Information Extraction from Multiple Syntactic Sources

Shubin Zhao

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May 2004

Ralph Grishman

Dedicated to my mother

Acknowledgements

I would like to thank my advisor Ralph Grishman for his guidance in academics. He is a great researcher with keen interests in science, constant efforts in doing things by hand and great personality. He is the example I followed and will continue to follow. Without him this thesis would not be possible.

This research was supported in part by the Defense Advanced Research Projects Agency as part of the Translingual Information Detection, Extraction and Summarization (TIDES) program, under Grant N66001-001-1-8917 from the Space and Naval Warfare Systems Center, San Diego, and by the National Science Foundation under Grant IIS-0081962 and IIS-00325657. This thesis does not necessarily reflect the position of the U.S. Government.

I would like to thank my committee members for reviewing this thesis and providing valuable suggestions. I also thank other people in the NLP group for the good discussions and inspiring ideas.

I would also thank the staff in the Computer Science Department, Anina Karman and Rosemary Amico whose efforts made my study in the CS department as a Phd student a pleasant process.

I'm grateful to my families for their support and encouragement of my academic pursuit.

Finally I would like to thank my wife Bing. Her support, care, and jokes make my life more colorful and enjoyable.

Abstract

Information Extraction is the automatic extraction of facts from text, which includes detection of named entities, entity relations and events. Conventional approaches to Information Extraction try to find syntactic patterns based on deep processing of text, such as partial or full parsing. The problem these solutions have to face is that as deeper analysis is used, the accuracy of the result decreases, and one cannot recover from the induced errors. On the other hand, lower level processing is more accurate and it can also provide useful information. However, within the framework of conventional approaches, this kind of information can not be efficiently incorporated.

This thesis describes a novel supervised approach based on kernel methods to address these issues. In this approach customized kernels are used to match syntactic structures produced from different preprocessing phases. Using properties of a kernel, individual kernels are combined into a composite kernel to integrate and extend all the information. The composite kernels can be used with various classifiers, such as Nearest Neighbor or Support Vector Machines (SVM). The main classifier we propose to use is SVM due to its ability to generalize in large dimensional feature spaces. We will show that each level of syntactic information can contribute to IE tasks, and low level information can help to recover from errors in deep processing.

The new approach has demonstrated state-of-the-art performance on two benchmark tasks. The first task is detecting slot fillers for management succession events (MUC-6). For this task two types of kernels were designed, a surface kernel based on word n-grams and a kernel built on sentence dependency trees; the second task is the ACE RDR evaluation, which is to recognize relations between entities in text from newswire and broadcast news transcript. For this task, five kernels were built to represent information from sentence tokenization, syntactic parsing and dependency parsing. Experimental results for the two tasks will be shown and discussed.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

1.1 The Problem

Information Extraction (IE), a subarea of Natural Language Processing, extracts facts from text and puts them into structured representations such as templates or databases. IE research has been promoted by U.S. Government-sponsored programs (MUCs and ACE).

The techniques used by Information Extraction depend greatly on the sub-language used in a domain, such as financial news or medical records. The training data for an IE system is often sparse as the target domain changes quickly. Traditional IE approaches[6][22][44][53] try to generate patterns from training data by human observation or by learning with predefined pattern templates. There are also statistical approaches[16][43] which encode event patterns in probabilistic CFG grammars.

Most of these approaches are based on generative models. They assume that events occur in text in certain patterns and these patterns can be discovered and expressed by a metalanguage. However this assumption may not be completely

Processing	Performance (F-measure)
Tokenization	99%
POS tagging	94%
NP chunking	92%
Parsing	88%

Table 1.1: Performance of different levels of text processing

right and it has limitations.

First, this assumption requires deep processing (parsing) of text. Parsing, either shallow or deep, constructs the syntatic structure of text, which contains more regularity than word sequences. Generalization at the syntax level is much more effective, thus extraction patterns are mostly expressed at the syntax level. Lower levels of information such as word sequences are ignored by these approaches, even though they may still contain useful information. Incorporating them would make the patterns too complicated to manipulate.

The general problem in Natural Language Processing is that processing of text is incremental, such that errors in shallow analysis propagate to deeper analysis. Hence when the processing level gets deeper, it becomes less accurate. Table 1.1 shows the state-of-the-art performance of text processing at different levels. This table shows an error rate of about 10% in deep processing. When an approach is based solely on deep representations of text, these errors are irrecoverable.

Generally speaking, systems based on a single level of representation are forced to choose between shallower representations, which will have fewer errors, and deeper representations, which may be more general.

Second, patterns are local. A pattern often needs anchor points (such as words in a regular expression) and matches other items continuously from there. They can only describe syntax in a local context. It is difficult and often ineffective to use patterns for a whole sentence. In pattern matching, the long distance or non-contiguous information outside of the local context is missed. But it could be useful for an IE task.

Statistical models are based on sentence parsing. But they only attempted to model short-range context, such as the relation between two named entities. In long and complicated sentences, this is still local context. It is not clear how long distance information can affect prediction of the local context.

1.2 The Solution

The idea to overcome these problems is to make no prior assumption about the syntactic structure an event may assume. We should consider all syntactic features in text and use a discriminative classifier to make decisions based on them. Discriminative classifiers make no attempt to resolve the structure of the target class. They only care about the decision boundary that separates the objects. In our case, we only need criteria to separate the targets (named entities, relations or event slots) from text using all the syntactic information from preprocessing.

This thesis presents a new framework that uses kernel functions to represent different levels of syntactic structure. With the properties of kernel functions, individual kernels can be combined freely into composite kernels to incorporate information. The main classifier we propose to use is SVM (Support Vector Machine), mostly due to its ability to generalize in high dimensional feature spaces. But other classifiers that work with kernel functions can also be used where appropriate.

In this framework, by combining different levels of syntactic analysis of a sentence, the generalization ability of a classifier can help to overcome errors in deep processing results. Each kernel represents features from the analysis of whole sentences. Using a discriminative classifier like SVM, these features are weighted in the training phase according to their contribution to separating the targets. Therefore, in addition to the local features, long distance information can also come into play.

The new approach has demonstrated state-of-the-art performance on two benchmark tasks: detecting slot fillers for management succession events (MUC-6) and finding relation between entities (ACE RDR). For the MUC-6 task, two types of kernels were designed, including a surface kernel based on word n-grams and a kernel built on sentence dependency trees. The ACE RDR (Relation Detection and Recognition) task is to recognize relations between entities from newswire text and broadcast news transcripts. For this task, five kernels were built to represent information from sentence tokenization, syntactic parsing and dependency parsing.

Experimental results for the two tasks will be provided and discussed. Comparisons with the start-of-the-art approaches will also be included. For the relation detection task, we will also compare performance of different classifiers under the same kernel setups.

1.3 Overview

This section describes the chapter structure of this thesis. Chapter 1 is an introduction of the central issues and the new approach proposed in this thesis. Chapter 2 covers the background of Information Extraction and its basic component technologies, including named entity recognition, entity relation detection and event extraction.

Chapter 3 surveys the classical approaches for Information Extraction, ranging from rule-based approaches and symbolic learning to statistical models.

Chapter 4 covers the basics of kernel methods and Support Vector Machines. It also includes their applications in Information Extraction or Natural Language Processing in general. Since we only focus on their application, the complete machine learning theories behind them are not covered. References are given for deeper theoretical interests.

Chapter 5 provides an analysis of the limitations in prior approaches. Then a new supervised model based on kernel methods is proposed to address these issues. The discriminative model uses kernels to represent information from different levels of syntactic preprocessing and combines them into composite kernels for a classifier. This chapter will show the structure of this model and explain the intuitions behind it.

Chapter 6 briefly describes the GLARF dependency analyzer developed at NYU. It explains the features of GLARF and shows examples of its analysis.

Chapter 7 introduces an application of this approach to slot filler detection for the MUC-6 events. The kernels used for this task will be formulated and explained. There will also be experiment results for different kernel setups and the comparison to a good rule-based system.

Chapter 8 presents another application of this approach applied to the entity relation detection task specified by the ACE RDR project in 2004. We will show performance on different kernel setups and a comparison using different classifiers. Evaluation results on the official RDR test data are also included.

Chapter 9 provides discussion and proposes further directions to explore.

Chapter 2

Background

2.1 Information Extraction

Text has been a major way to store and convey information in human society. With the development of the internet and digital media, a user can have instant access to a huge amount of text. The volume of text available on the web is accumulating at a constantly increasing speed. The world in text is full of information and how to locate the specific information a user needs becomes a critical issue.

A central goal of Natural Language Processing (NLP) is to be able to understand the underlying meaning of texts and translate them into machine-comprehensible representations. Then the computational power of machines would enable us to manipulate the information in more user-friendly ways, such as producing summaries or answering questions.

However, human knowledge about the world is complicated. Even after decades of research, there is still no effective way to represent the full range of real world knowledge. Although it is impossible to obtain a universal representation of knowledge, we can make the problem tractable by confining the domain of the text. Then it is possible to represent the underlying world knowledge or semantics in a simple format like templates. Information Extraction (IE) is the practical way to get one step closer to the goal of NLP. It is domain-dependent.

In a narrow sense, Information Extraction reduces facts in text into a structured representation[23], such as tables or templates. The concept of information extraction was first introduced by Harris[25] in the 1950's. Its first application[48] was reported in the 1980's within the medical domain. Since then it has evolved greatly. Now Information Extraction involves extracting facts of different granularity, ranging from named entities and entity relations to complete events.

Since there are different flavors of Information Extraction, the focus of this thesis will be extraction from free text. Here is an example of input and output for extraction of corporate management succession events:

Input:

> *An extraordinary shareholders meeting of AB Volvo in Gothenburg, Sweden, elected Bert-Olof Svanholm chairman of the Swedish automotive group, in line with an earlier proposal.*
>
> *Mr. Svanholm is president of ABB Asea Brown Boveri Ltd., an engineering concern jointly owned by Asea AB of Sweden and BBC Brown Boveri AG of Switzerland. He succeeds Pehr G. Gyllenhammar, who resigned in December after the collapse of a plan to merge Volvo's vehicle operations with those of French partner Renault SA.*

The expected output is shown in table 2.1.

The output of this example is represented as a table. The fields of the table are predefined items of a management succession event, such as the company

Type	Event 1	Event 2
Company	AB Volvo	ABB Asea Brown Boveri Ltd.
Post	chairman	president
Person_In	Bert-Olof Svanholm	–
Person_Out	Pehr G. Gyllenhammar	Mr. Svanholm
...	...	...

Table 2.1: Table of management succession events.

involved (Company), the management position (Post), the person who assumed the position (Person_In) and the person who left the position (Person_Out). This table is customizable, depending on what kind of information is of interest. For example, someone may also like to know when the succession happened or in which country this company is located. Then extraction of these items will also become part of the IE task.

In the sample text above, there are names that have to be identified correctly, such as "Bert-Olof Svanholm" and "Pehr G. Gyllenhammar" as person names, "AB Volvo" and "ABB Asea Brown Boveri Ltd." as company names. These are called Named Entities. Recognition of them from text is a basic technology for Information Extraction.

Once we have the event information stored in tables, we can build relational databases from them, where different events can be indexed by their common attributes. Then it is possible to answer questions like "Who was named chairman by AB Volvo in 1986?" or even "Who left ABB Asea and joined AB Volvo?". As the growth of information surrounding us accelerates, information extraction becomes an intriguing and fast-developing technology for people. Although it is still far from perfection, it already shows its power to facilitate collecting and accessing information.

The example above shows a tabular representation of the output. There are also other means to represent information, such as templates or XML records.

Most of the information extraction approaches are based on the idea of finding invariant expressions or patterns across documents from annotated text. Generation of patterns can be done through human observation or machine learning techniques. As a domain dependent technology, there are two central issues in Information Extraction:

1) Portability. The domains of interest can be very different, e.g. medical records, sports news and financial reports. Real life applications often require an extraction system for a new domain to be developed in a short period of time. So portability is an important feature for an IE system. A good IE approach should be able to adapt to a new domain without too much human effort.

2) Data Sparseness. Annotation of large amount of data is costly and time-consuming, so training data is often sparse for IE tasks. In a standard MUC (Message Understanding Conference) evaluation, the training data is hundreds news articles and 100k - 200k words. So an IE approach should be able to generalize from limited domain data, in addition to other general purpose data independent of domain.

The difficulty of Information Extraction is that for human languages, there are lots of ways to express the same event. The following are examples of language phenomena given by Ralph Grishman[24] that an IE system has to deal with.

- Name descriptors: *IBM, the famous computer manufacturer, ap-*

pointed Harriet Smith as president.

- Sentence modifiers: *IBM unexpectedly appointed Harriet Smith yesterday as president.*
- Tense: *IBM has appointed / will appoint Harriet Smith as president.*
- Clause structure: *Harriet Smith, who was appointed president by IBM, ...*
- Nominalization: *IBM announced the appointment of Harriet Smith as president.*
- Conjunction: *IBM declared a special dividend and appointed Harriet Smith as president.*
- Anaphoric reference: *IBM has made a major management shuffle; the company appointed Harriet Smith as president this week.*
- Need for inference: *Thomas J. Watson resigned as president of IBM, and Harriet Smith succeeded him.*

2.1.1 Named Entity Recognition

Text usually contains all kinds of names, for example person names, company names, sports teams, chemicals and lots of other names from a specific domain. Other common units can also fall into this category, such as time expressions, numbers or job titles. These names are referred to as named entities (NE) in Information Extraction. Failing to recognize them as a unit would affect the accuracy of deeper analysis of text, such as chunking or parsing. Therefore named entity recognition becomes a basic component technology for Information Extraction or Natural Language Processing in general.

The following shows an annotation of named entities of the sample text in section 2.1. The span and type of each NE are indicated in the SGML annotation.

```
<ENAMEX TYPE="PERSON">Mr. Svanholm<\ENAMEX> is president of
<ENAMEX TYPE="ORG">ABB Asea Brown Boveri Ltd.<\ENAMEX>, an
engineering concern jointly owned by <ENAMEX TYPE ="ORG">
Asea AB<\ENAMEX> of <ENAMEX TYPE="LOC">Sweden<\ENAMEX> and
<ENAMEX TYPE="ORG">BBC Brown Boveri AG<\ENAMEX> of <ENAMEX
TYPE="LOC">Switzerland<\ENAMEX>. He succeeds <ENAMEX
TYPE="PERSON">Pehr G. Gyllenhammar<\ENAMEX>, who resigned
in December after the collapse of a plan to merge <ENAMEX
TYPE="ORG">Volvo<\ENAMEX>'s vehicle operations with those
of <ENAMEX TYPE="LOC">French<\ENAMEX> partner <ENAMEX
TYPE="ORG">Renault SA<\ENAMEX>.
```

The task of named entity recognition is to find the span of a name and determine the type of the entity. For English, capitalization is a good clue to identify the word span of a name. But it is not easy to recognize "A Real New York Bargain" as a company name. In other languages or transcripts of English speech where capitalization information is not available, this becomes a more difficult task.

The types of named entities also depend on the domain and the task. In the management succession domain, "chief executive officer" is considered a job name, while in the disease outbreak domain, "dengue haemorrhagic fever" should be identified as a disease name. There are seven types of common NEs defined by MUC for newswire text. They are person, organization, location, date, time, money and percentage. Sekine et al. (2004)[52] defined about 200 categories of common named entities, organized in a hierarchical structure.

Various approaches have been applied to detect NEs in text, such as Decision Tree[51], Maximum Entropy[10][41] and Hidden Markov Model[9]. The HMM model is simple but effective in capturing the sequential relations between words inside and around a name. Many named entity recognition implementations are based on this model. Recently Support Vector Machines were also applied to this task with good performance reported[7].

