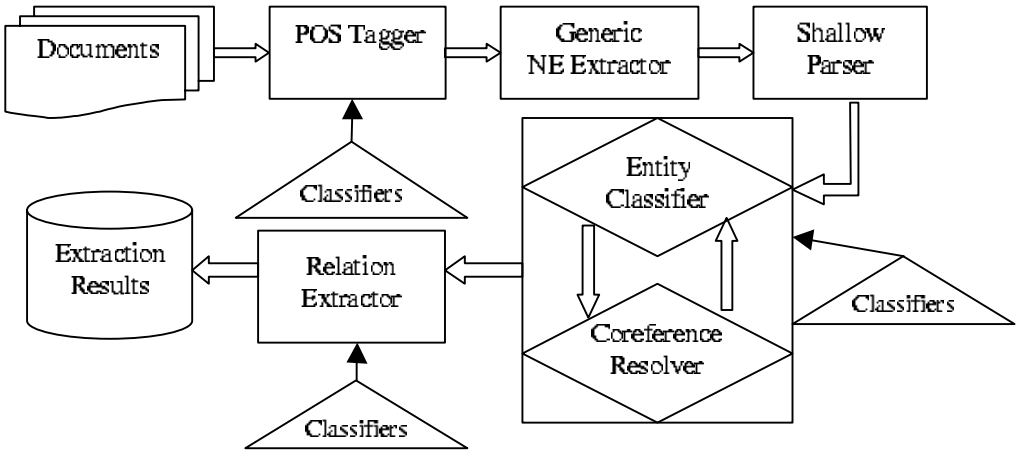

Figure 6.2: Applying Information Extraction System

The process comprises a sequence of tasks. Output of one task is considered as input for the following task, with the exception of entity classification and coreference resolution that can be

conducted simultaneously. The learned classifiers are loaded from outside the extraction system and used to perform the required classification tasks within the corresponding modules.

The architecture of the system allows for classifier transparency. That is, new classifiers can be easily "plugged-in" in the extraction system at run time.

The extraction results consists of entities, their equivalence classes, and relations that have been discovered by the system. The results are represented in the XML form and can be stored in the database, if necessary.

6.3 Final Remarks

We presented a variety of machine learning techniques and explained how they can be effectively applied to the problem of information extraction.

We experimentally evaluated the SNOW system for part of speech tagging, developed novel decoding algorithms for entity classification and coreference resolution, and introduced the classification methodology and designed novel kernels for relation extraction. In our work, we were guided by the prevailing philosophy that such a complex and multi-task problem as information extraction can be reasonably segmented into a number of simple learning subproblems. Different properties of the learning problems give rise to different learning techniques being appropriate for their solution. It was our goal to select and extend the suitable learning techniques for their subsequent application to the extraction subproblems.

For complex learning tasks, the process of applying learned classifiers leads to interesting *inference* problems, when the classifiers are to be used in conjunction with constraints governing their application. We studied an example of such constraints in coreference resolution and entity classification. We foresee further advances in developing inference mechanisms that involve a broader range of extraction subproblems, classifiers, and constraints.

In general, natural language processing represents an extremely fertile area for advancing the state of the art of machine learning research. In particular, applications of machine learning to information extraction require tackling new challenges in dealing with sparse data, noise, and complex dependencies. Furthermore, classifier application leads to complex inference problems that are only beginning to be addressed in both learning theory and practice.

We believe that our work will be a valuable contribution to understanding the phenomenon of learning in natural language, and the ability of artificial systems to reproduce the phenomenon.

References

[1] *Proceedings of the 3rd Message Undertanding Conference (MUC-3)*, San Francisco, US, 1991. Morgan Kaufmann Publishers.

[2] *Proceedings of the 4th Message Undertanding Conference (MUC-4)*, San Francisco, US, 1992. Morgan Kaufmann Publishers.

[3] *Proceedings of the 5th Message Undertanding Conference (MUC-5)*, San Franciscco, US, 1993. Morgan Kaufmann Publishers.

