1-1-2009

Bootstrapping events and relations from text

Ting Liu
University at Albany, State University of New York, tl7612@albany.edu

The University at Albany community has made this article openly available. Please share how this access benefits you.

Follow this and additional works at: https://scholarsarchive.library.albany.edu/legacy-etd

Part of the Computer Sciences Commons

Recommended Citation
https://scholarsarchive.library.albany.edu/legacy-etd/73

This Dissertation is brought to you for free and open access by the The Graduate School at Scholars Archive. It has been accepted for inclusion in Legacy Theses & Dissertations (2009 - 2024) by an authorized administrator of Scholars Archive.
Please see Terms of Use. For more information, please contact scholarsarchive@albany.edu.
Bootstrapping Events and Relations from Text

By

Ting Liu

A Dissertation
Submitted to the University at Albany, State University of New York
In Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy

Department of Computer Science

2009
Bootstrapping Events and Relations from Text

By

Ting Liu

COPYRIGHT 2009
UNIVERISTY AT ALBANY

ABSTRACT

Bootstrapping Events and Relations from Text

Ting Liu

Supervisor: Professor Tomek Strzalkowski
Department of Computer Science

Information Extraction (IE) is a technique for automatically extracting structured data from text documents. One of the key analytical tasks is extraction of important and relevant information from textual sources. While information is plentiful and readily available, from the Internet, news services, media, etc., extracting the critical nuggets that matter to business or to national security is a cognitively demanding and time consuming task. Intelligence and business analysts spend many hours poring over endless streams of text documents pulling out reference to entities of interest (people, locations, organizations) as well as their relationships as reported in text. Such extracted “information nuggets” are then entered into a structured database for further analysis that may expose various trends or hidden relationships.

In this thesis, we constructed a semi-supervised machine learning method, which we call BEAR (Bootstrapping Events and Relations from Text), that effectively exploits statistical and structural properties of natural language discourse in order to rapidly acquire rules to detect mentions of events and other complex relationships in text, extract their key attributes, and construct template-like representations. The learning process exploits descriptive and structural redundancy, which is common in language and is considered critical for achieving successful communication despite distractions, different contexts, or incompatible semantic models between a speaker/writer and a hearer/reader. We also take advantage of the high degree of
referential consistency in discourse (e.g., as observed in word sense distribution, and arguably applicable to larger linguistic units), which enables the reader to efficiently correlate different forms of description across coherent spans of text.

Our system has been tested on the ACE-2005 corpus, which is U.S. Government official dataset for evaluating Information Extraction technology. The final results show that BEAR has significantly improvement comparing with the base run and performs better than currently event extraction systems.
ACKNOWLEDGMENTS

I would like to thank all people who gave me support and inspiration during my doctoral study.

For most, I am deeply grateful to my supervisor, Professor Tomek Strzalkowski, for his wisdom, guidance, and infinite patience. He was always accessible and provided insights, suggestions, and ideas that made this research a reality. He also spent a lot of time helping me improve my writing and presentation skill. I could not imagine having a better advisor and mentor to help me to be a real researcher.

I would like to thank my committee, Professor George Berg, and Professor Andrew Haas, for their insight suggestions and encouragement. I would also like to thank Dr. Nick Webb, who gave me great comments on my thesis and helped me a lot on my defense.

Finally, my deepest gratitude goes to my wife, daughter, and my parents. Without their tremendous love and support, I would not be able to go through such challenging journey.
# TABLE OF CONTENTS

Chapter 1: Introduction ...........................................................................................................1

Chapter 2: The literature Review .............................................................................................8

2.1 Information Extraction ......................................................................................................8

2.2 Message Understanding Conference (MUC) ..................................................................9

2.2.1 Introduction ..................................................................................................................9

2.2.2 Research issues addressed in MUC ..........................................................................12

2.2.2.1 MUC evaluation and data ......................................................................................13

2.2.2.2 Architecture and achievements of IE systems in MUCs .....................................15

2.2.3 Unsolved Problems in MUC .......................................................................................19

2.3 Automatic Content Extraction (ACE) Evaluation .........................................................22

2.4 Bootstrapping techniques in Information Extraction ..................................................26

2.5 Pattern representations techniques in Information Extraction ....................................29

2.6 Conclusions .....................................................................................................................32

Chapter 3: Pattern generation and validation ......................................................................34

3.1 Structure of an event ........................................................................................................34

3.2 An extended representation of events ...........................................................................40

3.2.1 Syntactic representations .........................................................................................40

3.2.2 Semantic representations .........................................................................................44

3.3 Pattern generation ..........................................................................................................50

3.4 Pattern validation ............................................................................................................53

3.4.1 Validation of the event trigger ..................................................................................55

3.4.2 Validation of sub-patterns .......................................................................................57

3.4.3 Validation of the whole event pattern ......................................................................59

3.5 Order in applying patterns .............................................................................................62

3.6 Experimental Results – first iteration ..........................................................................62

3.6.1 Experimental dataset ...............................................................................................63

3.6.2 Experiment metrics ...................................................................................................64

3.6.3 Experiments and Evaluations ....................................................................................65

3.7 Conclusions .....................................................................................................................68

Chapter 4: Unsupervised Pattern Learning ......................................................................70

4.1 New Patterns derived from known patterns ..................................................................70

4.1.1 Finding new triggers for known patterns .................................................................70

4.1.2 Deriving new patterns through sub-pattern mutation ............................................72

4.2 Creating new patterns from the linguistic context of the extracted events ..................76

4.2.1 Forming new patterns around alternative triggers ...................................................76

4.2.2 Finding new events through trigger and role coreference .....................................80

4.2.3 Prepositions as role indicators .................................................................................85

4.3 Relaxing pattern constraints ...........................................................................................90

4.4 Pattern validation using both training and test corpora .............................................91

4.5 Experiments and Evaluation ...........................................................................................92

4.5.1 BEAR performance and analysis .............................................................................93

4.5.1.1 BEAR performance at various pattern thresholds ..............................................93
4.5.1.2 Failure Analysis of BEAR’s performance.................................95
4.5.2 BEAR performance with various combination of learning methods
.......................................................... 101
4.6 Conclusions............................................................................................... 103
Chapter 5: Event Coreference ........................................................................ 105
5.1 Event mention coreference......................................................................... 105
5.2 Prior research in event coreference ................................................................. 106
5.3. Features selected for computing event coreference in BEAR ............... 107
5.4. Experiments and discussion ......................................................................... 111
  5.4.1 Corpora........................................................................................................ 111
  5.4.2 ACE event scoring system........................................................................ 112
  5.4.3 Experiment strategies.................................................................................... 114
  5.4.4 Tuning the coreference classifier on the human-annotated training corpus
       ................................................................................................................... 114
       5.4.4.1 Event coreference resolution baseline used.............................. 114
       5.4.4.2 Find best features for BEAR coreference............................... 115
       5.4.5 Comparing BEAR performance against ACE systems.............. 117
5.5 Conclusion and Future plans.......................................................................... 119
Chapter 1

INTRODUCTION

One of the key analytical tasks is extraction of important and relevant information from textual sources. While information is plentiful and readily available, from the Internet, news services, media, etc., extracting the critical nuggets that matter to business or to national security is a cognitively demanding and time consuming task. Intelligence and business analysts spend many hours poring over endless streams of text documents pulling out reference to entities of interest (people, locations, organizations) as well as their relationships as reported in text. Such extracted “information nuggets” are then entered into a structured database for further analysis that may expose various trends or hidden relationships.

Information extraction technology was conceived to ease this burden, particularly in the face of the current information overload, and to automate the nugget extraction process to the greatest possible extent, thus freeing the analyst to concentrate on understanding the meaning and the implications of the extracted facts. However, in order to be useful, the automated technology must be highly reliable, accurate, and rapidly adaptable to new tasks and domains. This has been an elusive target. While some basic IE capabilities reached the level of practical usability (for example, extraction of references to people names, or organization names), other more complex tasks, such as extraction of events (or more broadly n-ary relations) that tie these entities remain very hard to automate. One of the great obstacles for IE to become a usable technology is the high cost associated with developing an automated system and then adapting it to the changing user needs and data characteristics.
Overcoming this brittleness is the goal of much of Information Extraction (IE) research today, a sub-area of Natural Language Processing (NLP). It is also the subject of this thesis.

Generally speaking, IE research focuses at three levels: extraction: named entity, (binary) relation, and event (n-ary relation). Named entity extraction locates and classifies in-text references to people, locations, organizations, as well as many other specific entity types such as weapons or chemical compounds. The techniques developed for named entity recognition are already quite mature (Strzalkowski and Wang 1996) (Bikel et al, 1999) (Zhang et al, 2004).