2.1.2 Entity Relation Detection

In addition to Named Entities, there are also other mentions of an entity in text, such as nominal or pronominal mentions. In the sample text in section 2.1, there are mentions like "Bert-Olof Svanholm", "Mr. Svanholm" and "He". They all refer to the same person. In this thesis, the term "entity" refers to the real world referent of a mention. So in the example above, all three mentions refer to the same entity. Mentions of an entity can be names (Named Entities), nominals or pronouns.

After all the mentions in a text have been identified, we need to recognize their relations. The first issue is coreference, which is to classify mentions into entities. After coreference resolution, the three mentions above should be represented by one person entity named by "Bert-Olof Svanholm".

Another type of relation is the association between mentions (or entities). In the text *Mr. Svanholm is president of ABB Asea Brown Boveri Ltd.*, there is *person-affiliation* relation between *Mr. Svanholm* and *ABB Asea Brown Boveri Ltd.*; in the text *Centre College in Danville, KY* there is a *located-in* relation between the entities *Centre College* and *Danville, KY.*

Recognition of relations between entities can help us to connect events. In the disease outbreak domain, for example, this is particularly useful because events are inclusive. There could be initial report about an outbreak in a state, and then follow-ups about its spread in cities or neighboring states. Then recognizing the physical relations about locations will be crucial to relate events.

Relation detection finds relations between two entities in the same sentence. Miller et al. (2000)[43] proposed a statistical parsing model to generate customized relation tags between entities, which indicate the existence of relations. Roth and Yih (2002)[47] described a probabilistic approach which related the identification of entities and relations using global inference. There is also discriminative approach[57][20] in which kernels are developed to predict relations. These approaches will be covered in more detail in chapter 8.

Entity relations are also domain dependent and this is still a developing area. New topics may arise along with the ongoing research.

2.1.3 Event Extraction

This task includes extraction of an event and all of its arguments. For example, for a terrorist attack event, we expect to extract the time, location, perpetrator, target, damage, victims, etc. An event often spans several sentences, so we may need inference to figure out all the slots of an event. Logic and real world knowledge are often needed in this process.

As stated in section 2.1, extraction of an event is a challenging task given the variations of human language. Many approaches have been proposed and tried for this tasks, including rule-based approaches[22][6], symbolic learning

Project	Year	Domain
MUC-1	1987	Navy Operations
MUC-2	1989	Navy Operations
MUC-3	1991	Terrorism in Latin America
MUC-4	1992	Terrorism in Latin America
MUC-5	1993	Corporate Joint Venture and Microelectronics
MUC-6	1995	Corporate Management Succession
MUC-7	1998	Airplane Crashes/Rocket Launches

Table 2.2: List of the Message Understanding Conferences.

approaches[21][45][11] and statistical approaches[16][42]. We will explain them in detail in chapter 3.

2.2 The MUCs

The MUCs (Message Understanding Conferences)[1][2][3][4][5], sponsored by the U.S. Government, were the first attempt to standardize the task of Information Extraction and establish benchmark corpora. Starting with the first MUC in 1987, there were seven MUC evaluations carried out in a ten years span. These evaluations greatly promoted the research in information extraction. Table 2.1 lists the year and topics (domains) of each evaluation.

The goal of the MUC program was to provide a platform on which various IE approaches can be compared. In each evaluation, training data, test data and a scoring metric were provided to participants. Later evaluations were designed to be carried out with a real scenario constraint: for MUC-6 and 7, an IE system had to be developed within a month on a new domain. Each participant had to finish the development of their IE system and submit the result on test data within a month from the time training data is released. Then results from all participants were scored and ranked. There were also post-evaluation conferences to discuss the results, to identify new issues which emerged and to improve the understanding of information extraction research.

The MUCs were a great impetus to research in information extraction. Many new problems were identified and separated, such as named entity recognition and entity relation detection. The only downside is that due to the competitive nature of the evaluations, participating systems tended to converge to a few best performing approaches.

2.3 ACE Evaluations

The ACE (Automatic Content Extraction) evaluation program [1] is a successor to the MUCs. After the development of IE in the MUCs, the tasks of information extraction and their difficulties became better understood by researchers. So the ACE program aims to form solid foundations for each component technology of information extraction and eventually lead to the extraction of content from text at event or higher level. Therefore the tasks of information extraction have been tackled in a bottom-up fashion. The ACE program started with the building blocks of content extraction, such as named entity recognition and entity coreference resolution, and is stepping up to entity relation detection and event extraction.

[1]Task description: http://www.itl.nist.gov/iad/894.01/tests/ace/
ACE guidelines: http://www.ldc.upenn.edu/Projects/ACE/

Year	Tasks	Languages
2000	EDT pilot study	English
2001	EDT, RDC	English
2002	EDT, RDC	English
2003	EDT, RDC	English, Chinese, Arabic
2004	EDR, RDR	English, Chinese, Arabic

Table 2.3: List of the ACE evaluations. EDT and RDC are abbreviations of "Entity Detection and Tracking" and "Relation Detection and Characterization"

The ACE evaluations largely follow the scheme of the MUCs. Its development cycle includes specifying the tasks, developing training and test data, carrying out an evaluation and discussing the results from all participating sites. One difference from the MUC evaluations is that it is multi-source and multilingual. Each evaluation includes text from newswire documents and broadcast news transcripts, and some include text derived from OCR; it covers several languages: the latest evaluations included tasks in English, Chinese and Arabic. Table 2.2 lists the tasks and languages of the ACE evaluations.

The EDT (Entity Detection and Tracking)[2] task is to detect mentions of an entity including proper names, nominals and pronouns in selected semantic classes. All mentions detected in a document are grouped into entities by coreference resolution. The classes of interest cover the common types in news documents, such as names of persons, organizations, facilities and so on. The

[2]The EDT and RDC tasks were relabeled EDR (Entity Detection and Recognition) and RDR (Relation Detection and Recognition) for 2004.

RDC (Relation Detection and Characterization) task is to find predefined relations between entities, such as an *employee-of* relation between a person and a organization.

The tasks of ACE are being improved each time the evaluation is done. So the specification of each task and types of objects involved vary year by year. From the year 2005, the ACE project is making efforts to extract events in text.

2.4 Other Flavors of IE Research

The sections above survey IE research including supervised learning methods on free text. There are also other flavors of IE research, but they are not the focus of this thesis.

1) Unsupervised approach. Compared to hard-to-get annotated data, the amount of unannotated text is several orders of magnitude larger. Many approaches have been proposed to use this kind of text along with a small set of annotated data. The basic idea is to use the annotated domain data as seeds and extend the findings by exploring similar contexts in general text. The learning algorithms that have been applied include bootstrapping, active learning, co-training, etc. Unsupervised learning has also been a large research area in information extraction.

2) There is also research on extracting information from structured data, like HTML pages. Since a large amount of information on the web exists in organized forms like HTML pages, extracting this kind of information is very useful and practical. Text in this domain is structured or semi-structured, so syntax analysis is not so crucial as in extraction from free text. Similarity among structures of text is good clue to extract and classify information.

Chapter 3

Classical IE Approaches

In this thesis, Information Extraction is used in a narrow sense, that is to refer to deriving events from a selected domain and presenting the information in predefined templates. Event structure is defined by a template that usually contains named entities and other properties of the event.

Most of the classical IE approaches include three basic steps, preprocessing, finding pieces of evidence and merging them into events. Since there have been so many IE systems developed, it is not easy to classify them into distinct categories. This chapter will follow the general trend of natural language technology, which is a transition from pure hand-crafting to automated optimization, to introduce the IE models. These models will be roughly put into three categories: cascaded finite state models, example-based learning models and statistical models. In the following sections of this chapter, representative systems of each category will be discussed.

3.1 Cascaded finite state models

These are the dominant rule-based systems in Information Extraction. The idea of these models is that the "facts" relevant to an event can be described using patterns composed of syntactic constructs. With this idea the system consists of two major modules: a domain-independent syntactic analyzer to structure the input text, and a domain-dependent semantic analyzer to collect pieces of "facts" using patterns.

Both modules can be implemented using finite state rules designed by hand. So the core part of these models is a system of cascaded rules. The first module is syntactic processing which contains rules to parse the text input. This can be either full parsing or partial parsing, and the latter is often preferred due to its robustness. The second module extracts facts using higher level patterns based on the result of the syntactic processing.

After the two modules, a reference resolver is needed to merge the "facts" into events and to resolve the missing slots.

1. The FASTUS system[6] used cascaded finite state automata to recognize complex word(names, dates, etc), basic phrases and complex phrases. Anything that is not recognized as one of these phrases was ignored, which makes the whole system robust on inputs. Domain events were also encoded in finite state machines, where the input symbols were phrases. A merging procedure operating on the whole text combined domain events into full events after resolving corefered entities.

2. The Proteus system[22] used cascaded finite state transducers to detect management succession events. At a low syntactic level, transducers were

generated to find proper names, noun groups and verb groups; at a higher syntactic and semantic level, transducers were defined to account for basic events. For example start-job($person$, $position$) could match the text "IBM named Harriet Smith president" and "succeed($person1$, $person2$)" could match the text "He succeeds Mr. Ray". Then a merging procedure combined these event pieces using coreference and logical correlations among them.

3. The LaSIE-II system[27] included finite state recognition of domain-specific lexical patterns, partial parsing using a restricted context-free grammar and quasi-logical form (QLF) representation of sentence semantics. The cascaded syntactic parser could identify names and phrases. It produced a parse tree of a sentence: if full parsing failed, the best partial parse result is given. Then the parsing result of a sentence is mapped to a QLF representation.

   Semantic concept nodes were mapped from predefined constructs. There were three types of concept nodes in its top level ontology: objects, events and attributes, which represented either domain specific knowledge or general world knowledge. The lexical patterns matched an event in the text and generated hypothetical domain slots. For instance, the match of a *launch_event* pattern could generate slots of a vehicle, a payload, a date and a launch site. Then a discourse analyzer unified the concept nodes and resolved the hypothesized event slots.

The finite state models have demonstrated remarkable performance. For example, the Proteus system rivaled other systems in the MUC-6 evaluation. However, developing and managing rules by hand requires high human expertise.

The domain specific rules can not be easily reused for a new domain. Shifting the system to a new domain is comparable to the work of developing a new system.

3.2 Example-based Learning Models

The motivation of example-base learning models is to reduce human effort in building or shifting an IE system. Instead of creating patterns by hand, these models derive rules via generalization of examples. The learning process starts with patterns from specific examples (tagged text segments) and the syntactic structure of the surrounding text. Then it tries to generalize the patterns inductively by relaxing constraints and merging patterns. This process continues until no more generalization can be done without introducing too many errors. The result is a set of generalized patterns. The learning procedure is automatic or semi-automatic once the initial examples are given.

1. AutoSlog[44] learns a dictionary of patterns called concept nodes or concepts from text to match phrases for single slot. Each pattern has an anchor word, most often the head verb, to activate it. An example of a concept node is

   ```
   <subject> passive-verb
   ```

   with an anchor word *bombed*. When the concept node is activated and matched, it produces the target name of the bombing, which is the content of *subject*. AutoSlog could propose concept nodes from annotated examples using the 13 predefined concept types; however, it did not try to generalize the concept nodes and it also required human review to verify the learned concepts.

2. CRYSTAL[53] used inductive learning to generate a concept dictionary from pre-annotated training data. Each concept is a typed case frame along with constraints, usually syntactic constituents including subject, verb phrase, object and prepositional phrase. Constraints can be generalized from head words to semantic classes. It started from initial concept nodes, which could be as specific as word sequences, and tried to unify two similar concept nodes by relaxing the constraints. The merged concept was verified against the whole corpus. This process continued until no unification can be executed.

   The verification can be relaxed by setting an error bound parameter to make the learning process robust. The parameter also determined the trade-off between precision and recall of the learned patterns. The expressive power of this algorithm depends on the structure of the concept nodes and on the sophistication of the preprocessor which generates text segments as initial clues.

3. RAPIER[11] used a generalization algorithm similar to Inductive Logic Programming to derive symbolic rules for extraction of posted computer jobs. It only assumed simple syntactic preprocessing such as tokenization and Part-of-Speech tagging. The three parts of a rule are pre-filler, filler and post-filler, which form a linear sequence of words, POS tags or semantic classes. The learning algorithm is a bottom-up process: it started from a random pair of rules and kept compressing the rules as their least-general generalization (LGG). This process terminated when

the number of unsuccessful rule compression attempts exceeded a given threshold. The experimental performance on job postings was reported comparable to CRYSTAL. However, the target domain was constrained text. It is not clear how this algorithm would perform on free form text like news articles.

To get good coverage, the amount of training data is a crucial factor for these learning models. For example, the performance of CRYSTAL dropped substantially when the training set size was reduced from 1000 to 100 articles, simply because many morphological variations were not covered in the small corpus.

But generating a good amount of initial training data requires substantial manual work. So the example-based learning models did not solve the portability problem completely. It shifted the human work from developing rules to tagging examples.

The patterns used by these models were predefined by templates, which confines the expressive power of patterns. The design of pattern templates also requires human knowledge and it may be domain specific.

3.3 Statistical Models

Statistical event models were inspired by statistical parsing, which can produce multiple sentence structure alternatives along with their probabilities. Instead of separating syntactic and semantic patterns, statistical models tried to use an integrated model at sentence level to predict syntactically related units. There is no need to derive patterns, except that specially designed labels had to be inserted into a normal grammar to represent the semantic relations.

However, statistical parsing requires large amount of training data and labeling parse trees manually from text is an expensive process. In practice, the statistical models were only tried on small subtasks of IE, such as entity relation detection. There has been no attempt to model events in a sentence.

1. The SIFT system[42] developed by BBN used a statistical parser augmented with semantic labels to detect named entities and relations among them. A probabilistic cross-sentence model was also deployed to recognize long distance relations. The sentence model was trained from the Penn Treebank as general knowledge. The domain data was hand-labeled trees augmented with named entities and their relations, for instance the *location-of* relation between a company name and a location.

   To implement this, additional semantic nodes and labels were inserted into a regular parse tree. The resulting parser could then produce a special parse tree which may contain the semantic information indicated by the special tags. The cross-sentence model was a binary decision model that used features to determine if two entities in different sentences are corefered. Since there were few instances of cross-sentence relations, its effect on the whole system was not significant.

2. Collins et al. (1997)[16] used a carefully designed PCFG grammar to model events. The model treated words irrelevant to any event as noise. The target domain was MUC-6 management succession. All the events were divided into seven categories according to the appearance of names in template. For example, {OUT, POST} represented an event category that a person leaves a position. There were five non-terminal leaves in the grammar, namely IN, OUT, POST, IND(indicator word) and NOISE.

The CFG grammar productions were generated manually from training data, and probabilities for them were estimated using MLE.

The slot f-score of this model was 77.5% with indicator words labeled in test sentence. It is not clear how this model compares to other systems because it assumed specially annotated test data.

3.4 Limitations of Traditional Models

Most of these approaches are based on generative models. They assume that events occur in text in certain patterns and these patterns can be discovered and expressed by a metalanguage. However this assumption may not be completely right and it has limitations.

The pattern discovery and matching of these approaches have to be based on deep processing of text. That's because only at the syntax level can patterns be effectively generalized. Even though shallower levels of information (e.g. word sequence) may still be helpful, incorporating them would make patterns too complicated to manipulate. So they are ignored by most of the models.

Another issue is that deep processing is not very accurate. It depends on shallower processing results which may contain errors. For example the state-of-the-art performance of sentence parsing is about 88%. The 10% errorful results are irrecoverable for approaches based solely on parsing.

Chapter 4

Kernel-based Learning

This section covers the basics of Kernel Methods and Support Vector Machines and their applications in Information Extraction and Natural Language Processing. Since we only focus on their applications, the complete Machine Learning theories behind them are not covered. We only focus on their intuition and applications. References are given for deeper theoretical interests.