[4] *Proceedings of the 6th Message Undertanding Conference (MUC-6)*, San Francisco, US, 1995. Morgan Kaufmann Publishers.

[5] *Proceedings of the 6th Message Undertanding Conference (MUC-7)*, 1998.

[6] Computational Natural Language Learning, 2000. http://cnts.uia.ac.be/conll2000/.

[7] Journal of Machine Learning Research, Special Issue on Shallow Parsing, 2002.

[8] S. Abney. Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny, editors, *Principle-based parsing*. 1990.

[9] S. Abney. Partial parsing via finite-state cascades. In *Proceedings of ESSLLI Robust Parsing Workshop*, 1996.

[10] Automatic Content Extraction. http://www.nist.gov/speech/tests/ace/index.htm.

[11] S. Ait-Mokhtar and J. Chanod. Incremental finite-state parsing. In *Proceedings the 5th Conference on Applied Natural Language Processing*, 1997.

[12] E. Amaldi and V. Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147(1–2), August 1995.

[13] C. Aone and S. W. Bennett. Applying machine learning to anaphora resolution. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, volume 1040 of *Lecture Nodes in Artificial Intelligence*. Springer Verlag, Berlin, 1996.

[14] C. Aone, L. Halverson, T. Hampton, and M. Ramos-Santacruz. Sra: Description of the ie2 system used for muc-7. In *Proceedings of MUC-7*, 1998.

[15] C. Aone and M. Ramos-Santacruz. Rees: A large-scale relation and event extraction system. In *Proceedings of the 6th Applied Natural Language Processing Conference*, 2000.

[16] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[17] P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT Press, Cambridge, US, 1999.

[18] L. E. Baum. An Inequality and Associated Maximization in Statistical Estimation for Probabilistic Functions of Markov Processes. *Inequalities*, 3, 1972.

[19] A. Berger, S. Della Pietra, and V. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1), 1996.

[20] D. Bikel, R. Schwartz, and R. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3), 1999.

[21] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of ACM*, 36(4), 1989.

[22] E. Boros and A. Prekopa. Probabilistic bounds and algorithms for the maximum satisfiability problem. *Annals of Operations Research*, 21, 1989.

[23] A. Borthwick. *Maximum Entropy Approach to Named Entity Recognition.* PhD thesis, New York University, 1999.

[24] E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 1995.

[25] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the 17th National Conference on Artificial Intelligence*, Orlando, FL, July 1999.

[26] C. Cardie and K. Wagstaff. Noun phrase coreference as clustering. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[27] E. Charniak. *Statistical Language Learning.* MIT Press, Cambridge, Massachusetts, 1993.

[28] E. Charniak, N. Ge, and J. Hale. A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, 1998.

[29] C. Chekuri, A. Goldberg, D. Karger, M. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[30] P. Clifford. Markov random fields in statistics. In G. Grimmett and D. Welsh, editors, *Disorder in Physical Systems. A Volume in Honour of John M. Hammersley*, Oxford, 1990. Clarendon Press.

[31] M. Collins and N. Duffy. Convolution kernels for natural language. In *Proceedings of NIPS-2001*, 2001.

[32] Linguistic Data Consortium. Personal communication, 2002.

[33] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3), 1995.

[34] T. Cover. Geometrical and statistical properties of systems and linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 19, 1965.

[35] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines (and Other Kernel-Based Learning Methods).* Cambridge University Press, 2000.

[36] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society, B*, 39, 1977.

[37] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proc. 12th International Conference on Machine Learning*. Morgan Kaufmann, 1995.

[38] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley, New York, 1973.

[39] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Seqience Analysis.* Cambridge University Press, 1998.

[40] Y. Even-Zohar. *Multi-Class Classification in Natural Language Processing.* PhD thesis, Department of Computer Science, University of Illinois, 2001.

[41] R. A. Fisher. *Statistical Methods and Scientific Inference.* Oliver-and-Boyd, 1956.

[42] G. Forney. The Viterbi algorithm. *Procedings of IEEE*, 61, 1973.