Relation extraction finds references to semantic relations between any two entities mentioned in text. An example would be the relationship between an organization and the location of its headquarters. Although relation extraction is more difficult than entity extraction, there is a number of promising supervised and semi-supervised machine learning methods that have been developed (Zelenko et al, 2002) (Agichtein and Gravano 2000). Event extraction is to identify multi-way relations between 2 or more entities that are also temporally and geographically circumscribed, i.e., specific happenings such as buying or moving or attacking that involve specific participants at a specific time and place. Event extraction is the most complex of the IE tasks for which an automated technology is least developed. There are several reasons for this. First, events come in a variety of types, each of which may have a different number and types of participants (or arguments). For example, an Attack event would have arguments such as: the Attacker, the Target, the Instrument (or weapons used in the attack), the Time, and the Place. Furthermore, each time such an event is mentioned in text it may refer only to some of its arguments, and moreover it may use a nearly unlimited supply of syntactic forms to do so (c.f., Figure 1.1 that shows two mentions of an Attack event with different sets of arguments). Thus, an event extraction system needs strong adaptive ability for
the various structures in same domain and portability while working across subject matter domains. Existing approaches to event extraction are generally not easily adaptable. Most systems are built around manually created extraction rules or through supervised machine learning methods applied to manually annotated training corpora. In this thesis we describe a novel approach that delivers an adaptability breakthrough in the event extraction task.

This morning, at 8:36, a former employee opened fire in the auto supply warehouse where he once worked.

President Bush called upon the world to stop terrorism after a suicide bomb attack on a bus in Jerusalem today.

Figure 1.1 Two Attack event mentions

In this thesis, we constructed a semi-supervised machine learning method, which we call BEAR (Bootstrapping Events and Relations from Text), that effectively exploits statistical and structural properties of natural language discourse in order to rapidly acquire rules to detect mentions of events and other complex relationships in text, extract their key attributes, and construct template-like representations. The learning process exploits descriptive and structural redundancy, which is common in language and is considered critical for achieving successful communication despite distractions, different contexts, or incompatible semantic models between a speaker/writer and a hearer/reader. We also take advantage of the high degree of referential consistency in discourse (e.g., as observed in word sense distribution by Gale et al (1992), and arguably applicable to larger linguistic units), which enables the reader to efficiently correlate different forms of description across coherent spans of text.
Figure 1.2 shows the overall architecture of BEAR. The architecture we describe here consists of two parts: (1) supervised acquisition of initial extraction rules from an annotated training corpus (denoted by the upper dashed line region), and (2) self-adapting unsupervised multi-pass bootstrapping by which the system learns new rules as it reads un-annotated text using the rules learned from the supervised pass and from the subsequent unsupervised passes (the lower dashed region). When a sufficient quantity and quality of text material is supplied, the system will learn many ways in which a specific class of events can be described. This includes the capability to detect individual event mentions using a system of context-sensitive triggers and to isolate pertinent attributes such as Agent, Victim, Instrument, Time, Place, etc., as may
be specific for each type of event. This method produces an accurate and highly adaptable event extraction that significantly outperforms current information extraction techniques both in terms of accuracy and robustness, as well as in deployment cost.

**Organization of this thesis**

This thesis is organized in 4 chapters as follows. Chapter 2 presents the background of IE research and the current open research problems. We focus particularly on research associated with two U.S. Government sponsored programs: Message Understanding Conference (MUC) and Automatic Content Extraction program (ACE), which made the most significant contributions to IE research and evaluation. Thanks to MUC and ACE programs, IE researchers have a regular opportunity to evaluate and compare their systems and discuss effectiveness of differed techniques used. From the first MUC in 1991 to the most recent ACE (2007), IE systems have made big progress in accuracy and reliability evolving from rule-based (expert made) systems to supervised machine learning systems. In chapter 2, we also introduce the unsupervised bootstrapping technique that has achieved promising results in application to named entity extraction and also to relation extraction. These earlier results inspired our research into the use of this technique to improve accuracy, portability, and adaptability of event extraction.

Chapter 3 presents the pattern representation and validation, which is the supervised learning step (the first iteration) of BEAR. The patterns are automatically composed from annotated events in the training corpus. The data used in the experiments described in this thesis is the official ACE-2005 text corpus used in all recent ACE evaluations. Each event in the corpus has a sentence level span and contains a trigger and a list of event roles. In order to accurately
represent each event structure, we build syntactic dependency relations that connect the trigger to the event roles, which are assigned to named entities pre-identified in the ACE data. We also proposed a novel approach to determine the correct sense of each event trigger by traversing semantic relations in Wordnet, including synonym, hyponym, hypernym, and derived links.

Using a combination of syntactic structural and semantic information found in the training corpus, BEAR generates extraction patterns out of sample events. Our objective is to make these patterns highly adaptable to structural variations in event descriptions; to enable this, each pattern is composed of a list of sub-patterns, which are built over the dependency structure between the trigger and each event role. This way it will be easy to add a new subpattern into a pattern as well as to remove or replace it with another one.

After patterns are generated, we need to estimate their accuracy before they can be used. We calculate the projected pattern accuracy as a ratio of the number of correctly extracted events by this pattern to the total number of events it extracted from a training corpus. If this projected accuracy is over a certain threshold, the pattern will be retained for the next learning iteration, which will be an unsupervised learning step. This pattern validation and selection process is repeated after each learning cycle with the training corpus expanded by automatically extracted events in unannotated data.

In Chapter 4 we describe our patterns adaptation techniques that allow BEAR to automatically learn more extraction patterns beyond what was derived from the training corpus. Basically, we have two methods to do the adaption: one is to manipulate already known patterns so that they can be adjusted to slight variations on form and style of reference to events typical in text
documents; the other technique is to utilize the linguistic context surrounding extracted events to find new event structure, which are not comparable to any learned structures.

Also in Chapter 4, we describe a pattern validation process, which is necessary for the system to accept automatically derived patterns. After new patterns are created, they are tested against the already annotated data for their expected accuracy, so that only the good quality patterns are in fact learned.

In Chapter 5 we discuss how BEAR can be advanced from extracting event mentions to assembling complete event representations by finding and combining multiple coreferential event mentions within a single document. We propose an extended set of features for computing coreference probability and then test two classifiers, Maximum Entropy (ME) and Support Vector Machine (SVM), in order to select one with better average performance. The resulting event coreference processor is then added to BEAR so that full event representation can be obtained. This also allows us to compare BEAR performance against other contemporary IE systems that participated in ACE. The final result shows that BEAR significantly outperforms the best ACE systems.
Chapter 2

THE LITERATURE REVIEW

This chapter describes prior research done in information extraction (IE) area, especially in event extraction, which is the topic of this thesis. The present chapter is organized as follows: in section 2.1, we provide a definition of information extraction and its scope; in section 2.2, we discuss the Message Understanding Conference (MUC), which is the standardized evaluation of various IE tasks including event extraction. We also overview some of the participating systems and the problems still unsolved after 10 years of initial research (1987-1997). Section 2.3 contains an introduction to Automatic Content Extraction (ACE), which is another important research and evaluation program in IE following MUCs; a more detailed discussion of ACE will follow in Chapter 3. In section 2.4, we compare the effectiveness of language processing techniques in our system with those used in MUCs and ACE. In section 2.5, we discuss various pattern representations used in IE systems.

2.1 Information Extraction

The following is the definition of IE from NIST website¹,

Information Extraction is a technology that is futuristic from the user’s point of view in the current information-driven world. Rather than indicating which documents need to be read by a user, it extracts pieces of information that are salient to the user’s needs. Links between the extracted information and the original documents are maintained to allow the user to reference context.

¹ From NIST website about MUC, http://www.itl.nist.gov/ia/894.02/related_projects/muc/index.html
Chapter 2

The kinds of information that systems extract vary in detail and reliability. For example, named entities such as persons and organizations can be extracted with reliability in the 90th percentile range, but do not provide attributes, facts, or events that those entities have or participate in. The following results are current indicators of the state of the art in information extraction.

<table>
<thead>
<tr>
<th>Items of Information</th>
<th>Percentile Reliability (F-score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities</td>
<td>90</td>
</tr>
<tr>
<td>Attributes</td>
<td>80</td>
</tr>
<tr>
<td>Facts</td>
<td>70</td>
</tr>
<tr>
<td>Events</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 2.1 Reliability of IE

The goal of IE is to pull specific nuggets of information out of unprocessed text files so that they can be saved into a structured database. Most current IE systems require prior training on datasets that were pre-annotated for entities, events and other items of interest according to user-provided specifications. In the training phase, an IE system learns or calibrates the extraction rules from the annotated text corpus (with or without a human involved the process). The learned rules are then applied to an unannotated text corpus of similar genre and covering similar subject matter against which their performance can be objectively measured. As shown in Table 2.1, the event extraction task is more difficult than other IE tasks and it needs further research, which is the focus of this thesis.