To apply supervised learning models to a task, the common way involves a feature generation process to transform an object into a feature vector of real values. Then the training or testing procedure can be carried out in the real vector space. A typical classification problem can be formalized as this: given a set of labeled real vectors

$$(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n),$$

find a function $f(X, W)$ that satisfies $y_i = f(X_i, W)$, in which X_i $(1 \leq i \leq n)$ is a vector $(x_1, ..., x_N)$; y_i is the class label of X_i and W contains the parameters of function f. In a two class problem, y_i could be either -1 or +1, while in a multi-class classification problem, y_i could be an integer representing the class of X_i. In this chapter, we only focus on the two class classification problems. A multi-class problem can be reduced to two class problems using schemes like one-against-all or pairwise classification.

4.1 Linear Classifier

Given $X = (1, x_1, ..., x_N)$, a linear classifier or Perceptron[38][46] is a function f with parameter $W = (w_0, w_1, ..., w_N)$, such that

$$y = sign(f(X, W)) = sign(< W, X >) = sign(\sum_{i=0}^{N}(w_i x_i)). \qquad (4.1)$$

$sign(x)$ is the sign function that gives +1 when $x \geq 0$ and -1 when $x < 0$. So $f(X, W) = 0$ is a hyperplane that separates the two classes. The extra dimension (dimension 0) in X and W is to accommodate the scalar distance from the origin to the hyperplane. So to determining function f is equivalent to to finding the vector W.

Given labeled training data $S = (X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)$, the iterative learning procedure in table 4.1 produces the weight vector W in f.

It can be proved that if the examples are separable in a two class classification problem, the Perceptron learning procedure terminates within a finite number of iterations.

In non-separable cases, since there will always be classification errors in training, a stopping criterion has to be set to terminate the learning procedure. Such a criterion could be a threshold that specifies the number of tolerable training errors.

From the training procedure we can derive that

$$W = \sum_{i=i}^{n} \alpha_i y_i X_i.$$

```
W(0) := (0,..., 0);
repeat
for each $(X_i, y_i)$ in S do
   if $y_i < W(t), X_i >< 0$
     W(t+1) := W(t) + $y_i X_i$;
   end if
end for
until no mistakes found within the for loop
```

Table 4.1: The iterative learning procedure of Perceptron

To get W we only need to compute $\alpha_i, i = 1, ...n$. To classify a new example X, the prediction would be

$$sign(\sum_{i=i}^{n} \alpha_i y_i < X_i, X >).$$

This is called the dual form, in which the primal variables are eliminated and everything is represented by X_i. In the next section we will show that this can be generalized to form a kernel space.

The Perceptron training procedure terminates when it finds the first successful separation hyperplane (or the first hyperplane that satisfies the stopping criterion). So the separation it produces may not reflect the actual distribution of the examples correctly. Therefore the chances are good that even when the training performance is good, the classifier may not do well on unseen examples. This problem is more severe in non-separable cases, in which the resulting hyperplane can be very skewed from the real example distribution.

Some real problems could be linearly separable, such as using bag-of-words to classify text. But lots of others are linearly non-separable problems. Since linear classifiers do not generalize very well on unseen examples, their application is very limited.

However, in the next section we will see that a simple improvement to the linear classifier resulted in a much more powerful model, the Support Vector Machine.

4.2 Support Vector Machine

Although the Support Vector Machine (SVM)[8][18][55][19] arrived via a theoretical path associated with Structural Risk Minimization (SRM), its basic idea is rather simple. Instead of producing any successful separation hyperplane, a Support Vector Machine finds the one that separates the examples with largest margin.

In the separable case, a Support Vector Machine assumes that given a set of labeled examples :

$$S = (X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n),$$

where $y_i \in \{-1, +1\}$ is the class label of X_i, there exist two parallel hyperplanes $p1$: $< W, X > +b = 1$ and p_2: $< W, X > +b = -1$ bounding each class of examples. Figure 1 illustrates the case in two dimensions. (A Perceptron could produce any hyperplane between p_1 and p_2.)

The two classes of examples satisfy either $< W, X > +b \geq 1$ or $< W, X > +b \leq -1$. We can combine the two constraints as one:

$$y_i(< W, X_i > +b) \geq 1 \qquad 1 \leq i \leq n \tag{4.2}$$

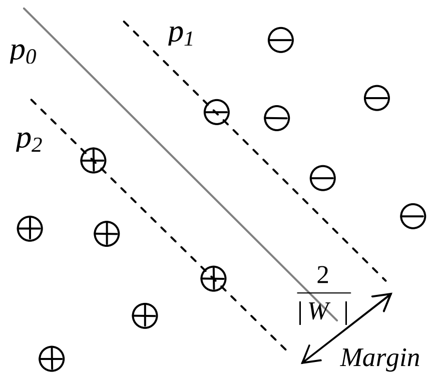

Figure 4.1: Separable case in Support Vector Machine

An SVM produces the hyperplane p_0: $< W, X > +b = 0$, whose margin to p_1 or p_2 is the largest among hyperplanes. Intuitively p_0 may best reflect the separation of all the examples, either seen or unseen.

Since the margin (or distance between p_1 and p_2 in Figure 4.1) is $2/\|W\|$ maximizing the margin is equivalent to minimizing a loss function $L_P(W) = \|W\|^2$, subject to the separation constraints (4.2).

To solve the optimization problem, we can use the following Lagrangian:

$$L(W, b, \vec{\alpha}) = \frac{1}{2}\|W\|^2 - \sum_{i=1}^{n} \alpha_i(y_i(< W, X_i > +b) - 1). \tag{4.3}$$

At the optimal point, we have the following equations:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{4.4}$$

and

$$\frac{\partial L}{\partial W} = W - \sum_{i=1}^{n} \alpha_i y_i X_i = 0 \tag{4.5}$$

From (4.5) we can see that $W = \sum_{i=1}^{n} \alpha_i y_i X_i$ also has a dual form. Combining all the equations we can get the following optimization problem:

$$\begin{aligned} max_{\vec{\alpha}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j < X_i, X_j > \\ subject\, to \quad & \alpha_i \geq 0, \quad 1 \leq i \leq n \\ & \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned}$$

This is a quadratic optimization problem. The gram matrix is formed by the inner products of examples $\{X_i\}$, so it is symmetric and positive semi-definite. Then the search space for W is concave, which guarantees a unique solution no matter how many examples are involved.

In the non-separable case, classification errors have to be allowed in training. The constraints in (4.2) are relaxed by slack variables ξ_i ($\xi_i > 0$). This is shown in 4.6.

$$y_i(< W, X_i > +b) \geq 1 - \xi_i \tag{4.6}$$

When $0 < \xi_i < 1$, example X_i still gets classified correctly, although it falls into the margin area between p_1 and p_2. When $\xi_i > 1$, example X_i would fall onto the other side of p_0, thus we will see a training error on it. Figure 2 illustrates the non-separable case for SVMs.

This allows the learning of SVMs to ignore noisy examples on the boundary of each class and fit the separation hyperplane to the real distribution of examples. This also happens even when examples are separable. So an SVM can produce good generalization based on all examples rather than examples on the class boundaries.

In the non-separable case, the loss function is a trade-off between minimizing training error and the margin. The problem can be formalized as

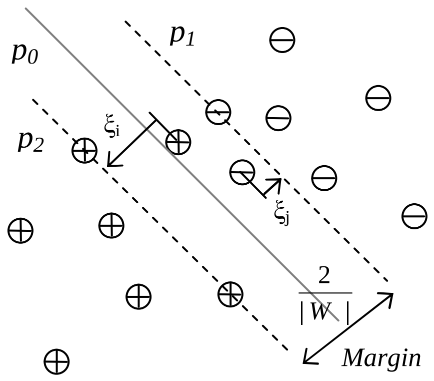

Figure 4.2: Non-separable case in Support Vector Machine

$$\text{minimizing } L_P(W, \bar{\xi}) = \|W\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{subject to}$$

$$y_i(< W, X_i > +b) \geq 1 - \xi_i \qquad 1 \leq i \leq n,$$

where C determines the trade-off between the margin and training errors.

The optimization of SVMs has many interpretations. In the SRM theory, this can be explained by the trade-off between training errors and generalization ability of an SVM. The margin of an SVM determines its generalization ability. Maximizing the margins reduces the VC dimension of the SVM, thus increasing its capacity or generalization ability. In SVM training, we don't want the classifier to overfit the training data by simply minimizing training errors, nor do we want it to make too many errors to represent the training data at all just for better generalization. In real applications, the way to find an optimal point in this trade-off is to use validation data to determine C.

As a computing problem on large matrices, many efficient solutions have been proposed. One popular solution is SMO (Sequential Minimal Optimization), which breaks the big problem into small two dimensional analytical problems. There are also many efficient SVM implementations available for research purposes, such as SVM^{light}[30] and libSVM[12].

4.3 SVM in NLP

SVM's generalization ability in high dimensional feature space makes it a good model for NLP tasks. Many NLP problems involve lots of lexical or syntactic features. Traditional learning models can not handle large number of features, so feature selection has to be applied to eliminate most of the features. We may lose useful information about the original text by doing this. However, with SVM, it is possible to use all the features and let the classifier decide the separation criteria of examples.

In text classification, all the words in a document and their frequencies are usually taken as features. So the feature space dimension could be tens of thousands. Many of the features are relevant so they can not be easily removed. Joachims[31] gave a theorectical analysis of using SVM to classify text. In his experiment on the benchmark Reuters dataset, the average performance was a few percent better than k-NN.

Other applications were also reported using SVM's that produced state-of-the-art performance. Such applications include NP chunking and NE recognition. Kudo et al. (2001)[34] applied SVM to NP chunking, in which each word in a sentence was classified by SVM to decide its chunk label. Features of a word include the two preceeding and succeeding words, their Parts-of-Speech and the chunk labels of the two preceeding words. Pairwise classification was used to decide between two chunk labels. The final prediction is the label voted from pairwise classifications. The experimental result on base NP chunking outperformed previous approaches.

SVMs were also applied to Named Entity (NE) recognition. Yamada et al. (2001)[56] applied the previous SVM chunking scheme to do Japanese NE detection. In Isozaki et al. (2002)[28], redundant character morphological information (n-best results of a statistical morphological analyzer) was used as input to an SVM to recognize NE in Japanese. The result on the IREX NE extraction task was better than the previously reported methods.

Mayfield et al. (2003)[37] developed a language independent NE recognizer using SVM. The input features include word compositional information, position of the word in the sentence or document and other information from a statistical Part-of-Speech tagger. The experimental results on English and German were reported better than a basic HMM NE recognizer.

There also other SVM applications in NLP which involves kernels. They will be covered in section 4.6.

4.4 Kernel Space

In the dual form of the Perceptron algorithm or Support Vector Machine, W is a linear combination of X_i:

$$W = \sum_{i=i}^{n} \alpha_i y_i X_i$$

Then the original problem of finding W is converted to determining α_i. To classify unseen examples (test phase),

$$\begin{aligned} y = sign(f(X, W)) =& sign(< W, X > +b) \\ =& sign(\sum_{i=0}^{N}(\alpha_i y_i < X_i, X >) + b). \end{aligned}$$

From the dual form we can see that the only operator involved in training and test is the inner product between two examples. Mathematically this inner product operator can be replaced by a binary function $K(X_i, X_j)$ which produces a real value. As long as the Gram matrix $V = \{v_{ij}\}_{n\times n} = \{K(X_i, X_j)\}_{n\times n}$ is symmetric and positive semi-definite, it can be seen as a new inner product defined on $\{X_i\}$. If a function $K(X_i, X_j)$ has these properties, it is called a kernel function or just kernel.

The nice thing about kernel functions is that their domain does not have to be vectors, they can be the original objects. As long as a function satisfies the conditions, it becomes a valid kernel.

For a kernel function $K(X_1, X_2)$ there exists an implicit mapping

$$\begin{aligned} \Phi : \ell &\mapsto F_N \\ X &\mapsto \Phi(X) \end{aligned}$$

such that $K(X_1, X_2) =< \Phi(X_1), \Phi(X_2) >$. Φ maps the X in the original object domain ℓ to a high dimensional vector in space F_N. N could be a very large number or even infinity. For a polynomial kernel $K(X, Y) = (< X, Y >)^2$, where $X = (x_1, x_2)$, $Y = (y_1, y_2)$ are vectors,

$$\begin{aligned} K(X, Y) =& (x_1y_1)^2 + 2x_1y_1x_2y_2 + (x_2y_2)^2 \\ &= < (x_1^2, \sqrt{2}x_1x_2, x_2^2), (y_1^2, \sqrt{2}y_1y_2, y_2^2) > \end{aligned}$$

So $\Phi(X) = \Phi((x_1, x_2)) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

This can be seen as a way of creating high-order features from the original feature space with very little computational cost. The dimension of the new

Polynomial	$K(x,y) = (k(x,y)+c)^d$
Redial Basis Function(RBF)	$K(x,y) = exp(-\|x-y\|^2/2\sigma^2)$
Sigmoidal	$K(x,y) = tanh(\kappa k(x,y)+\theta)$
Inv. multiquadric	$K(x,y) = 1/\sqrt{\|x-y\|^2+c^2}$

Table 4.2: List of common kernels. $k(x,y)$ can be either a normal inner product on a vector space or a user-defined kernel function on an object domain. $\|x-y\|^2$ can be computed as $k(x,x) - 2k(x,y) + k(y,y)$.

feature space depends on the function itself. A polynomial kernel

$$K_p(x_i, x_j) = (< x_i, x_j > +1)^d$$

applied to vectors creates a new feature space of dimension m^d where each feature is a combination of no more than d original features. For example, when $d = 5$ and $m = 256$, the dimension of the new feature space is $O(2^{40})$, which is a tremendous number. But the computation of this kernel is trivial.

For some kernels, the kernel space can not be represented by a finite number of features. For instance a Gaussian kernel

$$\phi_p(x_i, x_j) = e^{-\|x_i - x_j\|^2/2\sigma^2}.$$

The dimension of its underlying vector space is infinite.

There is a well-known caveat of using a large number of features named the "Curse of Dimensionality": when the dimension of the example space increases, the number of samples we need to estimate the example distribution increases exponentially. So even though kernels can introduce many more features, the limited number of examples in any real applications may limit the effectiveness of training a classifier. Fortunately, it can be proved that the complexity of a

Support Vector Machine depends mainly on its margin, not the dimension of the example space. So we can control the complexity of an SVM by controlling its margin during training. It has been shown that it is possible to train an SVM effectively on normal size of data with millions of features.

Table 4.2 lists some common kernels which have been applied in various domains. The values of variables x and y in kernel K need not to be a feature vector, they can be the original object in domain ℓ, such as images or text documents. The kernel matches two objects directly and produces a similarity value. K can be either an implementation that integrates the normal procedure of feature generation and inner product computation, or just a comparison of two objects with a customized similarity metrics. As long as K satisfies the conditions, it forms a valid kernel. The custom kernels on the object domain are of the most interest to us. Most of the kernels designed in this thesis are of this type.

4.5 Kernel Properties

Kernel functions have many nice properties. Most of them can be derived from the properties of positive semi-definite matrices. The following properties are of interest to IE applications.

1. Closed under scalar product. If $K(x, y)$ is a kernel on $X \times Y$, α is a real scalar, then $\alpha K(x, y)$ is a kernel.

2. Closed under sum. If $K_1(x, y)$ and $K_2(x, y)$ are kernels on $X \times Y$, the sum $K_1(x, y) + K_2(x, y)$ is a kernel. With the previous property, the kernel set is closed under linear combination.

3. Closed under product. If $K_1(x, y)$ and $K_2(x, y)$ are kernels on $X \times Y$, the product $K_1(x, y) \times K_2(x, y)$ is a kernel. With the previous properties, the kernel set is closed under polynomial transformation.

4. Closed under tensor product. If $K_1(x, y)$ is a kernel on $X \times Y$ and $K_2(u, v)$ is a kernel on $U \times V$, then

$$K_1 \otimes K_2((x, u), (y, v)) = K_1(x, y) \times K_2(u, v)$$

is a kernel on $(X \times U) \times (Y \times V)$.