[43] D. Freitag and A. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the 7th Conference on Artificial Intelligence*, Menlo Park, CA, 2000. AAAI Press.

[44] Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3), 1999.

[45] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[46] A. Garg, S. Har-Peled, and D. Roth. On generalization bounds, projection profile, and margin distribution. In *Proceedings of the 19th International Conference on Machine Learning*, 2002.

[47] A. Garg and D. Roth. Learning coherent concepts. In *Proceedings of the 12th International Conference on Algorithmic Learning Theory*. Springer, 2001.

[48] L. Goldfarb. A new approach to pattern recognition. In *Progress in pattern recognition 2*. North-Holland, 1985.

[49] A. R. Golding and D. Roth. A Winnow-based approach to context-sensitive spelling correction. *Machine Learning*, 34, 1999.

[50] T. Graepel, R. Herbrich, and K. Obermayer. Classification on pairwise proximity data. In *Advances in Neural Information Processing Systems 11*, 1999.

[51] J. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. Unpublished manuscript.

[52] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[53] D. Haussler. Convolution kernels on discrete structures, 1999.

[54] K. Humphreys, R. Gaizauskas, S. Azzam, C. Huyck, B. Mitchell, H. Cunningham, and Y. Wilks. University of Sheffield: Description of the LaSIE-II system as used for MUC-7. In *Proceedings of MUC-7* [5].

[55] F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, Massachusetts, 1997.

[56] T. Joachims. Text categorization with support vector machines: learning with many relevant features. *European Conf. Mach. Learning, ECML98*, April 1998.

[57] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, chapter 11. MIT Press, Cambridge, US, 1999.

[58] Thorsten Joachims. *Learning Text Classifiers with Support Vector Machines*. Kluwer Academic Publishers, Dordrecht, NL, 2002.

[59] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3), March 1987.

[60] M. Kearns, R. Schapire, and L. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2/3), 1994.

[61] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts, 1994.

[62] C. Kennedy and B. Boguraev. Anaphora for everyone: pronominal anaphora resolution without a parser. In *Proceedings of the 16International Conference on Computational Linguistics*, 1996.

[63] R. Khardon, D. Roth, and L. Valiant. Relational learning for NLP using linear threshold elements. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, S.F., 1999. Morgan Kaufmann Publishers.

[64] J. Kivinen, M. K. Warmuth, and P. Auer. The perception algorithm versus winnow: Linear versus logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, 97(1–2), 1997.

[65] R. Kneser and H. Ney. Improved backing-off for M-gram language modeling. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Detroit, MI, 1995.

[66] Y. Krymolowski and D. Roth. Incorporating knowledge in natural language learning: A case study. In S. Harabagiu, editor, *Use of WordNet in Natural Language Processing Systems: Proceedings of the Conference*. Association for Computational Linguistics, Somerset, New Jersey, 1998.

[67] T. Kudo and Y. Matsumoto. Chunking with support vector machines. In *Proceedings of the 2nd meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

[68] J. Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer Speech and Language*, 6(3), July 1992.

[69] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1–2), 2000.

[70] D. Landau, R. Feldman, O. Zamir, and Y. Aumann. TextVis: An integrated visual environment for text mining. *Lecture Notes in Computer Science*, 1510, 1998.

[71] S. Lappin and H. Leass. An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4), 1994.

[72] S. Lauritzen. *Graphical Models.* Clarendon Press, Oxford, 1996.

[73] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2, 1987.

[74] N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms.* PhD thesis, Technical Report UCSC-CRL-89-11, University of California Santa Cruz, 1989.

[75] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, February 2002.

[76] D. Luenberger. *Linear and Nonlinear Optimization.* Addison-Wesley, 1984.

[77] J. McCarthy and W. Lehnert. Using decision trees for coreference resolution. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, 1995.

[78] A. Mikheev, C. Grover, and M. Moens. Description of the LTG system as used for MUC-7. In *Proceedings of MUC-7* [5].