2.2 Message Understanding Conference (MUC)

2.2.1 Introduction

\footnote{Especially when switching to a data set in different domain, the performance entity extraction should not be affected significantly because name entity extraction is fairly domain independent. On the contrary, the event extraction is highly related to the syntactic structures, which are quite different in various domains, used for event expression. So if the measures are crossing different domains, the event extraction performance probably gets worse results.}
The Message Understanding Conferences (MUCs) were funded by the Defense Advanced Research Projects Agency (DARPA) to provide a forum for evaluation and to stimulate research into IE. MUC organizers created a common set of rules including datasets and tasks under which performance of disparate systems could be easily compared. There were 7 MUC conferences organized between 1987 and 1998, during which time, many IE problems have been explored and a lot of techniques have been developed to address and solve these problems. Comparing the systems from MUC 7 and those from MUC-1 (known as MUCK I), the progress and achievements can be easily identified. Although each MUC may have its own specific focus and subtasks, there were two major tasks that were present in each MUC:

1. Improve the event extraction performance
2. Automatically extract different types of events crossing different domains without human intervention.

In MUCs, the training data, which contained manually annotated sample events, was provided to participating groups ahead of time so that the system adapted to recognize these events. Later, a corpus of previously unseen and unannotated documents was supplied for evaluation. The events were defined as templates with slots, where each slot specified an entity involved in the event and its role. Figure 2.1 shows an example text document from the training corpus of MUC-3 evaluation, which used a dataset of messages from Foreign Broadcast Information Service (FBIS). In the message quoted in Figure 2.1, a kidnapping event is mentioned. The message supplies a number of details about the event: the date (3 April), the victim (Liberal Senator Federico Estrada Velez), and the attackers (The Extraditables), among others. Figure 2.2 shows how the same event is represented in a template. Although this template specifies as
many as 17 possible slots that may be present, the system only needs to fill those that are explicitly mentioned in text.

![Figure 2.1 Example of a MUC3 message]

<table>
<thead>
<tr>
<th>SLOT TYPE</th>
<th>VALUES of SLOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. MESSAGE ID</td>
<td>TST1-MUC3-0080</td>
</tr>
<tr>
<td>1. TEMPLATE ID</td>
<td>1</td>
</tr>
<tr>
<td>2. DATE OF INCIDENT</td>
<td>03 APR 90</td>
</tr>
<tr>
<td>3. TYPE OF INCIDENT</td>
<td>KIDNAPPING</td>
</tr>
<tr>
<td>4. CATEGORY OF INCIDENT</td>
<td>TERRORIST ACT</td>
</tr>
<tr>
<td>5. PERPETRATOR: ID OF INDIV(S)</td>
<td>&quot;THREE HEAVILY ARMED MEN&quot;</td>
</tr>
<tr>
<td>6. PERPETRATOR: ID OF ORG(S)</td>
<td>&quot;THE EXTRADITABLES&quot; / &quot;EXTRADITABLES&quot;</td>
</tr>
<tr>
<td>7. PERPETRATOR: CONFIDENCE</td>
<td>CLAIMED OR ADMITTED : &quot;THE EXTRADITABLES&quot; / &quot;EXTRADITABLES&quot;</td>
</tr>
<tr>
<td>8. PHYSICAL TARGET: ID(S)</td>
<td>-</td>
</tr>
<tr>
<td>9. PHYSICAL TARGET: TOTAL NUM</td>
<td>-</td>
</tr>
<tr>
<td>10. PHYSICAL TARGET: TYPE(S)</td>
<td>-</td>
</tr>
</tbody>
</table>
2.2.2 Research issues addressed in MUC

The groups participating in MUC evaluations prepared detailed reports about the techniques used in their systems, as well as an error analysis of their systems performance: what worked, what did not work, and how the performance may be improved in the future. These analyses not only helped the participating groups to improve their systems but also helped MUC organizers decide the scope of tasks for subsequent MUCs. In this section we discuss two topics: the first topic is the selection of the evaluation data in MUCs and their effect on performance of the participating systems. The second topic is an overview of language processing techniques employed in the participating systems and their effectiveness.

---

3 All possible correct answers to a slot are filled in the template and are separated by slashes in the slot.
2.2.2.1 MUC evaluation and data

Adopted from IR evaluation strategy, MUC conferences introduced equivalents of precision and recall into IE evaluation, as defined in (2.1) and (2.2) below. $N_{\text{correct}}$ is the number of template slots that have been correctly filled. $N_{\text{incorrect}}$ is the number of slots that were incorrectly filled or unfilled. $N_{\text{key}}$ is the number slots annotated by human experts in test corpus.

$$
\text{Precision } (P) = \frac{N_{\text{correct}}}{N_{\text{correct}} + N_{\text{incorrect}}} \quad (2.1)
$$

$$
\text{Recall } (R) = \frac{N_{\text{correct}}}{N_{\text{key}}} \quad (2.2)
$$

However, it is not straightforward to compare systems using both precision and recall, since these metrics are conversely correlated and increase in one can be achieved at the expense of the other. In order to facilitate a more meaningful ranking of the systems, the F-measure (2.3) was introduced into evaluation system since MUC 4 as a single number score. It is a weighted combination of precision and recall, giving more weights to recall or precision depending upon an envisioned possible application.

$$
F - \text{measure} = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R} \quad (2.3)
$$

The adoption of this evaluation technique is one of significant contributions of MUCs to IE research and it was widely accepted for measuring performance of IE systems also outside of MUCs.

Another key issue in accurate evaluation of IE performance is selection of the test data. In MUCK I\(^4\) and MUCK II, brief operational messages exchanged between Navy ships were used. The corpus size in MUCK II was only about 3,000 words for training and 158 words for

---

\(^4\) MUCK I has only “DEMO” session and no task defined
testing, which seems laughably small now. It was literally possible for the research teams to analyze the data manually and create semantic structures that fit the provided messages. Thus the first two MUC’s should be considered more as preliminary tryouts than as serious evaluations. In MUC 3, the corpus size was increased rather dramatically to 400,000 words for the training corpus and 33,000 words for the test corpus, which rendered heavily manual knowledge-based approaches quite impractical. This change of scale also explains why average system performance in MUC-3 (Table 2.2) falls off sharply when compared to MUCK II. But it also gave impetus to development of fully automated systems, especially those based on robust, approximate methods such as cascaded finite state transducers (Hobbs et al. 1997), at the expense of the more accurate but hand-crafted heuristic approaches (e.g., Grishman et al. 1991).

MUC 4 had a similar set of tasks as MUC 3 and it used the same training corpus as MUC 3. Of course, the MUC-4 test corpus was different; it was from the same source as MUC-3 text corpus but from a different time period. As expected, the test corpus contained references to different events than those in the training corpus, which required a significant shift in system training from mere classification or adaptation to generalization with respect to latent event structures (Sundheim 1992). Since MUC 5, a new evaluation corpus was created for each evaluation, an extraordinarily time consuming task. Therefore MUC 6 and MUC 7 mark the return to small data sets, which is partly due to the complexity of these evaluations: only 100 training documents and another 100 test documents were prepared. This change hurt the

---

5 data set was switched to news wire reports
6 Both MUC 3 training and testing data set originated from Foreign Broadcast Information Service (FBIS) archival database from 1989 to early 1990. The testing corpus of MUC 4 was selected from the FBIS articles from August to December 1998.
7 Since MUC 6, MUC task is no longer only template (event) based. Several new tasks (Table 2.2), such as entity extraction, the detection of entity coreferences, etc. were introduced.
Chapter 2

entity extraction performance of some participating systems (A. Borthwick et al. 1998) (S. Miller et al. 1998) in MUC 6 and MUC 7, because these systems used statistical models and machine learning algorithms and needed bigger training datasets.

In the ten years of MUC program, different types of events were defined and various corpora were selected from different resources and different time period. There were two main purposes of doing this. One purpose was to evaluate the robustness of participating systems. Another one was to push participating system towards a greater portability across domains\(^8\).