5. Closed under direct sum. If $K_1(x, y)$ is a kernel on $X \times Y$ and $K_2(u, v)$ is a kernel on $U \times V$, then

$$K_1 \oplus K_2((x, u), (y, v)) = K_1(x, y) + K_2(u, v)$$

is a kernel on $(X \times U) \times (Y \times V)$.

Property 1 through 3 apply to kernels within the same domain. Property 4 and 5 are combinations of kernels from different domains. These are often useful when we need to integrate kernels representing different syntactic sources. For instance, when the new kernel sums a word trigram kernel on sentence S and a tree kernel on parse tree T of sentence S, its domain is $S \times T$, the direct sum of the word sequence and parse tree of the sentence. Property 4 is a linear combination. If we see the two individual kernels representing features from two domains, then the resulting kernel simply adds up all the features; Property 5 is high order extension similar to a polynomial kernel, but operates on two different domains. In terms of features, the resulting kernel produces all feature pairs from each domain.

With all five properties, we can apply polynomial combination to any finite number of kernels from any domain and the result is still a valid kernel on a new domain. The new domain could be nested direct sums of the original domains.

For information extraction tasks, we can develop kernels to represent syntactic information from different processing levels of text, such as tokenization, chunking or parsing. Then the properties of kernels provide flexible ways to combine and extend them. This forms the theoretical basis of the approach proposed in this thesis.

4.6 Kernel Applications in NLP

The inputs to NLP applications are structures like word sequences, trees or graphs. Most NLP applications involve finding syntactic patterns from these structures. A traditional way to tackle this problem often involves human knowledge to identify smaller pieces as features and then apply a learning algorithm on the features. Obviously, this could be insufficient due to the limitation of human observation over large amounts of data.

With kernel functions things can be much easier. All we need is a kernel function to match two structured objects and produce a similarity value. As long as the kernel is mathematically valid, it can be plugged into any learning algorithm which has dual form representations.

Several types of kernels have been designed for general NLP structures, such as the string kernel[36] for word sequences and convolution kernel[26] for any discrete structures. There are also customized kernels designed for specific tasks, such as the subtree kernel[14], dependency graph kernels[15][20] and relation kernel[57].

4.6.1 String Kernel

The string kernel was originally designed to improve document classification. The prior approaches (Salton et al. 1975[49] and Joachims 1998[31]) were based on word vectors, in which each occurring word and its frequency in a document is considered as a feature. But the ordering of words is ignored. To address this problem, Lodhi et al. (2001)[36] designed a string kernel on word sequences which considers either contiguous or non-contiguous subsequences of words as features. The kernel was defined as the following:

> Definition 1 (String subsequence kernel) Let Σ be a finite alphabet. A string is a finite sequence of characters from Σ, including the empty sequence. For strings s, t, we denote by $|s|$ the length of the string $s = s_1 ... s_{|s|}$, and by st the string obtained by concatenating the strings s and t. The string $s[i:j]$ is the substring $s_i \dots s_j$ of s. We say that u is a subsequence of s, if there exist indices $i = (i_1, \dots, i_{|u|})$, with $1 \leq i_1 < \dots < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, \dots, |u|$, or $u = s[i]$ for short. The length $l(i)$ of the subsequence in s is $i_{|u|} - i_1 + 1$. We denote by Σ^n the set of all finite strings of length n, and by Σ^* the set of all strings
>
> $$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$
>
> We now define feature spaces $F_n = R^{\Sigma^n}$. The feature mapping ϕ for a string s is given by defining the u coordinate $\phi_u(s)$ for each $u \in \Sigma^n$. We define $\phi_u(s) = \sum_{i:u=s[i]} \lambda^{l(i)}$, for some $\lambda \leq 1$. These features measure the number of occurrences of subsequences in the string s weighting them according to their lengths. Hence, the inner

product of the feature vectors for two strings s and t give a sum over all common subsequences weighted according to their frequency of occurrence and lengths

$$K_n(s,t) = \sum_{u \in \Sigma^n} (\phi_u(s)\phi_u(t) = \sum_{u \in \Sigma^n} \sum_{i:u=s[i]} \lambda^{l(i)} \sum_{j:u=t[j]} \lambda^{l(j)}$$
$$= \sum_{u \in \Sigma^n} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{l(i)+l(j)}.$$

The decay factor λ is used to penalize the gaps in the subsequence. When the length of the gaps is n, the penalty for the value is λ^n. For a long subsequence with long gaps in it, the weight assigned to it would be exponentially low. So they would not affect the result very much. This is reasonable considering there is more noise in combinations of long distance words. The penalty of gaps can be adjusted by λ: a larger value of λ extends the size of the window allowing gaps.

It can be shown that using dynamic programming kernel $K_n(s,t)$ can be computed in $O(|s||t|)$ time. This is interesting considering that the number of features it represents is exponential in the sequence size.

This kernel was experimented with text categorization on the benchmark Reuters dataset. The result of the contiguous kernel was slightly better than just using word features; part of the reason could be the geometric decay factor that might not capture the long distance word dependency very well.

4.6.2 Convolution Kernel for Natural Language

Collins et al. (2001)[15] proposed a kernel for common NLP tasks, such as tagging and parsing, based on convolution kernels[Haussler 1999]. The tree kernel counts common subtrees between two parse trees: $K(T_1, T_2) =< \vec{h}(T_1), \vec{h}(T_2) >$, where $h(T) = (t_1, ..., t_d)$ is a vector containing all the d subtrees in tree T. Then for two trees with N_1 and N_2 nodes

$$\begin{aligned} K(T_1, T_2) &= \sum_{i=1}^{d_1} \sum_{j=1}^{d_2} t_i t_j \\ &= \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} C(n_i, n_j), \end{aligned}$$

where $C(n_i, n_j)$ is a function that counts common subtrees rooted at node n_i in T_1 and n_j in T_2. The computation of $K(T_1, T_2)$ takes $O(N_1 N_2)$ time using dynamic programming.

Since there are an exponential number of subtrees in a tree, $K(T, T)$ on two identical trees would produce a huge value. To avoid this problem, a penalty factor λ has to be used to down-weight large subtrees. Then

$$K(T_1, T_2) = \sum_{i=1}^{d_1} \sum_{j=1}^{d_2} \lambda^{size_i} t_i t_j.$$

They also proposed a linear model to transform the ranking problems in Natural Language Processing to margin-based classification problems. The basic idea is to use classification to separate the correct answer (not necessarily the one with best score) from other candidates. Each candidate x_i is represented by a kernel $h(x_i)$. Then a standard kernel-based classifier with slight modification can be used. This scheme was tried with Voted Perceptron for reranking parsing alternatives. The performance improved from 74% of a standard PCFG to around 80%.

4.6.3 Other Kernels

There are many other kernels designed for a specific task. Zelenko (2003)[57] designed a kernel called a relation kernel on shallow parse trees to extract relations between entities, such as a *person-affiliation* relation between a person and organization. This kernel is a recursive match of trees from the root down to the leaves. At each node, a subsequence kernel similar to string kernels is used to match sequences of child nodes. This kernel produced good performance on extraction of two types of entity relations.

Suzuki et al. 2003[54] described a kernel for Hierarchical Directed Acyclic Graph (HDAG). It is a convolutional kernel which allows approximate match of components between two HDAGs. Each node in a HDAG is associated with attributes. The features for each HDAG are attribute sequences on a node path. Nodes can be skipped for inexact match. The kernel performed better than a string kernel or dependency graph kernel in experiment on question classification.

There is a general algorithm named Rational Kernel[17] that subsumes the previous two kernels and the string kernels. It is defined on finite state automatons or transducers. It can be applied to a lattice structure produced during a Viterbi search.

4.6.4 Comments about Convolution Kernels in NLP

Most of the kernels designed for NLP tasks are descendents of convolution kernels. They match the similarity of two discrete structures by counting common subcomponents of them. If two large components match, then all the smaller components within them match also. The similarity value produced by the kernel is exponential in the size of the common component. So an example is similar to itself in exponential magnitude, while comparing with a different example, the similarity is a constant number. Collins et al. (2001)[14] shows that for the subtree kernel, the difference can be 10^6 vs. 100 in magnitude. Using a penalty factor λ $(0 < \lambda < 1)$ does not solve this problem completely. It reduces the value differences, but there is not good justification about the weighting scheme.

In terms of the kernel matrix, the diagonal values would be much larger than other entries. This may cause overfitting in learning, in which the model tends to memorize the examples in the training data and would not recognize an unseen example unless it is very similar to a seen example. In terms of performance, overfitting incurs high precision but low recall. This problem was addressed by Schoelkopf et al. (2002)[50]. A element-wise transformation is proposed to reduce the value difference, for example by applying a polynomial kernel with a degree $0 \leq p \leq 1$. However, it is not clear how the resulting kernel space relates to the original one after the non-linear mapping. The solution does not work very well in general case.

On the other hand, the same problem in feature-based kernels is tolerable. For these kernels the number of features is a low order polynomial of the size of the objects, e.g. the number of words in a sentence or number of nodes in a parse tree. The diagonal values are linear in the number of features. Thus the difference between values on the diagonal and other entries is small. In our experiments, overfitting is not a big issues for these kernels.

In the work presented in the following chapters, most of the kernels are feature-based kernels.

Chapter 5

A Discriminative Model

5.1 Motivation

To address the problems existing in prior IE approaches, a new discriminative model based on the kernel method is proposed here. It incorporates different levels of syntactic information to find useful clues for an IE task. The idea is that an IE model should not commit itself to only deep analysis. Shallow information, such as word collocations, may also give us important clues.

From chapter 4, we can see that kernel functions can match objects directly. Kernels can be seen as representations of objects using large number of features. In this model, we will design kernels to represent each level of processing result and combine them using kernel properties. Many classifiers can be used with kernels. The most popular ones are SVM (Support Vector Machines), KNN (k-Nearest-Neighbors), and voted perceptrons.

Support Vector Machines are linear classifiers that produce a separating hyperplane with largest margin. This property gives it good generalization ability in high-dimensional spaces, making it a good classifier for our approach where using all the levels of linguistic clues could result in a huge number of features. Given all the levels of features incorporated in kernels and training data with target examples labeled, an SVM can pick up the features that best separate the targets from other examples, no matter which level these features are from. In cases where an error occurs in one processing result (especially deep processing) and the features related to it become noisy, an SVM may pick up clues from other sources which are not so noisy. This forms the basic idea of our approach. Therefore under this scheme we can overcome errors introduced by one processing level; more particularly, we expect accurate low level information to help with less accurate deep level information.

5.2 The Kernel-based Model

This discriminative framework makes no assumption about the text structure of events. Instead, kernels are used to represent syntactic information from various syntactic sources. The structure of this model is shown in Figure 5.1.

In this model, text is first extracted from input documents (usually articles in SGML format). Then it is preprocessed at different levels. The preprocessing modules include a part-of-speech tagger, name tagger, sentence parser and GLARF dependency analyzer, but are not limited to these. Other general or custom tools can also be included.

The triangles in the diagram are kernels that encode the corresponding syntactic processing result. Individual kernels can be combined into composite kernels as input to a classifier. In the training phase, targets are labeled in the text as examples to train the classifier. In the test phase, unlabeled text is processed in the same way as training data. Candidate examples are generated

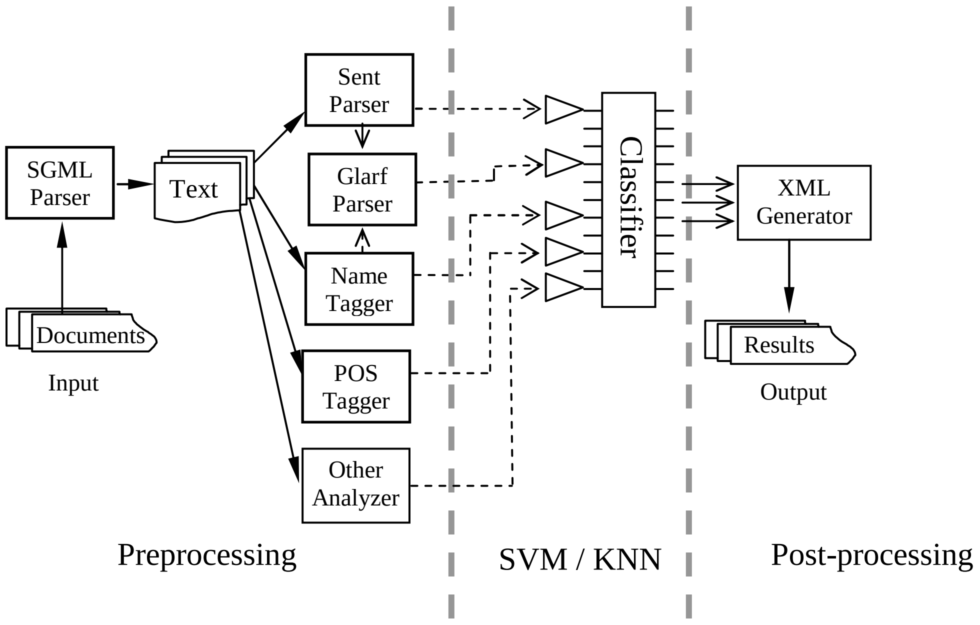

Figure 5.1: Infrastructure of the discriminative model

from the text based on the preprocessing results. Then the classifier can make predictions on candidates to identify the targets. The main kernel we propose to use is based on the dependency analysis produced by GLARF[39], mostly due to its ability to capture more regularizations in text. Chapter 6 will introduce features of GLARF and show examples of its outputs.

Most IE tasks are multi-class classification. After each classifier makes predictions, post-processing can be applied to resolve conflicts or to improve the predictions using other heuristics. The final result is outputted in XML.

5.3 Syntactic Kernels

To make use of syntactic information from different levels, we can develop kernel functions or syntactic kernels to represent a certain level of syntactic structure.

The possible syntactic kernels include

1. Sequence kernels: representing sequence level information, such as bag-of-words, n-grams or a string kernel.

2. Phrase kernels: representing information at an intermediate level, such as kernels based on multiword expressions, chunks or shallow parse trees.

3. Parsing kernels: representing detailed syntactic structure of a sentence, such as kernels based on parse trees or dependency graphs.

These kernels can be used alone or combined with each other using the properties of kernels. They can also be combined with general kernels like polynomial or RBF kernels to generate a high-order, non-linear decision surface. This can be done either on individual kernels or on the composite kernel.

In practice each kernel can be tested for the task as the sole input to a classifier, to determine if this level of information is helpful or not. After figuring out all the useful kernels, we can try to combine them to make a composite kernel as final input to the classifier. The way to combine them and the parameters in combination can be determined using validation data.

Since the preprocessing modules are standard analyzers, once a kernel is developed for a certain level of information, it could be reused when the underlying domain changes.

5.4 Applications

Many information extraction tasks can be implemented by this model, such as named entity recognition, entity relation detection and slot filler detection for events. In this thesis, experiments are carried out on entity relation detection and slot filler recognition.

Chapter 6

GLARF: The Dependency Analyzer

The Treebank-based parsers currently exhibit the best performance among parsers. They produce consistent tagging following the specification of Penn Treebank II (PTB-2). However, for applications like information extraction, PTB-2 parsing does not provide detailed information about grammatical relations. For example, it doesn't provide information about object, indirect object, appositives, etc.

In section 2.1, examples are given to illustrate some of the common grammatical phenomena an information extraction application has to consider. Such phenomena may include passive verbs, nominalizations, clause structures and conjunctions. Based on PTB-2 parsing results, an information extraction approach has to generate separate patterns to cover each case, which makes it difficult to generalize when training data is sparse.