[79] S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz, R. Stone, and R. Weischedel. Algorithms that learn to extract information - BBN: Description of the SIFT system as used for MUC-7. In *Proceedings of MUC-7*, 1998.

[80] T. Minka. Algorithms for maximum-likelihood logistic regression. Technical report, Department of Statistics, Carnegie-Mellon University, 2001.

[81] R. Mitkov. *Anaphora Resolution*. Longman, 2002.

[82] M. Munoz, V. Punyakanok, D. Roth, and D. Zimak. A learning approach to shallow parsing. Technical Report 2087, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1999.

[83] K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: An empirical study. In K. Laskey and H. Prade, editors, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, S.F., Cal., 1999. Morgan Kaufmann Publishers.

[84] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In O. Etzioni, J. Müller, and J. Bradshaw, editors, *Proceedings of the 3rd International Conference on Autonomous Agents*, Seattle, WA, USA, 1999. ACM Press.

[85] V. Ng and C. Cardie. Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

[86] W. Soon; D. Lim; H. Ng. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 12 2001.

[87] A. Novikoff. On convergence proofs on perceptrons, 1962.

[88] M. Pasca and S. Harabagiu. High-performance question/answering. In *24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001.

[89] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers, Inc., 1987.

[90] E. Pekalska, P. Paclik, and R. Duin. A generalized kernel approach to dissimilarity-based classification. *Journal of Machine Learning Research*, 2, 2001.

[91] L. Ramshaw and M. Marcus. Exploring the the nature of transformation-based learning. In J. Klavans and P. Resnik, editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*. MIT Press, Cambridge, Massachusetts, 1996.

[92] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In E. Brill and K. Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Somerset, New Jersey, 1996.

[93] A. Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34, 1999.

[94] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan Books, Washington D.C., 1962.

[95] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2), 1996.

[96] D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of the 15th National Conference on Artificial Intelligence*, Menlo Park, 1998. AAAI Press.

[97] D. Roth. Learning in natural language. In D. Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, S.F., 1999. Morgan Kaufmann Publishers.

[98] D. Roth. Reasoning with classifiers. Invited Talk at ECML-2002, 2002.

[99] D. Roth and D. Zelenko. Part of speech tagging using a network of linear separators. In *Proceedings of COLING-ACL-98*, 1998.

[100] D. Roth and D. Zelenko. Toward a theory of learning coherent concepts. In *Proceedings of the 7th Conference on Artificial Intelligence*, Menlo Park, CA, 2000. AAAI Press.

[101] D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits, and Macromolecules*. CSLI Pulications, 1999.

[102] H. Schutze and Y. Singer. Part-of-speech tagging using a variable context Markov model. In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, Hillsdale, NJ, 1994. Erlbaum Associates.

[103] C. Sidner. Toward a computational theory of definite anaphora comprehension in English. Technical report, MIT Press, 1979.

[104] F. Damerau T. Zhang and D. Johnson. Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, 2, 2002.

[105] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), November 1984.

[106] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979.

[107] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[108] V. Vapnik. *Statistical Learning Theory*. John Wiley, 1998.

[109] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2), 1971.

[110] C. Watkins. Dynamic alignment kernels. Technical report, Department of Computer Science Royal Holloway, University of London, 1999.

[111] D. Zelenko, C. Aone, and A. Richardella. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3, 2003.

[112] T. Zhang. Statistical behavior and consistency of support vector machines, boosting, and beyond. In *Proceedings of the 19th International Conference on Machine Learning*. Morgan Kaufman, 2002.

Features for Relation Extraction

In the process of feature engineering, we found the concept of node depth (in a relation example) to be very useful. The depth of a node $P_1.p$ (denoted $depth(P_1.p)$) within a relation example P is the depth of P_1 in the tree of P. We also defined a derived attribute $TypeTag$ as follows:

$$P.TypeTag = \begin{cases} P.Type, & \text{if } P.Type \neq \Lambda \\ P.Tag, & \text{if } P.Type = \Lambda \end{cases}$$

The features are itemized below (the lowercase variables $text$, $typeTag$, $role$, and $depth$ are instantiated with specific values for the corresponding attributes).