<table>
<thead>
<tr>
<th>Evaluation/Task</th>
<th>Domain (Topic)</th>
<th>Named Entity extraction</th>
<th>Coreference</th>
<th>Scenario Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUC 2</td>
<td>Fleet Operations</td>
<td></td>
<td></td>
<td>R &lt; 0.7 P &lt; 0.8</td>
</tr>
<tr>
<td>MUC 3</td>
<td>Terrorist activities in Latin America</td>
<td></td>
<td>R &lt; 0.5 P &lt; 0.7</td>
<td></td>
</tr>
<tr>
<td>MUC 4</td>
<td>Terrorist activities in Latin America</td>
<td></td>
<td>F &lt; 0.57</td>
<td></td>
</tr>
<tr>
<td>MUC 5</td>
<td>Corporate Joint Ventures, Microelectronic production</td>
<td>F &lt; 0.57</td>
<td>JV(^9) F &lt; 53% ME(^10) F &lt; 50%</td>
<td></td>
</tr>
<tr>
<td>MUC 6</td>
<td>Negotiation of Labor Disputes and Corporate management Succession</td>
<td>F &lt; 0.97</td>
<td>R &lt; 63% P &lt; 72% F &lt; 0.57</td>
<td></td>
</tr>
<tr>
<td>MUC 7</td>
<td>Aircraft accidents, Rocket/Missile Launches</td>
<td>F &lt; 0.94</td>
<td>F &lt; 62% F &lt; 0.51</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2 the evaluation result of MUC conferences

2.2.2.2 Architecture and achievements of IE systems in MUCs

The fact that linguistic techniques play an important role in the IE process was proved in MUC evaluation conferences. Linguistic features reflect syntactic or semantic attributes of the words in text. They help building logical structures and relations out of linear text. If systems

---

\(^8\) The portability of a system means how much efforts need to be done when switching a current IE system for a new type of event

\(^9\) JV – Joint Venture

\(^10\) ME – MicroElectronic
can learn to extract these features accurately from a training text, it should make it easier to process “unseen” data by assuming that it is comparable to the training data. In MUC 3, systems that were not using linguistic techniques (Hilster and Amnon Meyers 1991) (Weir et al. 1991) or were only using limited linguistic techniques (Dolan et al. 1991) (Deogun and Raghavan 1991) showed relatively low performance. Other systems, such as BBN (Ralph Weischedel et al. 1991) and NYU (Ralph Grishman, Sterling, and Macleod 1991), derived syntactic structures from text, while UMASS (Lehnert et al. 1991) and GE (Krupka et al. 1991) used both syntactic and semantic processing to analyze documents. The latter systems were listed at top ranks in the evaluation and the UMASS system achieved the best F-score. Starting with MUC 4, linguistic processing was standard for all participating systems. Usually, linguistic processing techniques can be grouped into 3 main categories, syntactic, semantic, and discourse. Syntactic processing includes chunking sentences, detecting phrases, and assigning part of speech tags, etc. It requires either shallow or full phrase structure analysis of sentences. Semantic processing includes word sense disambiguation, named entity extraction, event recognition, and semantic role labeling. Coreference resolution is an example of discourse processing.

Although MUC participating systems represented many different approaches, they usually contained following components:

1. Preprocessor, which contains sentence boundary detection, part of speech tagging, lexical look up and disambiguation, etc

2. Filter, which removes the information likely to be irrelevant. This process helps to improve the process efficiency.

3. Shallow parser, which chunks a sentence into noun phrases, verb phrases, etc.
Chapter 2

4. Full Parser, which derives a complete phrase structure of a sentence according to the underlying grammar, usually producing a parse tree.

5. Concept based pattern matcher, which does extraction of named entities

6. Semantic interpretation, which translates parse trees or shallow parser chunks into some logic forms or event frames

7. Coreference resolution, which links references to the same entities across the document and also resolves pronominal references.

8. Template filler, which fills an event template using semantic structures created by semantic interpretation.

The goal of MUC conferences was not only to improve system performance of each of the IE tasks, but also to make the solutions portable across domains. This is because only the systems that can work effectively on many different types of events in many different domains without requiring extensive re-engineering effort, would qualify for real-life applications. For example, in MUC 3 and MUC 5, BBN (Ralph Weischedel et al. 1991) (Ralph Weischedel et al. 1993) spent 3 months/person adjusting their system to new domains. MUC organizers have made a lot of effort to push research into adaptability and portability issues. Although making an IE system portable across different types of events and domains is an extraordinarily hard task, some modules have a good potential to achieve portability: for example, part of speech tagging (POS), sentence parsing, named entity extraction, and coreference resolution (Ralph Grishman and Sundheim 1996). In MUC 6, named entity extraction was introduced as a separate task and several systems achieved significant success by reaching F-score nearly as high as 97%. Several different approaches have been applied to named entity extraction. For example, both BBN (Weischedel 1995) and NYU (Grishman 1995) systems used statistical

---

11 There were 7 domains involved in the tasks of 6 MUC conferences (MUCK I is not considered)
12 The task of name entity extraction involves finding person, organization, geographic location, time, currency, and percentage expressions.
Chapter 2

language modeling and pattern matching techniques. MITRE system (Aberdeen et al. 1995) applied a combination of hand-crafted rules and automatically trained rules. All these approaches showed competitive results\(^\text{13}\) on the automatic named entity extraction task.

In MUC 7, more participating systems used various machine learning approaches in the task of named entity extraction. IdentiFinder\(^\text{TM}\) (Miller et al. 1998), BBN’s system, applied a HMM model and achieved a high F-score of 90.44% during the evaluation. In NYU system, MENE (Borthwick et al. 1998), Maximum Entropy model was used and its F-score was 84.22. In Lin’s (Lin 1998) system, named entities were extracted through collocation\(^\text{14}\) statistics, which based on Naïve-Bayes classifier. The system F-score reached 86.37. Besides all these approaches, further results in named entity extraction were reported outside of MUC evaluations. For example, RoboTag (Bennett, Aone, and Lovell 1997) is a decision-tree-based multilingual\(^\text{15}\) learning system for detecting and recognize named entities. Although this system wasn’t created for participation in MUC evaluation, it achieved an F-score of 88.1 when evaluated on the MUC corpus. The significant progress achieved in MUC-7 was thanks to successful adaption of machine learning techniques to named entity extraction task. Systems using ML only require annotated training data and can automatically learn from it to generate language models that will then apply to unannotated data sets. This dramatically reduces the amount of manual labor required to develop NE extraction and improves the portability to new domains (systems still need to be trained for each new domain). After MUC conferences, Automated

---

\(^\text{13}\) The F-score of named extraction evaluation in BBN system was 93.65 and in NYU system was 88.19, the highest F-score is 96.42 in MUC 6

\(^\text{14}\) A collocation is a habitual word combination, such as “weather a storm”, “file a lawsuit”, and “the falling yen”.

\(^\text{15}\) RoboTag was created based on a different machine learning algorithm comparing with the systems in MUCs and did name extraction in English and Japanese. MUCs program only evaluates system based on English.
Chapter 2

Content Extraction (ACE) workshops continued to provide forum for evaluation while steering research towards even more automatic and robust systems.

During the ten years development under MUCs, many advances have been made in IE area, which became one of the major subfields of natural language processing and attracted many researchers’ attentions. As the benchmarks of IE area, the evaluation method and data selection in MUCs are still the standard for current IE systems. With deeper exploration into this area, the original goal of MUCs shifted from domain-dependent event extraction to domain independent entity and event extraction. The participating systems were making a lot of effort to shift the system design from purely handcrafted rules to automatic training using machine learning techniques. This goal has been only partially achieved thus far: only some of the components of IE process have been made automatically adaptable at the end of MUCs. There remain many hard problems that require continuing research effort. These will be discussed in Section 2.2.3.

2.2.3 Unsolved Problems in MUC

The first unsolved problem is portability of event extraction. For all systems participating in MUCs, the semantic interpretation of the whole event depends upon manually designed rules built for a narrow domain. As mentioned in the previous section, this situation makes portability very hard, because adding a new type of event requires a linguist to define new rules and also to expand the existing domain knowledge as needed. For example, the event extraction tasks of MUC5, MUC 6, and MUC 7 had the similar levels of complexity (Bagga 1998), which means the task wasn’t getting harder except for changing event types and
domains. Nonetheless, significant effort was required by all participating groups just to adapt their systems to new domains and then to maintain the performance at the level comparable to earlier competitions (Table 2.2). In MUC 7, at least one system, Proteus (Yangarber and Grishman 1998), automatically learned preliminary extraction patterns from annotated data samples. Nonetheless, this still required manual preparation of the training collection before pattern generation, which was then followed by manual pattern refinement.\(^\text{16}\)

The second problem plaguing MUC systems was low recall in the event extraction task. The recall in MUC is measured by comparing the number of template slots that were correctly filled to the total number of available template slots in open templates. In MUC 7, the average recall of participating systems was near 50% and it hasn’t significantly changed from previous MUC conferences. One of the most effective ways to improve recall is to maximize generality of the extraction rules and patterns, but only when it is done without sacrificing precision. Rule generalization can be both syntactic and semantic. At the syntactic level, rules can be expressed in terms of phrase structures and part-of-speech tags, rather than using individual words. At the semantic level, references to specific named entities can be replaced with corresponding entity type tags. As an example, consider the following sentence describing a satellite launch event:

> “Arianespace Co. has launched an Intelsat communications satellite.”