GLARF[39] (Grammatical and Logical Argument Representation Framework) is a deep text analyzer which aims to capture more regularizations based on PTB-2 parsing. It recognizes noncanonical constructions (passives, conjunctions, filler-gap constructions, etc) and represents them in terms of canonical forms (simple declarative clauses). GLARF implementations can produce results in different forms. In our experiments, the dependency triples in the form of (ROLE FUNCTOR, ARGUMENT) were used. An example of this dependency triple is

```
(OBJ named, Harriet_Smith),
```

which can be derived from "IBM named Harriet Smith president" or "Harriet Smith was named president by IBM". OBJ is the type of this predicate argument relation, and "Harriet_Smith" is the object of "named".

GLARF produces logical relations instead of surface relations. For the sentence "Harriet Smith was named president by IBM", the dependency (OBJ named, Harriet_Smith) is a logical relation.

The GLARF analysis of a sentence can be assembled into a dependency graph, where each typed predicate-argument relation is represented by a labeled dependency arc. GLARF can also include base form nouns and verbs where words are normalized by morphological analysis. For a verb, "named", "naming" and "names" would be represented by one base form "name". For a noun, its singular form is produced as the base form. With this result, it is possible for an application to cover different word forms with one pattern.

6.1 Example

Here are examples of the GLARF analysis of two sentences.

Sentence (1):

He succeeds Robert L. Purdum, 58, who retired.

Parse tree:

```
(S1
 (S (NP (PRP He))
    (VP (VBZ succeeds)
      (NP (NP (NNP Robert) (NNP L.) (NNP Purdum))
      (, ,)
      (NP (CD 58))
      (, ,)
      (SBAR (WHNP (WP who)) (S (VP (VBD retired))))))
    (. .)))
```

GLARF output:

(SBJ succeed, He)

(OBJ succeed, Robert_L._Purdum)

(APPOSITE Robert_L._Purdum, 58)

(RELATIVE Robert_L._Purdum, who)

(SENT who, retired)

(SBJ retire, Robert_L._Purdum)

Sentence (2):

The deadlock in Palestinian-Israeli talks is exacerbated by the worse violence in four years.

Parse tree:

```
(S1
 (S (NP (NP (DT The) (NN deadlock))
        (PP (IN in)
         (NP (NNP Palestinian-Israeli) (NNS talks))))
    (VP (AUX is)
     (VP (VBN exacerbated)
      (PP (IN by)
       (NP (NP (DT the) (JJS worst) (NN violence))
           (PP (IN in) (NP (CD four) (NNS years)))))))
    (. .)))
```

GLARF output:

(T-POS deadlock, The)

(ADV deadlock, in)

(OBJ in, talks)

(N-POS talks, Palestinian-Israeli)

(SBJ exacerbated, violence)

(COMP exacerbated, by)

(OBJ exacerbated, deadlock)

(AUX1 exacerbated, is)

(T-POS violence, the)

(A-POS violence, worst)

(DGCOMP violence, in)

(OBJ in, years)

(T-POS years, four)

Chapter 7

Kernel Slot Filler Detection

7.1 Introduction

Traditional IE approaches try to generate patterns for events by various means using training data. For example, the FASTUS (Appelt et al., 1996)[6] and Proteus (Grishman, 1996)[22] systems, which performed well for MUC-6, used hand-written rules for event patterns. The symbolic learning systems, like AutoSlog (Riloff, 1993)[44] and CRYSTAL (Fisher et al., 1996)[21], generated patterns automatically from specific examples (text segments) using generalization and predefined pattern templates. There are also statistical approaches (Miller et al., 1998[42]; Collins et al., 1997[16]) trying to encode event patterns in statistical CFG grammars. All of these approaches assume events occur in text in certain patterns. However this assumption may not be completely correct and it limits the syntactic information considered by these approaches for finding events, such as information on global features from levels other than deep processing. When training data is limited, these approaches may also be less effective in their ability to generate reliable patterns.

Chapter 5 proposed a new approach to overcome these problems. It makes no prior assumption about the syntactic structure an event may assume; instead, it considers all syntactic features in the target text and uses a discriminative classifier to decide that automatically. Discriminative classifiers make no attempt to resolve the structure of the target classes. They only care about the decision boundary to separate the classes. In our case, we only need criteria to predict event elements from text using the syntactic features provided. This seems a more suitable solution for IE where training data is often sparse.

This chapter presents an implementation (ARES: Automated Recognition of Event Slots)[58] of the framework proposed in chapter 5 to find slot fillers for the MUC-6 events. The event domain is corporate management succession. We will develop kernels at two levels: word sequences and predicate-argument dependencies. Two tasks will be used for evaluation: detecting event occurrence in sentence and finding slot fillers of an event. We will show that a simple bag-of-words model can give us reliable information about event occurrence. When combining with deep level kernels, word n-grams information also helps.

The classifier we chose to use is Support Vector Machine, mostly due to its ability to work in high dimensional feature spaces. The experimental results of this approach show that it can outperform a hand-crafted rule system in slot filler detection in the management succession domain.

7.2 Related Work

There have been a number of SVM applications in NLP using particular levels of syntactic information. Lodhi et al. (2002)[36] compared a word-based string kernel and n-gram kernels at the sequence level for a text categorization task.

The experimental results showed that the n-gram kernels performed quite well for the task. Although string kernels can capture common word subsequences with gaps, its geometric penalty factor may not be suitable for weighting the long distance features. Collins et al. (2001)[14] suggested kernels on parse trees and other structures for general NLP tasks. These kernels count small subcomponents multiple times so that in practice one has to be careful to avoid overfitting. This can be achieved by limiting the matching depth or using a penalty factor to downweight large components.

Zelenko et al. (2003)[57] devised a kernel on shallow parse trees to detect relations between named entities, such as the person-affiliation relation between a person name and an organization name. The so-called relation kernel matches from the roots of two trees and continues recursively to the leaf nodes if the types of two nodes match.

All the kernels used in these works were applied to a particular syntactic level. This chapter presents an approach for information extraction that uses kernels to combine information from different levels and automatically identify which information contributes to the task. This framework can also be applied to other NLP tasks.

Chieu et al. (2003)[13] reported a feature-based SVM system (ALICE) to extract MUC-4 events of terrorist attacks. The Alice-ME system demonstrated performance competitive with rule-based systems. The features used by Alice are mainly from parsing. Comparing with ALICE, our system uses kernels on dependency graphs to replace explicit features, an approach which is fully automatic and requires no enumeration of features. The model we proposed can combine information from different syntactic levels in principled ways. In our experiments, we used both word sequence information and parsing level syntax information. The training data for ALICE contains 1700 documents, while for our system it is just 100 documents. When data is sparse, it is more difficult for an automatic system to outperform a rule-based system that incorporates general knowledge.

7.3 Event and Slot Kernels

Here we will introduce the kernels used by ARES for event occurrence detection (EOD) and slot filler detection (SFD).

7.3.1 Event Occurrence Detection Kernels

In Information Extraction, one interesting issue is event occurrence detection, which is determining whether a sentence contains an event occurrence or not. If this information is given, it would be much easier to find the relevant entities for an event from the current sentence or surrounding sentences. Traditional approaches do matching (for slot filling) on all sentences, even though most of them do not contain any event at all. Event occurrence detection is similar to sentence level information retrieval, so simple models like bag-of-words or n-grams could work well. We tried two kernels to do this, one is a sequence level n-gram kernel and the other is a GLARF-based kernel that matches syntactic details between sentences. In the following formula, we will use an identity function $I(x, y)$ that gives 1 when $x \equiv y$ and 0 otherwise, where x and y are strings or vectors of strings.

1. N-gram kernel

It counts common n-grams between two sentences. Given two sentences: $S_1 =< w_1, w_2, ..., w_n >$, and $S_2 =< w_1, w_2, ..., w_m >$, a bigram kernel

$$\psi_{bi}(S_1, S_2) = \sum_{i=1}^{n-1} \sum_{j=1}^{m-1} (I(w_i, w_j) + (I(< w_i, w_{i+1} >, < w_j, w_{j+1} >))$$

Here kernels are inclusive, so the bigram kernel includes both bigrams and unigrams. For the unigram kernel a stop list is used to remove words other than nouns, verbs, adjectives and adverbs.

2. Glarf kernel

   This kernel is based on the GLARF dependency result. Given the triple outputs of two sentences produced by GLARF: $G_1 = < r_i, p_i, a_i >$, $1 \leq i \leq N_1$ and $G_2 = < r_j, p_j, a_j >$, $1 \leq j \leq N_2$, where r_i, p_i, a_i correspond to the role label, predicate word and argument word respectively in GLARF output. This kernel matches the labels, predicate words and argument words of two triples respectively.

$$\psi_g(G_1, G_2) = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} (I(< r_i, p_i, a_i >, < r_j, p_j, a_j >) + \alpha I(p_i, p_j) + \beta I(a_i, a_j))$$

   The matches of predicate and argument words can give us partial score when two triples do not match exactly, which is often true given the sparse data. In EOD experiments, α and β were set to 1.

7.3.2 Slot Filler Detection Kernels

Slot filler detection (SFD) is the task of determining which named entities fill a slot in some event template. Two kernels were proposed for SFD: the first one matches local contexts of two target NEs, while the second one combines the first one with an n-gram EOD kernel.

1. $\psi^1_{SFD}(E_1, E_2)$: This kernel was also defined on a GLARF dependency graph (DG), a directed graph constructed from its typed predicate-argument outputs. The arcs labeled with roles go from predicate words to argument words. This kernel matches local context surrounding a name in a GLARF dependency graph. In preprocessing, all the names of the same type are translated into one symbol, such as *PER* for all person names. The matching starts from two example nodes (NE nodes of the same type) in the two DG's and recursively goes from these nodes to their successors and predecessors, until the words associated with nodes do not match. In our experiment, the matching depth was set to 2. Each node n contains a predicate word w and relation pairs $\{< r_i, a_i >\}$, $1 \leq i \leq p$ representing its p arguments and the roles associated with them.

   A matching function $C(n_1, n_2)$ on two nodes is defined as
   $$\sum_{i=1}^{p_1} \sum_{j=1}^{p_2} (I(< r_i, a_i >, < r_j, a_j >) + I(r_i, r_j))$$
   Then a recursive kernel $\psi(n_1, n_2)$ on two nodes is defined as
   $$C(n_1, n_2) + \sum_{\substack{k_i \in Succ(n_1) \\ k_j \in Succ(n_2)}}^{k_i \equiv k_j} \psi(k_i, k_j) + \sum_{\substack{k_i \in Pred(n_1) \\ k_j \in Pred(n_2)}}^{k_i \equiv k_j} \psi(k_i, k_j),$$
   where $k_i \equiv k_j$ is true if the predicate words associated with them match. Functions $Succ(n)$ and $Pred(n)$ give the successor or predecessor node set

of a node n. The computation of $\psi(n_1, n_2)$ is similar to a breadth-first graph traversal algorithm. Each node is visited only once. If a node is visited before, ψ simply returns 0.

The reason for setting a depth limit is that it covers most of the local syntax around a node. In most cases the match of $\psi(n_1, n_2), n_1 \neq n_2$ stops within two steps.

Then we define

$$\psi^1_{SFD}(E_1, E_2) = \psi(E_1, E_2),$$

where E_1 and E_2 are the entity nodes we are interested in two dependency graphs. The proof that ψ^1_{SFD} is a kernel is provided in appendix A.

2. $\psi^2_{SFD}(< S_1, E_1 >, < S_2, E_2 >)$: This kernel combines linearly the n-gram event kernel and the slot kernel above, in the hope that the general event occurrence information provided by the EOD kernel can help the slot kernel to ignore NEs in sentences that do not contain any event occurrence.

$$\psi^2_{SFD}(< S_1, E_1 >, < S_2, E_2 >) = \alpha\psi_{tri}(S_1, S_2) + \beta\psi^1_{SFD}(E_1, E_2),$$

where α, β were set to be 1 in our experiments. The Glarf event kernel was not used, simply because it uses information from the same source as $\psi^1_{SFD}(E_1, E_2)$. The n-gram kernel was chosen to be the trigram kernel, which gives us the best EOD performance among n-gram kernels. We also tried the dependency graph kernel proposed by Collins et al. (2001)[14], but it did not give us a better result.

7.4 Experiments

7.4.1 Corpus

The experiments with ARES were done on the MUC-6 corporate management succession domain using the official training data and, for the final experiment, the official test data as well. The training data was split into a training set (80%) and validation set (20%). In ARES, the text was preprocessed by the Proteus NE tagger and Charniak sentence parser. Then the GLARF processor produced dependency graphs based on the parse trees and NE results. All the names were transformed into symbols representing their types, such as *PERSON* for all person names. The reason is that we think the name itself does not provide a significant clue; the only thing that matters is what type of name occurs at a certain position.

Two tasks have been tried: one is EOD (event occurrence detection) on sentences; the other is SFD (slot filler detection) on named entities, including person names and job titles. EOD is to determine whether a sentence contains an event or not. This would give us general information about sentence level event occurrences. SFD is to find name fillers for event slots. The slots we experimented with were the person name and job title slots in MUC-6. We used the SVM package SVMlight in our experiments, embedding our own kernels as custom kernels.

7.4.2 EOD Experiments

In this experiment, ARES was trained on the official MUC-6 training data to do event occurrence detection. The data contains 1940 sentences, of which 158 are

Kernel	Precision	Recall	F-score
Unigram(ψ_{uni})	66.0%	66.5%	66.3%
Bigram(ψ_{bi})	73.9%	60.3%	66.4%
Trigram(ψ_{tri})	77.5%	61.5%	68.6%
GLARF(ψ_g)	77.5%	63.9%	70.1%
Mix($\psi_g + \psi_{tri}$)	81.5%	66.4%	73.2%

Table 7.1: EOD performance of ARES using different kernels

labeled as positive instances (contain an event). 5-fold cross validation was used so that the training and test set contain 80% and 20% of the data respectively. Three kernels defined in the previous section were tried. Table 7.1 shows the performance of each kernel. Three n-gram kernels were tested: unigram, bigram and trigram. Subsequences longer than trigrams were also tried, but did not yield better results.

The results show that the trigram kernel performed the best among n-gram kernels. GLARF kernel did better than n-gram kernels, which is reasonable because it incorporates detailed syntax of a sentence. But generally speaking, the n-gram kernels alone performed fairly well for this task, which indicates that low level text processing can also provide useful information. The mix kernel that combines the trigram kernel with GLARF kernel gave the best performance, which might indicate that the low level information provides additional clues or helps to overcome errors in deep processing.

7.4.3 SFD Experiments

The slot filler detection (SFD) task is to find the named entities in text that can fill the corresponding slots of an event. We treat job title as a named entity throughout this chapter, although it is not included in the traditional MUC named entity set. The slots we used for evaluation were PERSON_IN (the person who took a position), PERSON_OUT (the person who left a position) and POST (the position involved). We generated the two person slots from the official MUC-6 templates and the corresponding filler strings in text were labeled. Three SVM predictors were trained to find name fillers of each slot. Two experiments have been tried on MUC-6 training data using 5-fold cross validation.

The first experiment of ARES used the slot kernel alone, relying solely on local context around a NE. From the performance table (Table 7.2), we can see that local context can give a fairly good clue for finding PERSON_IN and POST, but not for PERSON_OUT. The main reason is that local context might be not enough to determine a PERSON_OUT filler. It often requires inference or other semantic information. For example, the sentence "Aaron Spelling, the company's vice president, was named president.", indicates that "Aaron Spelling" left the position of vice president, therefore it should be a PERSON_OUT. But the sentence "Aaron Spelling, the company's vice president, said ", which is very similar to first one in syntax, has no such indication at all. In complicated cases, a person can even hold two positions at the same time.