- For every node $P_1.p$ in a relation example, add the following features:
  - $depth(P_1.p) = depth \wedge P_1.TypeTag = typeTag \wedge P_1.Role = role$
  - $depth(P_1.p) = depth \wedge P_1.Text = text \wedge P_1.Role = role$
  - $depth(P_1.p) = depth \wedge P_1.Tag = tag \wedge P_1.Role = role$
- For every pair of nodes $P_1.p$, $P_2.p$ in a relation example, such that P_1 is the parent of P_2, add the following features:
  - $depth(P_1.p) = depth \wedge P_1.TypeTag = typeTag \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.TypeTag = typeTag \wedge P_2.Role = role \wedge parent$
  - $depth(P_1.p) = depth \wedge P_1.TypeTag = typeTag \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.Text = text \wedge P_2.Role = role \wedge parent$
  - $depth(P_1.p) = depth \wedge P_1.Text = text \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.TypeTag = typeTag \wedge P_2.Role = role \wedge parent$

- $depth(P_1.p) = depth \wedge P_1.TypeTag = text \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.Text = text \wedge P_2.Role = role \wedge parent$

- For every pair of nodes $P_1.p$, $P_2.p$ in a relation example, with the same parent P, such that P_2 follows P_1 in the P's children list, add the following features:

  - $depth(P_1.p) = depth \wedge P_1.TypeTag = typeTag \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.TypeTag = typeTag \wedge P_2.Role = role \wedge sibling$
  - $depth(P_1.p) = depth \wedge P_1.TypeTag = typeTag \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.Text = text \wedge P_2.Role = role \wedge sibling$
  - $depth(P_1.p) = depth \wedge P_1.Text = text \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.TypeTag = typeTag \wedge P_2.Role = role \wedge sibling$
  - $depth(P_1.p) = depth \wedge P_1.TypeTag = text \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.Text = text \wedge P_2.Role = role \wedge sibling$

- For every triple of nodes $P_1.p$, $P_2.p$, $P_3.p$ in a relation example, with the same parent P, such that P_2 follows P_1, and P_3 follows P_2 in the P's children list, add the following features:

  - $depth(P_1.p) = depth \wedge P_1.TypeTag = typeTag \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.TypeTag = typeTag \wedge P_2.Role = role \wedge depth(P_3.p) = depth \wedge P_3.Text = text \wedge P_3.Role = role \wedge siblings$
  - $depth(P_1.p) = depth \wedge P_1.TypeTag = typeTag \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.Text = text \wedge P_2.Role = role \wedge depth(P_3.p) = depth \wedge P_3.TypeTag = typeTag \wedge P_3.Role = role \wedge siblings$
  - $depth(P_1.p) = depth \wedge P_1.Text = text \wedge P_1.Role = role \wedge depth(P_2.p) = depth \wedge P_2.TypeTag = typeTag \wedge P_2.Role = role \wedge depth(P_3.p) = depth \wedge P_3.TypeTag = typeTag \wedge P_3.Role = role \wedge siblings$

Vita

Dmitry Zelenko was born September 14, 1973 in Minsk, Belarus. He graduated from the department of Applied Mathematics of the Belarusian State University in 1996. In 1995, he joined the Computer Science Department at the University of Illinois at Urbana-Champaign, where he received his M.Sc. degree in Computer Science in 1997. A year later he was awarded M.Sc. in Mathematics. In 1999, he took a leave from the University of Illinois to pursue his interests in the industry. He applied machine learning methods to design a system for detecting unusual market activities, which has been successfully deployed and used to monitor the NASDAQ market. In 2002, he resumed his studies at the University of Illinois and proceeded to work on his dissertation, "Machine Learning for Information Extraction", while simultaneously conducting the related research at SRA International. Currently, Dmitry Zelenko continues to research and develop adaptive methods for information extraction at SRA International. He lives in Washington, DC.