Using a chunk parser we can segment this sentence into phrase-like chunks, as shown below:

> “(Arianespace Co. np) (has launched v) (an Intelsat communications satellite np)”

\(^{16}\) Remove noisy pattern and generalize overly specific patterns
This phrase structure can be further analyzed and assigned semantic features, such as named entity tags, for example, “Arianespace Co.” may be tagged as Organization and “an Intelsat communications satellite” may be tagged as Payload. When the actual entities are replaced by their entity types, we obtain a generalized event representation, as follows:

“(Organization np) (launch v) (Payload np)”

Rule generalization helps an IE system to find more events; however, the generalized rules will not find events using syntactic or semantic structures different than the events present in the training set. For example, the following reference to the same satellite launch event uses a radically different syntactic structure than the previous example: “…Ariane’s 22nd Launch for International Telecommunications Provider Intelsat”. Clearly, a system trained on only some structural forms would likely miss many other references to this and other events. This is also the key obstacle to achieving a higher recall. Thus, a novel idea is needed to extract more events from text, including those described in new and unexpected ways.

The third problem left unsolved in MUC is word sense disambiguation. This problem arises when we attempt event extraction using a set of trigger words because words that indicate presence of events do so only when used in a correct sense. For example, in order to recognize correctly a launch event based on a verb trigger, we need to check if “launch” is used with the sense of “propel with force”, but not in another sense, e.g., “give a start”. Normally, extraction rules

17 Payload is different from the general name entities, such as Person and Organization, which are defined based on what they are. A Person won’t change to another category in whatever domains. However, the payload will classify any goods being delivered in a Launch event. So this kind of categorization is tightly related to some specific event and under some other context structure, it may not be true for the same entity to be that categorization.

18 The fundamental rule of pattern generalization is that the basic structure, such as the order of phrases and the number of phrases, of the event sentence has to be kept. Otherwise the pattern is made up instead of generalization.
should cover enough context to help sense disambiguation; however, this becomes difficult to maintain as rules are generalized to increase their recall. For example, a syntactic pattern for the Start-Position event, such as “(Person np) (join v) (Organization, np)” would match “Toyoda, who joined Toyota, based in Toyota city…”, but so it would “I joined the Chicago Tea Party…”, which is not an Start-Position event. This problem gets still more serious when the synonyms of an event anchor are introduced as potential additional event triggers, again to improve system recall, because each of these extra triggers may introduce additional senses into the mix. For example, set up, found, and smoothen are synonyms of launch in Wordnet, but they have nothing to do with LAUNCH event. This is a key problem in the automatic IE and solving it requires more research.

2.3 Automatic Content Extraction (ACE) Evaluation

ACE is another important program in IE area. It began at 1999 right after MUC. Here are the goals and tasks of ACE, “The objective of the ACE program is to develop automatic content extraction technology to support automatic processing of human language in text form. The program is devoted to three sources types. These are namely newswire, broadcast news (with text derived from ASR), and newspaper (with text derived from OCR). ACE technology R&D is aimed at supporting various classifications, filtering, and selection applications by extracting and representing language content (i.e., the meaning conveyed by the data). Thus the ACE program requires the development of technologies that automatically detect and characterize this meaning.” Since ACE uses data not only from newswire sources, but also from speech recorded broadcast news, as well as OCR-scanned documents and Internet blogs, the input text is often of poor quality when compared to MUC data. Like MUC, ACE started with the

---

19 This definition is from ACE home page, http://www.nist.gov/speech/tests/ace/
Chapter 2

task of named entity extraction and co-reference resolution, and the relation and event extraction tasks were added only in 2004.

Named entity extraction task was first introduced in MUC 6 and in MUC 6 and 7, its performance already reached a very high level (F-measure is 0.97 and 0.94). In ACE program, even though this task became more complex and the evaluation method were stricter, the performance (Table 2.3 shows scores of top 5 participating systems in ACE 2005) remains very good. Table 2.4 also shows a big progress (overall value < 71.9%) in named entity coreference task compared to system performance (F-score < 62%) in MUC 7.

<table>
<thead>
<tr>
<th>Site</th>
<th>Overall</th>
<th>Broadcast Conversations</th>
<th>Broadcast News</th>
<th>Newswire</th>
<th>Telephone</th>
<th>Usernet Newsgroups</th>
<th>Weblogs</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBN Technologies</td>
<td>85.1</td>
<td>86.8</td>
<td>84.2</td>
<td>84.6</td>
<td>93.7</td>
<td>74.0</td>
<td>85.0</td>
</tr>
<tr>
<td>SRA team #2</td>
<td>84.7</td>
<td>83.3</td>
<td>84.5</td>
<td>84.8</td>
<td>93.8</td>
<td>75.8</td>
<td>82.4</td>
</tr>
<tr>
<td>SRA team #1</td>
<td>83.7</td>
<td>84.0</td>
<td>84.4</td>
<td>84.0</td>
<td>89.9</td>
<td>74.6</td>
<td>81.5</td>
</tr>
<tr>
<td>University of Colorado</td>
<td>82.9</td>
<td>85.1</td>
<td>82.4</td>
<td>85.2</td>
<td>91.3</td>
<td>66.3</td>
<td>79.7</td>
</tr>
<tr>
<td>Lockheed Martin</td>
<td>68.3</td>
<td>69.1</td>
<td>73.6</td>
<td>73.2</td>
<td>59.5</td>
<td>55.6</td>
<td>66.2</td>
</tr>
</tbody>
</table>

Table 2.3 Entity Mentions (Top 5 of all participants)

<table>
<thead>
<tr>
<th>Site</th>
<th>Overall</th>
<th>Broadcast Conversations</th>
<th>Broadcast News</th>
<th>Newswire</th>
<th>Telephone</th>
<th>Usernet Newsgroups</th>
<th>Weblogs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRA team #1</td>
<td>71.9</td>
<td>72.7</td>
<td>77.1</td>
<td>72.8</td>
<td>62.9</td>
<td>61.5</td>
<td>67.6</td>
</tr>
<tr>
<td>BBN Technologies</td>
<td>71.7</td>
<td>71.8</td>
<td>75.4</td>
<td>72.2</td>
<td>67.7</td>
<td>59.7</td>
<td>71.6</td>
</tr>
</tbody>
</table>

---

7 types and 44 subtypes of name entities were defined in ACE program while only 3 types of name entities were defined in MUC 6 and MUC 7. The data corpus of ACE program is in the poor format (web blog, conversation, broadcast news, etc) comparing with the corpus of MUCs. Basically, the evaluation method is similar to F-measure except that a negative score will be applied when a wrong match is found in ACE evaluation method. Instead, a zero score will be assigned if a wrong match is found in F-measure evaluation system. More detailed scoring definition can be found in the appendix of ACE evaluation plan at the following address, http://www.nist.gov/speech/tests/ace/ace05/doc/ace05-evalplan.v2a.pdf

It shows one overall value for each participating system and one value while the system applied to each different corpus.
Table 2.4 Entities with coreference (top 5 of all participants)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SRA team #2</td>
<td>71.3</td>
<td>67.2</td>
<td>77.3</td>
<td>73.1</td>
<td>59.3</td>
<td>60.7</td>
</tr>
<tr>
<td>IBM</td>
<td>69.6</td>
<td>61.7</td>
<td>76.2</td>
<td>72.0</td>
<td>57.8</td>
<td>60.5</td>
</tr>
<tr>
<td>University of Colorado</td>
<td>68.5</td>
<td>67.2</td>
<td>73.7</td>
<td>72.7</td>
<td>65.1</td>
<td>50.2</td>
</tr>
</tbody>
</table>

Note: Using the Wilcoxon Signed Ranks test to compare system performance at the document level, no difference in system performance was found between SRA team #1, BBN Technologies, SRA team #2, and IBM, using a 5% p-value.

In ACE 2005, the event extraction task was redesigned; it introduced 33 subtypes of events, which were classified into 8 general types (Table 2.5). There were total of 35 roles defined for these events (Table 2.6)\(^{23}\)

<table>
<thead>
<tr>
<th>Types</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life</td>
<td>Be-Born, Marry, Divorce, Injure, Die</td>
</tr>
<tr>
<td>Movement</td>
<td>Transport</td>
</tr>
<tr>
<td>Transaction</td>
<td>Transfer-Ownership, Transfer-Money</td>
</tr>
<tr>
<td>Business</td>
<td>Start-Org, Merge-Org, Declare-Bankruptcy, End-Org</td>
</tr>
<tr>
<td>Conflict</td>
<td>Attack, Demonstrate</td>
</tr>
<tr>
<td>Contact</td>
<td>Meet, Phone-Write</td>
</tr>
<tr>
<td>Personnel</td>
<td>Start-Position, End-Position, Nominate, Elect</td>
</tr>
<tr>
<td>Justice</td>
<td>Arrest-Jail, Release-Parole, Trial-Hearing, Charge-Indict, Sue, Convict, Sentence, Fine, Execute, Extradite, Acquit, Appeal, Pardon</td>
</tr>
</tbody>
</table>