In this experiment, the SVM predictor considered all the names identified by the NE tagger; however, most of the sentences do not contain an event occurrence at all, so NEs in these sentences should be ignored no matter what

Accuracy	Precision	Recall	F-score
PER_IN	63.6%	62.5%	63.1%
PER_OUT	54.8%	54.2%	54.5%
POST	64.4%	55.2%	59.4%

Table 7.2: SFD performance of ARES with $\psi^1_{SFD}(E_1, E_2)$

their local context is. To achieve this we need general information about event occurrence, and this is just what the EOD kernel can provide. In our second experiment, we tested the kernel $\psi^2_{SFD}(< S_1, E_1 >, < S_2, E_2 >)$, which is a linear combination of the trigram EOD kernel and the SFD kernel. Table 7.3 shows the performance of the combination kernel, from which we can see that there is clear performance improvement for all three slots. For PER_OUT, the unigram EOD kernel was used. This is because the filling of many PER_OUT slots requires inference using world knowledge, such as when a person is promoted, he leaves his old position. In this case, the syntax around names is not very useful. The unigram EOD kernel was chosen by validation. It seemed to work better than the trigram kernel on PER_OUT.

In this experiment, we also tried to use the mix EOD kernel which gives the best performance, but it did not yield a better result. The reason we think is that the GLARF EOD kernel and SFD kernel are from the same syntactic source, so the information was repeated.

Since 5-fold cross validation was used, ARES was trained on 80% of the MUC-6 training data in these two experiments.

Accuracy	Precision	Recall	F-score
PER_IN	86.6%	60.5%	71.2%
PER_OUT	69.2%	58.2%	63.2%
POST	68.5%	68.9%	68.7%

Table 7.3: SFD performance of ARES with $\psi^2_{SFD}(< S_1, E_1 >, < S_2, E_2 >)$

Accuracy	Precision	Recall	F-score
PER_IN	77.3%	62.2%	68.9%
PER_OUT	58.9%	69.7%	63.9%
POST	77.1%	71.5%	73.6%

Table 7.4: Slot performance of ARES on the MUC-6 test data

7.4.4 Comparison with MUC-6 System

This experiment was done on the official MUC-6 training and test data, which contain 50K words and 40K words respectively. ARES used the official corpora as training and test sets. In the training data, all the slot fillers were manually labeled. We compared the performance of ARES with the NYU Proteus system, a rule-based system that performed well for MUC-6. To score the performance for these three slots, we generated the slot-filler pairs as keys for a document from the official MUC-6 templates and removed duplicate pairs. The scorer matches the filler string in the response file of ARES to the keys. The response result for Proteus was extracted in the same way from its template output. Table 7.4. shows the result of ARES using the combination kernel in the previous experiment.

Table 5 shows the test result of the Proteus system. Comparing the num-

Accuracy	Precision	Recall	F-score
PER_IN	85.7%	51.2%	64.1%
PER_OUT	78.4%	58.6%	67.1%
POST	83.3%	59.7%	69.5%

Table 7.5: Slot performance of the rule-based Proteus system for MUC-6

bers we can see that for slot PERSON_IN and POST, ARES outperformed the Proteus system by a few points. The result is promising considering that this model is fully automatic and does not involve any post-processing. As for the PERSON_OUT slot, the performance of ARES was not as good. As we have discussed before, relying purely on syntax might not help us much; we may need an inference model to resolve this problem.

7.5 Examples

In this section we will show some examples where adding the EOD trigram kernel helped the local SFD kernel. The sentence of an example contrains preprocessed tokens where all the automatically recognized NEs were transformed into special symbols (their NE type). Strings in parenthesis indicate the original names. For example in **PER*(Gary)*, **PER** is the token seen by the classifier and *Gary* is the original name in text.

Example (1):

Sentence (token sequence):

> *Dollar Time said *PER*(Gary) de Luca, 40 years old, the company 's *POST*(chief_operating_officer), was named *POST*(president).*

GLARF output:

(SBJ, said, Dollar Time)

(COMP, said, named)

(RED-RELATIVE, Luca, old)

(APPOSITE, Luca, *POST*)

(N-POS, Luca, *PER*)

(SBJ, old, Luca)

(T-POS, years, 40)

(T-POS, *POST*, company)

(T-POS, company, the)

(SUFFIX, company, 's)

(COMP, named, *POST*)

(OBJ, named, Luca)

(AUX, named, was)

The token sequence of this example contains errors from the NE tagger. *de Luca* was not recognized as part of the person name and *Dollar Time* was not recognized as an organization name. The target here was to classify **PER** as a positive PERSON_IN.

The prediction of kernel $\psi^1_{SFD}(E_1, E_2)$ on **PER** is -0.99906068, which is negative. Due to the NE error, GLARF failed to produce dependencies between **PER*(Gary)* and *was named *POST**. But the prediction of kernel ψ^2_{SFD} is 0.35327107, which is correct.

Intuitively the token sequence *was named *POST** helped kernel ψ^2_{SFD} to make the correct prediction because it occurs a lot in training data in the context of positive examples. After replacing the word *named* with a dummy word and running the classifier again with the modified sentence, the prediction of kernel

ψ^2_{SFD} turned to be -0.29386523. So the long distance information from kernel $\psi_{tri}(S_1, S_2)$ helped the classifier to get the right answer.

Example (2):

Sentence (token sequence):

*Mr. *PER*(Walsh), 51, who remains *POST*(chairman), will be on indefinite medical leave.*

GLARF output:

(RELATIVE *PER*, who)

(APPOSITE *PER*, 51)

(SBJ be, *PER*)

(SBJ *POST*, *PER*)

(PRD remains, *POST*)

(SENT who, remains)

(AUX be, will)

(OBJ on, leave)

(ADV be, on)

(A-POS leave, medical)

(A-POS leave, indefinite)

The target of this example was to classify **PER*(Walsh)* as a negative PERSON_IN. The prediction of kernel $\psi^1_{SFD}(E_1, E_2)$ on **PER** is 0.68899627, which is wrong. Actually if a sentence contains a similar relative clause, the target is often classified wrong by kernel $\psi^1_{SFD}(E_1, E_2)$. The reason might be that this kind of structure often occurs in positive examples in training data.

The prediction of kernel $\psi^2_{SFD}(< S_1, E_1 >, < S_2, E_2 >)$ is -0.22528701, which is correct. So the global information of words avoided the false alarm. To see which words are critical, we first replaced the words *indefinite, medical* and *leave* with dummy words, then the prediction of ψ^2_{SFD} became -0.01907587, which is still correct, but with much less confidence. By replacing words *remains, indefinite, medical* and *leave*, the prediction is 0.050356181, which is wrong. The analysis shows that these words were associated with negative predictions. They came into play in the composite kernel to make the prediction correct. (There might also be other words that made similar contributions.)

7.6 Discussion and Further Work

This chapter describes an implementation of the discriminative approach proposed in this thesis is to find slot fillers for MUC-6 events. In our experiment, it outperformed a hand-crafted system on sparse data by combining different levels of syntactic clues. The result shows that low level syntactic information can also come into play in finding event occurrences or slot fillers. So it should not be ignored by IE tasks.

For slot filler detection, several classifiers were trained to find names for each slot and these classifiers make independent decisions. However, entity slots in events are often strongly correlated, for example the PER_IN and POST slots for management succession events. They often appear in pairs in a sentence. Since these classifiers take the same input and produce different results, correlation models can be used to integrate these classifiers so that the identification of slot fillers might benefit each other.

It is also possible to experiment with the tasks that are more difficult for pattern matching, such as determining the *On-the-job* status property in the management succession domain or the *Stage-of-execution* property for the terrorist attack domain. For these "implicit" slots, it is difficult to pick up a specific syntax pattern as a reliable indication. For example a passive form verb "bombed" could imply the attack was successful. It would be interesting to try this kernel approach with multiple levels of information to see what kind of clues the classifier picks up.

Since events often span multiple sentences, another direction is to explore cross-sentence models, which is also difficult for traditional approaches. For our approach the division into sentences is not a critical factor. It is possible to extend the kernel from one sentence to multiple sentences. Probably we can explore the correlation between NE's in adjacent sentences.

It is also possible to run each classifier on a different range of text. For some slots we can expect to rely on local syntactic clues, such as for the PERSON_IN slot in MUC-6; while for others, such as the *Stage-of-execution* slot, we may like to use features from a whole paragraph.

Chapter 8

Entity Relation Detection

8.1 Introduction

Information extraction subsumes a broad range of tasks, including the extraction of entities, relations and events from various text sources, such as newswire documents and broadcast transcripts. One such task, relation detection, finds instances of predefined relations between pairs of entities, such as a *Located-In* relation between the entities *Centre College* and *Danville, KY* in the phrase *Centre College in Danville, KY*. The "entities" are the individuals of selected semantic types (such as people, organizations, countries) which are referred to in the text.

Prior approaches to this task (Miller et al., 2000[43]; Zelenko et al., 2003[57]) have relied on partial or full syntactic analysis. Syntactic analysis can find relations not readily identified based on sequences of tokens alone. Even "deeper" representations, such as logical syntactic relations or predicate-argument structure, can in principle capture additional generalizations and thus lead to the identification of additional instances of relations. However, a general problem in Natural Language Processing is that as the processing gets deeper, it becomes less accurate. Algorithms based solely on deeper representations inevitably suffer from the errors in computing these representations. On the other hand, low level processing such as tokenization will be more accurate, and may also contain useful information missed by deep processing of text. Systems based on a single level of representation are forced to choose between shallower representations, which will have fewer errors, and deeper representations, which may be more general.

Based on these observations, we proposed a discriminative model in chapter 5 to combine information from different syntactic sources using a kernel SVM. Chapter 7 showed that adding sentence level word trigrams as global information to local dependency context boosted the performance of finding slot fillers for management succession events. This chapter describes another implementation of this approach to identify entity relations.

In this experiment syntactic information from sentence tokenization, parsing and deep dependency analysis is combined using kernel methods. At each level, kernel functions (or kernels) are developed to represent the syntactic information. Five kernels have been developed for this task, including two at the surface level, one at the parsing level and two at a deep dependency level.

Our experiments show that each level of processing may contribute useful clues for this task, including surface information like word bigrams. Adding kernels one by one continuously improves performance. The experiments were carried out on the ACE RDR (Relation Detection and Recognition) task with hand-annotated entities. Using SVM as a classifier along with the full composite kernel produced the best performance on this task. This chapter will also show a comparison of SVM and KNN (k Nearest Neighbors) under different kernel setups.

8.2 Prior Work

Collins et al. (1997)[16] and Miller et al. (2000)[43] used statistical parsing models to extract relational facts from text. Both of these models rely on a single level of linguistic analysis, and so are vulnerable to errors in this level. Collins et al. (1997)[16] addressed a simplified task within a confined context in a target sentence.

Zelenko et al. (2003)[57] described a recursive kernel based on shallow parse trees to detect *person-affiliation* and *organization-location* relations, in which a relation example is the least common subtree containing two entity nodes. The kernel matches nodes starting from the roots of two subtrees and going recursively to the leaves. For each pair of nodes, a subsequence kernel on their child nodes is invoked, which matches either contiguous or non-contiguous subsequences of node. Compared with full parsing, shallow parsing is more reliable. But this model is based solely on the output of shallow parsing so it is still vulnerable to irrecoverable parsing errors. In their experiments, incorrectly parsed sentences were eliminated.

Culotta and Sorensen (2004)[20] described a slightly generalized version of this kernel based on dependency trees. Since their kernel is a recursive match from the root of a dependency tree down to the leaves where the entity nodes reside, a successful match of two relation examples requires their entity nodes to be at the same depth of the tree. This is a strong constraint on the matching of syntax so it is not surprising that the model has good precision but very low recall. In their solution a bag-of-words kernel was used to compensate for this problem. In our approach, more flexible kernels are used to capture regularization in syntax, and more levels of syntactic information are considered.

Kambhatla (2004)[32] described a Maximum Entropy model using features from various syntactic sources, but the number of features they used is limited and the selection of features has to be a manual process[1]. In our model, we use kernels to incorporate more syntactic information and let a Support Vector Machine decide which clue is crucial. Some of the kernels are extended to generate high order features. We think a discriminative classifier trained with all the available syntactic features should do better on the sparse data.

8.3 ACE RDR Task

ACE (Automatic Content Extraction) is a research and development program in information extraction sponsored by the U.S. Government. The 2004 evaluation defined seven major types of relations between seven types of entities. The entity types are PER (Person), ORG (Organization), FAC (Facility), GPE (Geo-Political Entity: countries, cities, etc.), LOC (Location), WEA (Weapon) and VEH (Vehicle). Each mention of an entity has a mention type: NAM (proper name), NOM (nominal) or PRO (pronoun); for example *George W. Bush*, *the president* and *he* respectively. The seven relation types are EMP-ORG (Employment/Membership/Subsidiary), PHYS (Physical), PER-SOC (Personal/Social), GPE-AFF (GPE-Affiliation), Other-AFF

[1]Kambhatla also evaluated his system on the ACE relation detection task, but the results are reported for the 2003 task, which used different relations and different training and test data, and did not use hand-annotated entities, so they cannot be readily compared to our results.

Frequency	Type	Example
1631	EMP-ORG	the *CEO* of *Microsoft*
1216	PHYS	a military *base* in *Germany*
529	GPE-AFF	*U.S. businessman*
365	PER-SOC	a *spokesman* for the *senator*
279	DISC	*many* of these *people*
212	ART	the *makers* of the *Kursk*
142	Other-AFF	*Cuban-American people*

Table 8.1: ACE relation types and examples. The heads of two entity arguments in a relation are marked. Types are listed in decreasing order of frequency of occurrence in the ACE corpus.

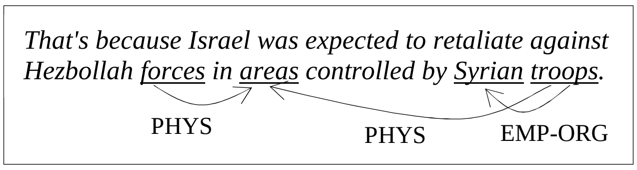
That's because Israel was expected to retaliate against Hezbollah forces in areas controlled by Syrian troops.
PHYS
PHYS
EMP-ORG

Figure 8.1: Example sentence from nwire text with three relations

(Person/ORG Affiliation), ART (Agent-Artifact) and DISC (Discourse). There are also 27 relation subtypes defined by ACE, but here we only focus on detection of relation types. Table 8.1 lists examples of each relation type.

Figure 8.1 shows a sample newswire sentence, in which three relations are marked.

In this sentence, we expect to find a PHYS relation between Hezbollah forces and areas, a PHYS relation between Syrian troops and areas and an EMP-ORG relation between Syrian troops and Syrian. In our approach, input text is preprocessed by the Charniak sentence parser (including tokenization and POS tagging) and the GLARF (Meyers et al., 2001)[39] dependency analyzer.

8.4 Kernels

In this section, the kernels used for the RDR task will be formally defined.

8.4.1 Definitions

In our model, kernels incorporate information from tokenization, parsing and deep dependency analysis. A relation candidate R is defined as

$$R = (arg_1, arg_2, seq, link, path),$$

where arg_1 and arg_2 are the two entity arguments which may be related; $seq = (t_1, t_2, ..., t_n)$ is a token vector that covers the arguments and intervening words; $link = (t_1, t_2, ..., t_m)$ is also a token vector, generated from seq and the parse tree; $path$ is a dependency path connecting arg_1 and arg_2 in the dependency graph produced by GLARF. $path$ can be empty if no such dependency path exists. The difference between $link$ and seq is that $link$ only retains the "important" words in seq in terms of syntax. For example, all noun phrases occurring in seq are replaced by their heads. Words and constituent types in a stop list, such as time expressions, are also removed. A token T is defined as a string triple,

$$T = (word, pos, base),$$

where $word$, pos and $base$ are strings representing the word, part-of-speech and morphological base form of T. Entity is a token augmented with other attributes,

$$E = (tk, type, subtype, mtype),$$

where tk is the token associated with E; $type$, $subtype$ and $mtype$ are strings representing the entity type, subtype and mention type of E. The subtype contains more specific information about an entity. For example, for a GPE entity, the subtype tells whether it is a country name, city name and so on. Mention type includes NAM, NOM and PRO.