Table 2.5 Types and Subtypes of events defined in ACE 2005

---

\(^{23}\) Since this thesis work is based in part on ACE data, more details about this data will be provided in Chapter 3.
Since a portion of the ACE 2005 corpus was available for system training, researchers applied machine learning algorithms to identify events in text. These included the NYU system (Grishman, Westbrook, and Meyers 2005) and the system from University of Amsterdam (Anh 2006), both of which used a similar approach that we describe briefly here. The NYU system uses a combination of pattern matching and statistical modeling. The patterns are created from parsed training data. A pattern consists of the head nodes (with all the syntactic or semantic features) in the paths between the anchor verb and its argument phrases representing event roles. In addition, the following 4 classifiers were trained based on the maximum entropy model:

- **Argument classifier** – given an event anchor and an entity mention, this classifier decides whether the mention is an event argument
- **Role classifier** – given an anchor and an event argument, this classifier selects which role this argument plays in the event

<table>
<thead>
<tr>
<th>Allowable Event Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
</tr>
<tr>
<td>Seller</td>
</tr>
<tr>
<td>Artifact</td>
</tr>
<tr>
<td>Giver</td>
</tr>
<tr>
<td>Org</td>
</tr>
<tr>
<td>Instrument</td>
</tr>
<tr>
<td>Target</td>
</tr>
<tr>
<td>Prosecutor</td>
</tr>
<tr>
<td>Position</td>
</tr>
<tr>
<td>Time-After</td>
</tr>
<tr>
<td>Time-At-End</td>
</tr>
<tr>
<td>Time-Holds</td>
</tr>
</tbody>
</table>

**Table 2.6 Roles defined in the events of ACE 2005**
Event classifier – given an anchor and a list of arguments, this classifier decides whether this anchor and the list of arguments constitute an event mention.

Event coreference classifier – given two event mentions from the same document, this classifier decides whether they refer to the same event.

During an event extraction task, syntactic patterns are first used to match all potential candidate events in the test data corpus. The first three classifiers are then run over these event candidates to decide which are the correct event mentions and then to determine their arguments and roles. The last classifier merges mentions of the same event to create a more complete representation. While the NYU group made a good progress towards solving the outstanding IE problems still unsolved after the MUC conferences, further research is needed especially in pattern coverage area. Specifically, the NYU system derives all its patterns from a training corpus (and then trims this set as necessary), which obviously does not guarantee the coverage, especially when the application data changes, including the subject matter, information sources, and writing styles. In order to provide robust, dependable performance in practical applications, the systems must be able to automatically adapt to the changing data. It is the goal of this thesis to address and solve the adaptability problem and we chose semi-supervised bootstrapping as the key techniques to discover more patterns directly from the data. In the following section we discuss the current use of bootstrapping in IE research and practice.

2.4 Bootstrapping techniques in Information Extraction

As a semi-supervised machine learning method, bootstrapping can start either with a set of predefined patterns and with a collection of seed examples identified and annotated by a
human (a domain expert or a linguist)\textsuperscript{24} in a training data set. The initial set of patterns is applied to a new, un-annotated data set and the result is a number of positive instances that can be used for training. These instances occur in various linguistic contexts, which then serve as a basis to create additional patterns. These new patterns can be subsequently applied to the dataset to collect even more instances, some of which will be positive and some not. This iteration may repeat until no new patterns are found. So defined, bootstrapping has been widely used in Nature Language Processing research, including in Word Sense Disambiguation, Name Entity extraction, and Relation Extraction.

In Word Sense Disambiguation, Yarowsky (Yarowsky 1995) proposed a bootstrapping technique for iteratively finding sense indicators in the text surrounding ambiguous words (e.g. “life” seen near “plant” may indicate a biological plant rather than a manufacturing plant). The program starts with a small number of seed examples (as few as two) but it produces surprisingly good results (accuracy of 90.6\textsuperscript{25}).

In the first experiment of bootstrapping in IE area, (Strzalkowski and Wang 1996) demonstrated that bootstrapping techniques can dramatically improve the recall of named entity extraction without significantly sacrificing precision. Their technique, applied to organization names (and later to a broad category such as “product”) starts with a few “simple” high-precision, low-recall patterns (e.g. any capitalized phrase followed by “Co.”, such as “Henry Kauffman Co.”, is a company name). After a number of samples are extracted using these seed patterns, additional patterns are derived from the context surrounding these samples. For example, “The president of" in “The president of American Electric Automobile Co.”, may

\textsuperscript{24} If the initial set consists of seed examples, the patterns will be created from the seeds first
\textsuperscript{25} Accuracy is the frequency of it taking on the majority sense for the discourse, when the word occurs more than once in a discourse.
give rise to a new pattern “president of + CNAME”. The new patterns find more entities and
the process iterates until no further patterns are found. The technique achieved a very high
performance (95% precision and 90% recall), which encouraged more research in IE area by
using bootstrapping techniques. In Snowball, (Agichtein and Gravano 2000) applied
bootstrapping technique to relation extraction task. The seeds are provided to the system at
the beginning. Each seed consists of two arguments and a relation between them. For
example, “MICROSOFT” and “REDMOND” are linked by Organization-Location relation.
Given the seeds, this system collected contextual information before, between, and after the
two arguments, into 5-dimensional vectors (3 for the contexts plus 2 for the arguments). A
clustering algorithm, based on Vector Space Model, was then used to group similar vectors so
that one pattern could be created from each group. The new patterns are subsequently used to
extract more relation instances and the process is repeated until no new patterns or relations
are found.

Yangarber (Yangarber et al. 2000) described a bootstrapping method for document
classification. An initial set of patterns in the Subject-Verb-Object form are provided at the
beginning to split a set of documents into relevant and non-relevant classes based on whether
they contain any of the initial patterns. Then a new set of patterns from relevant document sets
is generated. Each new pattern is then applied to the entire set of documents and these that
match proportionally more relevant documents than non-relevant documents (Formula 2.4\textsuperscript{26}),
are retained for the next iteration.

\[
\frac{|H \cap R|}{|H \cap U|} \gg \frac{|R|}{|U|} \quad (2.4)
\]

\textsuperscript{26} H is the document set which contains the new pattern; R is the relevant document set; U is the non-
relevant document set.
Chapter 2

Bootstrapping is an attractive technique for event extraction because it helps to overcome the problem of manual design and costly tuning of complex set of rules, exceptions, exceptions to exceptions and so forth. In theory, bootstrapping, when properly implemented, should allow rapid adaptation of event extraction to new domains and for automatic tuning for high-recall/high-precision performance. However, the technique has thus been only applied to relatively structurally simple problems and not directly to the task of event extraction, which is substantially more complex than NE extraction. In this thesis we develop bootstrapping techniques suitable for application to event extraction, a very challenging task.

2.5 Pattern representations techniques in Information Extraction

Pattern based extraction is one of more popular approaches to IE tasks. Yangarber et al.’s (2000) method discussed in the previous section used the Subject-verb-object (SVO) patterns, which were based upon the events described in documents. In this model, the verb is the trigger of an event pattern, while its subject and its object are the only arguments considered. This model, which was primarily designed for document classification, is of course too simple for complex event extraction. For example, in the following reference to a Conflict-Attack event: “…a second straight night and into this morning, hundreds of people have been rioting in benton harbor.”, one of the key roles is the event location (benton harbor), which is neither the subject nor the object of the trigger verb (rioting); another key attribute, time of the event is not even in the same clause as the trigger (a second straight night and into this morning). In part to address these limitations, Sudo et al. (2001) proposed an extension to the SVO model called the chain model. Basically, a chain is a dependency path linking an event role to its dominant verb (the trigger). Using chains splits SVO patterns into SV patterns and VO patterns, but in the chain model, the role was not limited to subject/object relation with the verb. As a result, the chain
model can capture more information from the event descriptions than the SVO model. For example, if we apply the chain model to Attack event in the sentence, “Monday, Israeli soldiers fired on four diplomatic vehicles in the northern Gaza town of Beit Hanoun”, then both the Attacker chain: “<N(subj, PER): Attacker > <V(fire): trigger>” and the Target chain: “<V(fire): trigger> <Prep(on)> <N(VEH): Target>” will be generated.

While the chain model finds more role patterns than the SVO model, it does so at the expense of reduced pattern precision because the chain patterns are applied independently of one another. For example, the Attacker chain patterns: “<N(subj, PER): Attacker > <V(fire): trigger>” would match the sentence in Figure 2.3, even though no attack event is present, only a Personnel-End-Position event. To overcome this problem, Sudo et al (2003) further expanded the chain model to the subtree model. The main difference between the two models is the amount of information included in a pattern. In the chain model, only the nodes in the path between the trigger verb and the role phrase are included in the pattern; in the subtree model, however, other children of the trigger verb in the dependency parse tree are also placed in the pattern. With this extra context added to the pattern, its extraction accuracy is improved. Under the subtree model, the Attacker pattern discussed above becomes: “<N(subj, PER): Attacker > <V(fire): trigger> <Prep(on)> <N(VEH): Target>”. We note that this pattern will no longer match the Personnel-End-Position event in Figure 3.11 because of the added preposition context.