It is worth pointing out that we always treat an entity as a single token: for a nominal, it refers to its head, such as *boys* in *the two boys*; for a proper name, all the words are connected into one token, such as *Bashar_Assad*. So in a relation example R whose seq is $(t_1, t_2, ..., t_n)$, it is always true that $arg_1 = t_1$ and $arg_2 = t_n$. For names, the base form of an entity is its ACE type(person, organization, etc.). To introduce dependencies, we define a dependency token to be a token augmented with a vector of dependency arcs,

$$DT = (word, pos, base, dseq),$$

where $dseq = (arc_1, ..., arc_n)$. A dependency arc is

$$ARC = (w, dw, label, e),$$

where w is the current token; dw is a token connected by a dependency to w; and $label$ and e are the role label and direction of this dependency arc respectively. From now on we upgrade the type of tk in arg_1 and arg_2 to be dependency tokens. Finally, $path$ is a vector of dependency arcs,

$$path = (arc_1, ..., arc_l),$$

where l is the length of the path and arc_i $(1 \leq i \leq l)$ satisfies $arc_1.w = arg_1.tk$, $arc_{i+1}.w = arc_i.dw$ and $arc_l.dw = arg_2.tk$. So path is a chain of dependencies connecting the two arguments in R.

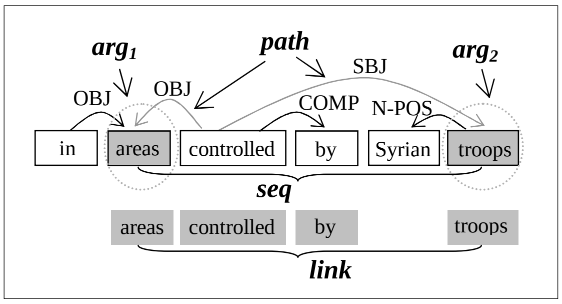

Figure 8.2: Illustration of a relation example R. The link sequence is generated from seq by removing some unimportant words based on syntax. The dependency links are generated by GLARF.

Figure 8.2 shows a relation example generated from the text " in areas controlled by Syrian troops". In this relation example R, arg_1 is ((“areas”, “NNS”, “area”, dseq), “LOC”, “Region”, “NOM”), and $arg1.dseq$ is ((OBJ, areas, in, 1), (OBJ, areas, controlled, 1)). arg_2 is ((“troops”, “NNS”, “troop”), “ORG”, “Government”, “NOM”) and $arg_2.dseq$ = ((A-POS, troops, Syrian, 0), (SBJ, troops, controlled, 1)). *path* is ((OBJ, areas, controlled, 1), (SBJ, controlled, troops, 0)). The value 0 in a dependency arc indicates forward direction from w to dw, and 1 indicates backward direction. The *seq* and *link* sequences of R are shown in Figure 8.2.

Some relations occur only between very restricted types of entities, but this is not true for every type of relation. For example, PER-SOC is a relation mainly between two person entities, while PHYS can happen between any type of entity and a GPE or LOC entity.

8.4.2 Syntactic Kernels

In this section we will describe the kernels designed for different syntactic sources and explain the intuition behind them.

We define two kernels to match relation examples at surface level. Using the notation just defined, we can write the two surface kernels as follows:

1. Argument

   $$\psi_1(R_1, R_2) = \sum_{i=1,2} K_E(R_1.arg_i, R_2.arg_i),$$

   where K_E is a kernel that matches two entities,

   $$\begin{aligned} K_E(E_1, E_2) =& K_T(E_1.tk, E_2.tk) + I(E_1.type, E_2.type) + \\ & I(E_1.subtype, E_2.subtype) + I(E_1.mtype, E_2.mtype) \end{aligned}$$

   and

   $$K_T(T_1, T_2) = I(T_1.word, T_2.word) + I(T_1.pos, T_2.pos) + I(T_1.base, T_2.base)$$

   K_T is a kernel that matches two tokens. $I(x, y)$ is a binary string match operator that gives 1 if $x = y$ and 0 otherwise. Kernel ψ_1 matches attributes of two entity arguments respectively, such as type, subtype and lexical head of an entity. This is based on the observation that there are type constraints on the two arguments. For instance PER-SOC is a relation mostly between two person entities. So the attributes of the entities are crucial clues. Lexical information is also important to distinguish relation types. For instance, in the phrase *U.S. president* there is an EMP-ORG relation between *president* and *U.S.*, while in *a U.S. businessman* there is a GPE-AFF relation between *businessman* and *U.S.*

2. Bigram kernel

$$\psi_2(R_1, R_2) = K_{seq}(R_1.seq, R_2.seq),$$

where

$$K_{seq}(seq, seq') = \sum_{0 \le i < seq.len} \sum_{0 \le j < seq'.len} (K_T(tk_i, tk'_j) + K_T(< tk_i, tk_{i+1} >, < tk'_j, tk'_{j+1} >))$$

Operator $< t_1, t_2 >$ concatenates strings in token t_1 and t_2 to produce a new token. So ψ_2 is a kernel that simply matches unigrams and bigrams between the *seq* sequences of two relation examples. The information this kernel provides is faithful to the text.

3. Link sequence kernel

$$\begin{aligned}\psi_3(R_1, R_2) &= K_{link}(R_1.link, R_2.link) \\ &= \sum_{0 \le i < min\_len} K_T(R_1.link.tk_i, R_2.link.tk_i),\end{aligned}$$

where *min_len* is the length of the shorter link sequence in R_1 and R_2. ψ_3 is a kernel that matches token by token between the link sequences of two relation examples. Since relations often occur in a short context, we expect many of them have similar link sequences.

4. Dependency path kernel

$$\psi_4(R_1, R_2) = K_{path}(R_1.path, R_2.path),$$

where

$$\begin{aligned}K_{path}(path, path') = \sum_{0 \le i < path.len} \sum_{0 \le j < path'.len} &(I(arc_i.label, arc'_j.label) + \\ &K_T(arc_i.dw, arc'_j.dw)) \times I(arc_i.e, arc'_j.e)\end{aligned}$$

Intuitively the dependency path connecting two arguments could provide a high level of syntactic regularization. However, a complete match of two dependency paths is infrequent. So this kernel matches the component arcs in two dependency paths in a pairwise fashion. Two arcs can match only when they are in the same direction. In cases where two paths do not match exactly, this kernel can still tell us how similar they are. In our experiments we placed an upper bound on the length of dependency paths for which we computed a nonzero kernel.

5. Local dependency

$$\psi_5(R_1, R_2) = \sum_{i=1,2} K_D(R_1.arg_i.dseq, R_2.arg_i.dseq),$$

where

$$K_D(dseq, dseq') = \sum_{0 \le i < dseq.len} \sum_{0 \le j < dseq'.len} (I(arc_i.label, arc'_j.label) + K_T(arc_i.dw, arc'_j.dw)) \times I(arc_i.e, arc'_j.e)$$

This kernel matches the local dependency context around the relation arguments. This can be helpful especially when the dependency path between arguments does not exist. We also hope the dependencies on each argument may provide some useful clues about the entity or connection of the entity to the context outside of the relation example.

8.4.3 Composite Kernels

Having defined all the kernels representing shallow and deep processing results, we can define composite kernels to combine and extend the individual kernels.

1. Polynomial extension

$$\Phi_1(R_1, R_2) = (\psi_1 + \psi_2) + (\psi_1 + \psi_3)^2/4$$

   This kernel combines the argument kernel ψ_1 and link kernel ψ_3 and applies a degree 2 polynomial kernel to extend them. This is equivalent to adding pairs of features as new features. Intuitively this introduces new features like: the subtype of the first argument is a country name and the word of the second argument is president, which could be a good clue for an EMP-ORG relation. The polynomial kernel is down weighted by a normalization factor because we do not want the high order features to overwhelm the original ones. In our experiment, using polynomial kernels with degree higher than 2 did not produce better results.

2. Full kernel

$$\Phi_2(R_1, R_2) = \Phi_1 + \alpha\psi_4 + \beta\psi_5 + \gamma\psi_2$$

   This is the final kernel we used for this task, which is a combination of all the previous kernels. In our experiments, we set all the scalar factors to 1. Different values were tried, but keeping the original weight for each kernel yielded the best results for this task.

All the individual kernels we designed are explicit. Each kernel can be seen as a matching of features and these features are enumerable on the given data. So it is clear that they are all valid kernels. Since the kernel function set is closed under linear combination and polynomial extension, the composite kernels are also valid. The reason we propose to use a feature-based kernel is that we can have a clear idea of what syntactic clues it represents and what kind of information it misses. This is important when developing or refining kernels, so that we can make them generate complementary information from different syntactic processing results.

8.5 Experiments

Experiments were carried out on the ACE RDR (Relation Detection and Recognition) task using hand-annotated entities, provided as part of the ACE evaluation. The ACE corpora contain documents from two sources: newswire (nwire) documents and broadcast news transcripts (bnews). In this section we will compare performance of different kernel setups trained with SVM, as well as different classifiers including KNN and SVM with the same kernel setup. The SVM package we used is SVM^{light}. The training parameters were chosen using cross-validation. One-against-all classification was applied to each pair of entities in a sentence. When SVM predictions conflict on a relation example, the one with larger margin will be selected as final answer.

8.5.1 Corpus

The ACE RDR training data contains 348 documents, 125K words and 4400 relations. It consists of both nwire and bnews documents. Evaluation of kernels was done on the training data using 5-fold cross-validation. We also evaluated the full kernel setup with SVM on the official test data, which is about half the size of the training data. All the data is preprocessed by the Charniak parser and GLARF dependency analyzer. Then relation examples are generated based these results.

	Kernel	Precision	Recall	F-score
A	argument (ψ_1)	52.96%	58.47%	55.58%
B	A + link ($\psi_1 + \psi_3$)	58.77%	71.25%	64.41%*
C	B-poly (Φ_1)	66.98%	70.33%	68.61%*
D	C + dep ($\Phi_1 + \psi_4 + \psi_5$)	69.10%	71.41%	70.23%*
E	D + bigram (Φ_2)	69.23%	70.50%	70.35%
F	argument+dep ($\psi_1 + \psi_4 + \psi_5$)	57.86%	68.50%	62.73%

Table 8.2: SVM performance on incremental kernel setups. Each setup adds one level of kernel(s) to the previous one except setup F. Performance was evaluated on the ACE training data with 5-fold cross-validation. F-scores marked by * are significantly better than the previous setup (at 95% confidence level)

8.5.2 Results

Table 8.2 shows the performance of the SVM on different kernel setups. The kernel setups in this experiment are incremental. From this table we can see that adding kernels continuously improves the performance, which indicates they provide additional clues to the previous setup. The argument kernel treats the two arguments as independent entities. The link sequence kernel introduces the syntactic connection between arguments, so adding it to the argument kernel boosted the performance. Setup F shows the performance of adding only dependency kernels to the argument kernel. The performance is not as good as setup B, indicating that dependency information alone is not as crucial as the link sequence.

Another observation is that adding the bigram kernel in the presence of all other levels of kernels improved both precision and recall, indicating that it helped in both correcting errors in other processing results and providing supplementary information missed by other levels of analysis. In another experiment evaluated on the nwire data only (about half of the training data), adding the bigram kernel improved F-score 0.5% and this improvement is statistically significant.

Table 8.3 shows the performance of SVM and KNN (k Nearest Neighbors) on different kernel setups. For KNN, k was set to 3. In the first setup of KNN, the two kernels which seem to contain most of the important information are used. It performs quite well when compared with the SVM result. The other two tests are based on the full kernel setup. For the two KNN experiments, adding more kernels (features) does not help. The reason might be that all kernels (features) were weighted equally in the composite kernel Φ_2 and this may not be optimal for KNN. Another reason is that the polynomial extension of kernels does not have any benefit in KNN because it is a monotonic transformation of similarity values. So the results of KNN on kernel $(\psi_1 + \psi_3)$ and Φ_1 would be exactly the same. We also tried different k for KNN and $k = 3$ seems to be the best choice in either case.

For the four major types of relations SVM does better than KNN, probably due to SVM's generalization ability in the presence of large number of features. For the last three types with many fewer examples, performance of SVM is not as good as KNN. The reason we think is that training of SVM on these types is not sufficient. We tried different training parameters for the types with fewer examples, but no dramatic improvement obtained.

We also evaluated our approach on the official ACE RDR test data and obtained very competitive scores. The primary scoring metric [2] for the ACE

[2]http://www.nist.gov/speech/tests/ace/ace04/software.htm

Type	KNN ($\psi_1 + \psi_3$)	KNN (Φ_2)	SVM($\psi_1 + \psi_3$)	SVM(Φ_1)	SVM (Φ_2)
EMP-ORG	75.43%	72.66%	70.12%	75.99%	77.76%
PHYS	62.19 %	61.97%	62.36%	65.08%	66.37%
GPE-AFF	58.67%	56.22%	54.50%	59.90%	62.13%
PER-SOC	65.11%	65.61%	69.44%	71.60%	73.46%
DISC	68.20%	62.91%	58.06%	61.44%	66.24%
ART	69.59%	68.65%	66.51%	68.35%	67.68%
Other-AFF	51.05%	55.20%	43.06%	45.89%	46.55%
Average	67.44%	65.69%	64.41%	68.61%	70.35%

Table 8.3: Performance of SVM and KNN (k=3) on different kernel setups. Types are ordered in decreasing order of frequency of occurrence in the ACE corpus. For SVM, the same training parameters were used for all 7 types.

evaluation is a 'value' score, which is computed by deducting from 100 a penalty for each missing and spurious relation; the penalty depends on the types of the arguments to the relation. The value scores produced by the ACE scorer for nwire and bnews test data are 71.7 and 68.0 respectively. The value score on all data is 70.1. The scorer also reports an F-score based on full or partial match of relations to the keys. The unweighted F-score for this test produced by the ACE scorer on all data is 76.0%. For this evaluation we used nearest neighbor to determine argument ordering and relation subtypes.

The classification scheme in our experiments is one-against-all. It turned out there is not so much confusion between relation types. The confusion matrix of predictions is fairly clean. We also tried pairwise classification, and it did not help much.

8.5.3 Examples

One interesting point in the experiment results is that the bigram kernel helped in the presence of all other kernels. Here we will show some examples to see the difference. Here Φ_2 represents the full composite kernel and Φ_{-bi} represents $\Phi_2 - \psi_2$.

Example (1):

Sentence (token sequence):

<u>She</u> is survived by <u>two sons</u>.

Target: PER-SOC

The relation example $R = \{$

arg_1=((“She”, “PRO”, “she”), “PER”, “”, “PRO”)

$arg1.dseq$ = ((OBJ survived She))

arg_2 = ((“sons”, “NNS”, “son”), “PER”, “”, “NOM”)

$arg_2.dseq$ = ((SBJ survived sons), (T-POS sons two))

$link$ = (She is survived by sons)

$path$ = ((OBJ She survived) (SBJ survived sons))

$\}$

For this example, Φ_{-bi} gives a wrong negative prediction, but the prediction of Φ_2 is positive. The relation here is implied by the context. It is very difficult to extract this relation by syntax. However, it is likely that the words “She” and “two sons” from the bigram kernel (including unigrams) pushed the prediction value to be positive.

Example (2):

Sentence (token sequence):

His office, clients and nearly everyone else he knows use America Online's messaging system.

Target: No-Relation

The relation example $R = \{$

arg_1=((“His”, “PRO”, “he”), “PER”, “”, “PRO”)

$arg1.dseq$ = ((T-POS office His), (T-POS clients His), (T-POS everyone His))

arg_2 = ((“everyone”, “NN”, “everyone”), “PER”, “”, “NOM”)

$arg_2.dseq$ = ((CONJ and everyone), (SBJ knows everyone), (ADV everyone nearly), (ADV everyone else), (T-POS everyone His))

$link$ = (office , clients and nearly)

$path$ = ((T-POS His everyone))

$\}$

For this example, the sentence parser failed to produce the correct structure of the sentence. It recognized *His office, clients and nearly everyone else* as a noun phrase. The parse tree was not well-formed. Therefore GLARF produced wrong dependencies like (T-POS His everyone) and (SBJ knows everyone).

The prediction of Φ_{-bi} is positive for PER-SOC, which is wrong. The prediction of Φ_2 is negative, which is correct. For this example, the type of *everyone* is PER (PERSON) and the dependency context is (T-POS His everyone). This context is very similar to cases like *His wife* or *His lawyer*, which contain PER-SOC relation. So the prediction using syntax is positive as PER-SOC. However, the word information from the bigram kernel pushed the prediction to be negative and disqualified this prediction.

It is worth to point out that Φ_{-bi} predicted no PER-SOC relation exists for example "*His* *office*", which is correct. The difference from the example above is that here *office* is of type "ORG". So there is no confusion in this case.

Both Φ_{-bi} and Φ_2 predicted PER-SOC relation in example "*His office, clients*". Even though this relation is not annotated in the key file, it seems to be a valid prediction.

8.6 Discussion

In this chapter, we have shown that using kernels to combine information from different syntactic sources performed well on the entity relation detection task. Our experiments show that each level of syntactic processing contains useful information for the task. Combining them may provide complementary information to overcome errors arising from linguistic analysis. Especially, low level information obtained with high reliability helped with the other deep processing results.

The design feature of our approach should be best employed when the preprocessing errors at each level are independent, namely when there is no dependency between the preprocessing modules. The model was tested on text with annotated entities, but its design is generic. It can work with noisy entity detection input from an automatic tagger. With all the existing information from other processing levels, this model can also be expected to recover from errors in entity tagging.

8.7 Further Work

1) In the ACE task, all mentions are classified into entity chains according to their coreference relations. So mentions “George W. Bush”, “the U.S. president” and “He” should be classified as one entity. For the relation detection task, the same relation may occur more than once in a document with different context. Combining the detected relations at different places can help us to correct the conflicts or retrieve the missed relations.

Relations reflect real world knowledge and some types of relations are persistent within a corpus, such as the presidency of “George W. Bush”. It is also possible to develop a model to do cross-document validation of relation findings, which should give a better coverage of relation occurrences.

Recognizing mentions in text has been shown to be a more difficult problem than recognizing named entities, mostly because there is more ambiguity associated with mentions. For example the nominal mention “the one” can refer to a wide range of entities.

This model can be easily extended to use an automatic mention recognition result. We can generate relation examples from both recognized mentions and mention candidates. Then we can run the relation detection algorithm to produce relations. By examining relation arguments, we may be able to correct some of the mention recognition errors.

Chapter 9

Further Development

9.1 Conclusion

This thesis describes a discriminative approach that combines syntactic clues automatically to do IE tasks. Each level of information is represented by a kernel and kernels from all levels can be combined into composite kernels in a principled way. This approach was tried on slot filler detection and entity relation recognition, both of which produced state-of-the-art performance.

With the properties of kernels, features from different syntactic levels can come into play naturally. The experimental results show that word sequence and shallow syntactic information are also helpful for IE tasks. The reliable information they provide can help to recover from errors in deep analysis of text. In slot filler detection, word ngrams also provide long distance information outside of the local context.

In the current framework, it is also possible to extend the sentence model to a multi-sentence model. Since an event usually spans multiple sentences, information in adjacent sentences is often helpful. There has been some attempt[42] to model such cross-sentence information. But for most prior approaches, this kind of information is not readily useable. In our framework, we can simply extend the kernels to cover a larger context.

9.2 The Learning Paradigm

Kernel functions have many nice properties. In the work described in this thesis, only linear combinations and polynomial extensions of kernels have been tried. There might be other ways to integrate kernels for information extraction tasks.

In the current setup, multiple classifiers are used to identify each category of target and these classifiers are independent. However, many of these categories are strongly correlated, for example the "PERSON_IN" and "POST_IN". They can be expected to occur in pairs in a sentence. So the prediction of one classifier should be a strong clue for the other. There are also exclusive relations: once a job position is taken by one person, it should not be taken by another person.

In a broader view of learning, we want to see (or use) models that take structured input and produce structured output so that the correlation in mapping structures is employed automatically (in a principled way) by the model. Such models may include conditional random fields[35] or other hybrid models[29].

9.3 Text Analysis

For most IE tasks, training data is sparse. Simple string match is not enough to capture semantic similarity between words. For example, the words "walk" and "go" do not match as strings, but they share very similar semantics. One way to derive this information is to use general-purpose corpora to cluster words into semantic classes. An alternative is to use available resources like WordNet to measure semantic distance of words.

In terms of dealing with sparse data, another direction is to do deeper text analysis to capture more regularizations in the data. Such analysis may include ongoing work like PropBank[33] by University of Pennsylvania and NomBank[40] by New York University. Analysis based on this work can generate regularized semantic representations for lexically or syntactically related sentence structures. Although deeper analysis may even be less accurate, our framework is designed to handle this. So we can expect them to provide further improvements in IE performance.

Appendix A

Proof: ψ^1_{SFD} is a kernel

First we define node set $U = \{n_i | 1 \leq i \leq N\}$, which contains all nodes from all the dependency graphs in the corpus. $C(n_1, n_2)$ can be seen as a dot product between two feature vectors generated from two graph nodes. For each node the features are $\{< r_i, a_i > | r_i \in R, a_i \in V\} \cup \{r_i | r_i \in R\}$. R and V here are the dependency label set and vocabulary set respective. Since dot product is always a kernel, $C(n_1, n_2)$ is a kernel.

As specified in chapter 7, ψ_{SFD} traverses each dependency graph from an entity node in a specific order. We can represent the node sequence seen by ψ_{SFD} as $(n_1, n_2, ..., n_k)$ for a graph with k nodes. Suppose m is the maximum length of all nodes sequences. Then we can assume all node sequences are of length m: for a sequence having less than m nodes, we pad empty nodes to the end and make it of length m.

We define $\tilde{C}(n_1, n_2)$ as an extension of $C(n_1, n_2)$. $\tilde{C}(n_1, n_2) = C(n_1, n_2)$ if both n_1 and n_2 are non-empty, otherwise $\tilde{C}(n_1, n_2) = 0$. Suppose the gram matrix formed by $C(n_1, n_2)$ is A, then the gram matrix B formed by $\tilde{C}(n_1, n_2)$

is

$$\begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix}$$

Since A is symmetric and positive semi-definite, B is symmetric and positive semi-definite. So $\tilde{C}(n_1, n_2)$ is a kernel.

Given two graphs G and G', we can rewrite

$$\psi_{SFD}(E, E') = \sum_{1 \leq i \leq m} \tilde{C}(n_i, n_i'),$$

where $(n_i | 1 \leq i \leq m)$ and $(n_i' | 1 \leq i \leq m)$ are the node sequences visited by ψ_{SFD} for graph G and G'. Since the linear combination of kernels is still a kernel, we can conclude that ψ_{SFD} is a kernel. $\diamondsuit$

Bibliography

[1] *Proceedings of the Third Message Understanding Conference(MUC-3).* Morgan Kaufmann, 1991.

[2] *Proceedings of the Fourth Message Understanding Conference(MUC-4).* Morgan Kaufmann, 1992.

[3] *Proceedings of the Fifth Message Understanding Conference(MUC-5).* Morgan Kaufmann, 1993.

[4] *Proceedings of the Sixth Message Understanding Conference(MUC-6).* Morgan Kaufmann, 1995.

[5] *Proceedings of the Seventh Message Understanding Conference(MUC-7).* 1998.

[6] D. Appelt, J. Hobbs, J. Bear, D. Israel, M. Kameyama, A. Kehler, D. Martin, K. Meyers, and M. Tyson. *SRI International FASTUS system: MUC-6 test results and analysis.* In Proceedings of the Sixth Message Understanding Conference, 1996.

[7] M. Asahara and Y. Matsumoto. *Japanese Named Entity Extraction with Redundant Morphological Analysis.* In Proceedings of Human Language Technology Conference(HLT-NAACL), 2003.

[8] I. G. B. Boser and V. Vapnik. *An training algorithm for optimal margin classifiers.* In Fifth Annual Workshop on Computational Learning Theory, 1992.

[9] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. *Nymble: a high-performance learning name-finder.* In Proceedings of the Fifth Conference on Applied Natural Language Processing, 1997.

[10] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. *Exploiting diverse knowledge sources via maximum entropy in named entity recognition.* In Proceedings of the Sixth Workshop on Very Large Corpora, Montreal, 1998.

[11] M. E. Califf and R. J. Mooney. *Relational Learning of Pattern-Match Rules for Information Extraction.* Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing, Menlo Park, CA, 1998.

[12] C.-C. Chang and C.-J. Lin. *Training nu-Support Vector Classifiers: Theory and Algorithms.* Neural Computation, 2001.

[13] H. L. Chieu, H. T. Ng, and Y. K. Lee. *Closing the Gap: Learning-Based Information Extraction Rivaling Knowledge-Engineering Methods.* In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 2003.

[14] M. Collins and N. Duffy. *Convolution Kernels for Natural Language.* In Proceedings of Neural Information Processing Systems, 2001.

[15] M. Collins and N. Duffy. *Parsing with a Single Neuron: Convolution Kernels for Natural Language Problems.* In Technical Report UCS-CRL-01-10. UC Santa Cruz, 2001.

[16] M. Collins and S. Miller. *Semantic tagging using a probabilistic context free grammar.* In Proceedings of 6th Workshop on Very Large Corpora, Montreal, Canada, 15-16 August., 1997.

[17] C. Cortes, P. Haffner, and M. Mohri. *Rational Kernels: Theory and Algorithms.* Journal of Machine Learning Research (JMLR), 2004.

[18] C. Cortes and V. Vapnik. *Support-Vector Networks.* Machine Learning, 1995.

[19] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines.* Cambridge University Press, 2000.

[20] A. Culotta and J. Sorensen. *Dependency Tree Kernels for Relation Extraction.* In Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 2004.

[21] D. Fisher, S. Soderland, J. McCarthy, F. Feng, and W. Lehnert. *Description of The Umass System As Used For MUC-6.* In Proceedings of the Sixth Message Understanding Conference, 1996.

[22] R. Grishman. *The NYU System for MUC-6 or Where's the Syntax?* In Proceedings of the Sixth Message Understanding Conference, 1996.

[23] R. Grishman. *Information Extraction: Techniques and Challenges.* Information Extraction (International Summer School SCIE-97), Springer-Verlag, 1997.

[24] R. Grishman. *Information Extraction.* The Oxford Handbook of Computational Linguistics, 2003.

[25] Z. S. Harris. *Linguistic Transformations for Information Retrieval.* In Proceedings of International Conference on Scientific Information, 1957.

[26] D. Haussler. *Convolution Kernels on Discrete Structures.* UC Santa Cruz Technical Report UCS-CRL-99-10, 1999.

[27] K. Humphreys, R. Gaizauskas, S. Azzam, C. Huyck, B. Mitchell, and Y. W. H. Cunningham. *University Of Sheffield: Description Of The LaSIE-II System As Used For MUC-7.* In Proceedings of the Seventh Message Understanding Conference, 1998.

[28] H. Isozaki and H. Kazawa. *Efficient Support Vector Classifiers for Named Entity Recognition.* In Proceedings of the 18th International Conference on Computational Linguistices, 2002.

[29] T. Jaakkola, M. Meila, and T. Jebara. *Maximum entropy discrimination.* Technical Report of Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1999.

[30] T. Joachims. *Making large-scale support vector machine learning practical.* MIT Press, Cambridge, MA, 1998.

[31] T. Joachims. *Text categorization with support vector machines: learning with many relevant features.* In Proceedings of 10th European Conference on Machine Learning(ECML-98), Chemnitz, DE, 1998.

[32] N. Kambhatla. *Combining Lexical, Syntactic and Semantic Features with Maximum Entropy Models for Extracting Relations.* In Proceedings of the

42nd Annual Meeting of the Association for Computational Linguistics, 2004.

[33] P. Kingsbury and M. Palmer. *From Treebank to PropBank.* In Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC), 2002.

[34] T. Kudo and Y. Matsumoto. *Chunking with support vector machines.* In Proceedings of NAACL, 2001.

[35] J. Lafferty, A. McCallum, and F. Pereira. *Conditional random fields: Probabilistic models for segmenting and labeling sequence data.* In Proceedings of the International Conference on Machine Learning (ICML), 2001.

[36] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. *Text Classification using String Kernels.* Journal of Machine Learning Research, 2002.

[37] J. Mayfield, P. McNamee, and C. Piatko. *Named Entity Recognition using Hundreds of Thousands of Features.* In Proceedings of Conference on Computational Natural Language Learning, 2003.

[38] W. McCulloch and W. Pitts. *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics, 1943.

[39] A. Meyers, R. Grishman, M. Kosaka, and S. Zhao. *Covering Treebanks with GLARF.* In ACL/EACL Workshop on Sharing Tools and Resources for Research and Education, 2001.

[40] A. Meyers, R. Reeves, C. Macleod, R. Szekeley, V. Zielinska, and B. Young. *The Cross-Breeding of Dictionaries.* Proceedings of LREC-2004, Lisbon, Portugal, 2004.

[41] A. Mikheev and C. Grover. *LTG: Description of the NE recognition system as used for MUC-7.* In Proceedings of the Seventh Message Understanding Conference, 1998.

[42] S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz, R. Stone, and R. Weischedel. *BBN: Description of The SIFT System As Used For MUC-7.* In Proceedings of the Seventh Message Understanding Conference, 1998.

[43] S. Miller, H. Fox, L. Ramshaw, and R. Weischedel. *A novel use of statistical parsing to extract information from text.* In Proceedings of the 6th Applied Natural Language Processing Conference, 2000.

[44] E. Riloff. *Automatically constructing a dictionary for information extraction tasks.* In Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93), 1993.

[45] E. Riloff. *Automatically constructing a dictionary for information extraction tasks.* Proceedings of the 11th National Conference on Artificial Intelligence(AAAI-93), 1993.

[46] F. Rosenblatt. *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological Review, 1958.

[47] D. Roth and W. tau Yih. *Probabilistic reasoning for entity and relation recognition.* In Proceedings of the 18th International Conference on Computational Linguistices, 2002.

[48] N. Sager, C. Friedman, and M. Lyman. *Medical Language Processing: Computer Management of Narrative Data.* Addision Wesley, 1987.

[49] G. Salton and C. S. Yang. *A Vector Space Model for Automatic Indexing.* Communication of the ACM, 1975.

[50] B. Schoelkopf, J. Weston, E. Eskin, C. Leslie, and W. S. Noble. *Dealing with Large Diagonals in Kernel Matrices.* Principles of Data Mining and Knowledge Discovery. Lecture Notes in Computer Science, 2002.

[51] S. Sekine, R. Grishman, and H. Shinnou. *A decision tree method for finding and classifying names in Japanese texts.* In Proceedings the Sixth Workshop on Very Large Corpora, 1998.

[52] S. Sekine and C. Nobata. *Definition, Dictionary and Tagger for Extended Named Entities.* In Proceedings of the Forth International Conference on Language Resources and Evaluation, 2004.

[53] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. *CRYSTAL: Inducing a Conceptual Dictionary.* In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI), 1995.

[54] J. Suzuki, T. Hirao, Y. Sasaki, and E. Maeda. *Hierarchical Directed Acyclic Graph Kernel: Methods for Structured Natural Language Data.* In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 2003.

[55] V. N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience Publication, 1998.

[56] H. Yamada, T. Kudoh, and Y. Matsumoto. *Japanese named entity extraction using support vector machines (in Japanese).* In IPSJ SIG Notes NL-142-17, 2001.

[57] D. Zelenko, C. Aone, and A. Richardella. *Kernel methods for relation extraction.* Journal of Machine Learning Research, 2003.

[58] S. Zhao, A. Meyers, and R. Grishman. *Discriminative Slot Detection Using Kernel Methods.* In Proceedings of the 20th International Conference on Computational Linguistices, 2004.