“Washington softball coach Heather Tarr confirmed Tuesday she has fired assistants Eve Gaw and Geoff Hirai.”

Figure 2.3 A Personnel-End-Position event
While the subtree model improved upon both the chain and SVO models, it tends to generate a very large number of spurious patterns that need to be filtered out thus adding significantly to the overall computational cost. Figure 2.4(a) shows an event on terrorism scenario. Figure 2.4(b) is the dependency tree of the sentence in Figure 2.4(a). In this event, “suicide bomber” is the Attacker, “heart of downtown Jerusalem” is the Place, and “himself and three other people”, and “scores” are the victim. Because five event roles are recognized in the event, five patterns will be generated within the chain model (Figure 2.4(c)). However, the subtree model would generate over a hundred patterns from the same example (Figure 2.4(c) shows a small part of them).

**Figure 2.4(a)** Example on Terrorism scenario

**Figure 2.4(b)** Dependency Tree of the example sentence (The entities to be extracted are shaded in the tree)

<table>
<thead>
<tr>
<th>Chain Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>(triggered (C-PERSON-SBJ))</td>
</tr>
<tr>
<td>(triggered (heart-IN (C-LOCATION-OF)))</td>
</tr>
<tr>
<td>(triggered (killing-ADV (C-PERSON-OBJ)))</td>
</tr>
<tr>
<td>(triggered (injuring-ADV (C-PERSON-OBJ)))</td>
</tr>
<tr>
<td>(triggered (C-DATE-ADV))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subtree Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>(triggered (C-PERSON-SBJ) (explosion-OBJ))</td>
</tr>
<tr>
<td>(triggered (explosion-OBJ) (C-DATE-ADV))</td>
</tr>
<tr>
<td>(killing (C-PERSON-OBJ))</td>
</tr>
<tr>
<td>(triggered (C-DATE-ADV) (killing-ADV))</td>
</tr>
<tr>
<td>(injuring (C-PERSON-OBJ))</td>
</tr>
</tbody>
</table>
Sudo (2003) used two scenarios to compare the performance of the subtree model and the chain model: the Management Succession (MS) scenario from MUC-6 (1995) about corporate managers assuming or leaving their posts, and the Murderer Arrest (MA) scenario, about arrests murder suspects. Their experiments show that for the MS scenario the subtree model significantly outperforms the chain model in precision; however, for the MA scenario both systems perform about the same. On the other hand, the increased precision of the sub-tree model comes at the expense of the lower recall as more complex patterns and less flexible patterns are created. For example, while the subtree pattern `<N(subj, PER): Attacker> <V(fire): trigger><Prep(at)> <N(VEH): Target>` matches the Attack event in “…pirates holding a hijacked ship off the coast of Somalia fired at one of its helicopters…” , it will not match the event in “…the Iraqis have fired sand missiles at aircraft…” , even though the structures of two events are quite similar. This explains why the more flexible pattern structure in the chain model better adapts to variations in event structure.

2.6 Conclusions

In this chapter, we presented a brief overview of IE research area and current techniques used in it. MUC and ACE provided great opportunities for researchers to evaluate and compare the
performance of their IE systems, and contributed to the progress in IE area. Newer systems depend less on human intervention, and utilize more statistical methods and machine learning. However, progress in event extraction has been slow because events have more complex structures and also a great variability of forms of reference, which makes development of appropriate training datasets exceedingly difficult. In chapter 2, we will describe our approach which overcomes this issue through bootstrapping technique applied to unannotated data.
Chapter 3

Chapter 3

PATTERN GENERATION AND VALIDATION

In this chapter, we describe the initial iteration of the BEAR system: the supervised learning from an annotated training data corpus. It will provide the basis for the unsupervised bootstrapping learning in later iterations. This chapter is organized as follows: First, we define the structure of the events used by BEAR. Next, we describe how event patterns are derived from annotated samples in training data set. We discuss in details the syntactic and semantic features that we collect in order to build these patterns. Further, we discuss how to validate these patterns and filter out those that we expect to have low accuracy. Finally, we evaluate event extraction accuracy of the remaining patterns using a set-aside test corpus.

3.1 Structure of an event

An event is something that happens at a given place and time\(^{27}\). It can be either one specific occurrence or an occurrence and its immediate outcome. These two options lead to different considerations what should be part of an event: the first is action-based and the other is scenario-based. A scenario-based event will contain a particular occurrence (e.g., an earthquake) and all subsequent occurrences that inevitably follow from it (e.g., people dying, property being damaged; rescue operations, etc.). A complete newspaper story will usually cover a scenario-style event. On the other hand, an action-based event is limited to a single occurrence and participants involved in it, while the subsequent occurrences are considered as separate, though possibly correlated events.

\(^{27}\)This definition is from one of the senses of event from Wordnet
“On the afternoon of May 12, 2008, an earthquake measuring 7.9 on the Richter scale hit Sichuan Province, a mountainous region in Western China. By the next day, the death toll stood at 12,000, with another 18,000 still missing...”

Figure 3.1 An earthquake event

Figure 3.1 shows an example, which can be considered a description of a scenario-based event. It includes an earthquake striking China recently (the happening) and the outcomes, 12,000 dead and 18,000 missing. The event templates defined for MUC evaluations are scenario-based (Sundheim 1991) (Onyshkevych, Okurowski, and Carlson 1993). A scenario-based event template provides room for complete specification of all relevant aspects of an event; however, a precise definition of what makes up a scenario event is difficult to come by. There are a lot of situations that may need to be considered in order to decide what is a part of a scenario. For example, the earthquake referred to in Figure 3.1 also caused a mountain collapse and landslide, but it is unclear whether these should be treated as part of the earthquake event. An earthquake may also trigger a volcano eruption or a tsunami, which may sometimes be treated as part of the earthquake event. Therefore, it is often quite tricky to precisely define the scope of a scenario-based event, while the definition action-based event is much easier.

Another drawback of scenario-based definition is the need to specify the internal structure of sub-events that make up a scenario. A scenario-event may be described across several sentences or even several paragraphs. Current information extraction techniques and machine learning are too limited to deal with such a level of complexity, therefore human involvement is necessary to hand craft extraction rules, a process that may take many months to complete for each new domain. This dramatically reduces the system portability and also its initial performance on a new domain.

---

28 This news was from The New York Times about an Earthquake happened in Eastern Sichuan, China in May 2008
On the other hand, for action-based definition, an event consists of only one occurrence so that it can be usually described within a single sentence. This dramatically simplifies event extraction. Although a single action-based event is unlikely to represent more than a fragment of a complete scenario, multiple action-based events may be linked together through various semantic relations between them to achieve completeness.

The example in Figure 3.1 contains references to least two action-based events: the Earthquake event and the Life-Die event, according to ACE event ontology discussed in the previous chapter. Action-based events can be automatically extracted, because current NLP techniques are quite robust at sentence level, including part-of-speech tagging (Toutanova et al. 2003), parsing (Charniak 1997) (Lin 1998), as well as some aspects of semantic analysis, such as named entity extraction (Miller et al. 1999) and (Borthwick et al. 1998).

In this thesis, we adopt action-based event definition and build a semi-supervised machine learning process that effectively exploits statistical and structural properties of natural language discourse in order to rapidly acquire rules, which can detect references to (or, mentions of) action-based events as well as other similar relationships in text. To demonstrate effectiveness of our algorithms, we use the dataset and tasks from ACE-2005 evaluations. This allows us to compare our results directly to the official scores obtained by other state-of-the-art event extraction systems, none of which is based on unsupervised training.

Below we provide brief definitions of some key concepts, to which we will often refer in this thesis.  

---

29 For more information, please go to ACE 2005 event annotation guide, http://projects.ldc.upenn.edu/ace/docs/English-Events-Guidelines_v5.4.3.pdf
Chapter 3

- **Event extent/span**, the scope of an event. It is assumed that events are described at sentence level. In other words, an entire description is always assumed to be contained within a single sentence.

- **Event Mention**, each description of or reference to an event in a document is called a mention of an event. An event may be mentioned multiple times in the same document.

- **Event Coreference**, when two event mentions refer to the same event, then we say that they corefer. For example, the following two mentions are coreferential, i.e., they both refer to the same historical event:

  “Maddux was killed in Philadelphia.”

  “…. Einhorn is accused of killing Maddux.”

- **Event Type and Subtype**, An event type indicates a broader class or domain to which this event belongs, e.g. Life, Conflict, Business, etc., while a subtype provides a more detailed classification of the event within each domain, e.g., Attack and Demonstrate are subtypes within the Conflict domain, while Injure and Die are subclasses inside the Life domain; similarly, Transfer-Ownership and Transfer-Money are subtypes in the Transaction domain, etc.

- **Event properties** specify certain additional attributes of an event. For example, the modality of an event is “assertion” if “the author or speaker makes reference to the event as though it were a real occurrence.” Similarly, the tense of an event is “past” if “the event occurred prior to the textual anchor time”.

- **Trigger**, is a key phrase that invokes a reference to an event of a certain type, e.g. the phrase “married” invokes a Life-Marry event (i.e., an event of type Life and subtype Marry) in “In 1927 she married William Gresser, a New York lawyer and musicologist.” It is the phrase, which most clearly signals an event occurrence in text.

- **Event specific attributes and general attributes**.
An event typically involves participants and other arguments whose selection and roles depend upon the type of event. For example, the Attacker in a Conflict-Attack event and the Defendant in a Justice-Sentence event are examples of event-type specific roles. There are also other entities or values, which are related to an event but do not count as the participants, for example, the location where the event occurred and the time when the event happened. These are usually called “general event attributes”, because they appear in most types of events. In this thesis, we will not distinguish event-specific attributes and general event attributes, and will treat them all uniformly.

For ACE English language event extraction task, there were seven broad event types defined each with a number of subtypes, resulting in thirty-three event subtypes. For all these events, there are total of thirty five event roles specified. This classification is based on a corpus of 1508 English language documents, which were manually annotated. Of these, 1198 documents made up a training corpus and 310 documents were set aside as a test corpus. The entire corpus contains 5624 annotated event mentions, of which 4160 are in the training corpus and the other 1390 in the test corpus. Comparing with the data sets used in the past, such as those used in MUC evaluations, the ACE data set is substantially larger and contains significantly more data both for training and testing. The corpus also contains a sufficient number of different types of events and it can be used to test the robustness and portability of participating systems.
<charseq START="2893" END="2908">Israeli soldiers</charseq>
</event_mention_argument>
<event_mention_argument REFID="APW_ENG_20030527.0232-E88-150" ROLE="Place">
<extent>
<charseq START="2947" END="2983">the northern Gaza town of Beit Hanoun</charseq>
</extent>
</event_mention_argument>
<event_mention_argument REFID="APW_ENG_20030527.0232-E87-149" ROLE="Target">
<extent>
<charseq START="2919" END="2963">four diplomatic vehicles</charseq>
</extent>
</event_mention_argument>
<event_mention_argument REFID="APW_ENG_20030527.0232-T14-1" ROLE="Time-Within">
<extent>
<charseq START="2885" END="2890">Monday</charseq>
</extent>
</event_mention_argument>
</event_mention>

Figure 3.2 An annotated event mention from the ACE corpus

Figure 3.2 is an example of a text annotated for one event mention, using the XML format adopted for ACE. The text describes an attack event: the time and the place where it happened, who was the attacker, and who was the target. As shown in Figure 3.2, each event is given a unique id and also assigned a number of properties, such as Type, Subtype, and Modality, etc. Furthermore, each event contains a list of mentions throughout the whole document that refer to this event. For each mention, there is a mention id and its start and end positions in the document. In the above example we have the following: the extent of the event mention is the entire sentence: Monday, Israeli soldiers fired on four diplomatic vehicles in the northern Gaza town of Beit Hanoun; the position in the document, START="2885" END="2983" is given in byte offsets; the anchor (trigger) of this mention is fired and its position is given as: START="2910" END="2914". The list of event arguments is also annotated so that for each argument, we have the value, e.g., Israeli soldiers is an argument at the position START="2893".

---

30 The original structure of an event is saved in XML format, which is hard to read.
and \textit{END}="2908" and it has the role "Attacker". Entities are also assigned unique id's and are further defined in a separate structure within the same file with events, as shown Figure 3.3\textsuperscript{31}.

Each entity contains a list of its mentions throughout the document.

\begin{verbatim}
<entity ID="APW_ENG_20030527.0232-E86" TYPE="PER" SUBTYPE="Group" CLASS="USP">
<entity_mention ID="APW_ENG_20030527.0232-E86-148" TYPE="NOM" LDCTYPE="BAR">
<extent>
<charseq START="2893" END="2908">Israeli soldiers</charseq>
</extent>
<head>
<charseq START="2901" END="2908">soldiers</charseq>
</head>
</entity_mention>
</entity>
\end{verbatim}

\textbf{Figure 3.3} An annotated entity in the ACE corpus

### 3.2 An extended representation of events

In order to support our objective of semi-supervised learning of event extraction, we require more information about an event structure than the roles and the trigger phrase; we also need to represent the linguistic context that links the trigger and roles together. In this section, we discuss how to combine syntactic and semantic features to obtain a more complete event representation.

#### 3.2.1 Syntactic representations

An event description can vary from very concise, newswire-style to very rich and complex as may be found in essays and other narrative forms. We need our system to be able to recognize any of these forms and to extract the basic template (or skeleton) of an event. A skeleton is the basic syntactic structure of the event that retains the heads of all key phrases while suppressing

\textsuperscript{31} For more detail information of entity definition, please refer the entity annotation guide, \url{http://projects.ldc.upenn.edu/ace/docs/English-Entities-Guidelines_v6.6.pdf}
the modifier structures inside them. The syntactic features of a phrase head that are of interest include the token word itself, its lemma, POS, and the syntactic relations with other phrase heads in the skeleton parse. Such structures cannot be obtained with keyword based analysis or linear word order processing of sentences (Agichtein and Gravano 2000) (Brin 1998), because these methods cannot distinguish a head from a modifier in a phrase. Therefore, we use Minipar (Lin 1998), a shallow parser, which is able to perform rapid syntactic analysis with a good accuracy. While Minipar only recognizes dependency relations between individual words in a sentence, the resulting structure is sufficient for our purposes, because the dependency relations always identify the head of a phrase, which is precisely what we need to build an event skeleton. Figure 3.4 shows a fragment of the dependency tree generated by Minipar from the sentence in Figure 3.2.

```xml
<TREE>
E3><><<U><><
... 2><Monday><><N><E1><mod
3><,><><U><E1><punc
4><Israeli><><A><5><mod
5><soldiers><soldier><N><6><s
6><fired><fire><V><E1><i
E4><><soldier><N><6><subj
7><on><><Prep><6><mod
8><four><><N><10><nn
9><diplomatic><><A><10><mod
10><vehicles><vehicle><N><7><pcomp-n
11><in><><Prep><10><mod
12><the><><Det><15><det
13><northern><><A><15><mod
14><Gaza><><N><15><nn
15><town><><N><11><pcomp-n
...
</TREE>

Figure 3.4 Node list from minipar
```

32 Precision is 88%; Recall is 80%; F-score is 83.8%. On a Pentium II 300 with 128MB memory, Minipar parses about 300 words per second.
Each line in Figure 3.4 is a 6-tuple, which represents one node in the parse tree with elements separated by “><”,

Node ID><Token/Lemma><POS><Head><Dependency Rel

where the elements are defined as follows:

- **Node id**: if a node represents a word (head or modifier) in the parsed sentence, then its id is a unique ordinal number. For the nodes which represent non-terminal elements, such as a clause, the node id starts with an “E”

- **Token and Lemma**, the word represented by the node.

- **Part of Speech Tag (POS)**, lexical category of a word, for example, noun (N), verb (V), Determiner (Det), etc.

- **Head**, the node id for which the current word is a modifier

- **Dependency Relationship** indicates the type of dependency between the current node and its head: subj (subject), obj (object), adjn (adjunct), empl (complement), spec (specifier), etc.
Figure 3.5 shows the tree structure generated by our system from the list of tuples in Figure 3.4. All modifier nodes are linked up as children to their phrase head nodes and these are subsequently linked up to larger phrase or clause heads, etc. making up the entire dependency tree. Removing all non-head nodes from this tree produces the “skeleton” dependency structure shown in Figure 3.6.
Thus the example sentence in Figure 3.3 after being stripped down to its “skeleton” becomes, in effect, “Monday, soldiers fired on vehicles in town”. Thus trimmed parse tree dramatically simplifies the representation of the event without losing the basic argument dependency structure, which is the core of our pattern representation.

3.2.2 Semantic representations

The syntactic dependency representation provides the structure necessary for locating references to events in text. However, syntactic dependency relations alone are not sufficient to determine the type of event mentioned in a sentence. For example, the syntactic dependency structure of an attack event, “Military officials say a missile hit his warthog and be was
forced to eject.”), is [(subj) hit (obj)], which can match many non-Attack events, such as “The car hit a tree” and “An interesting idea hit her”. But if we can augment this syntactic structure with additional semantic features, such as the entity type of each event role to enrich the dependency structure, it will become a more accurate representation for the target type of event. Thus, the type of entity for “a missile” is annotated as Weapon (WEA) in ACE data and the type of “his warthog” is Vehicle (VEH). The enriched dependency structure of the above example will be, [(WEA, subj) hit (VEH, obj)]. With constrains provided by the entity types, the new structure will not match examples with a non-WEA subject or a non-VEH object; thus the accuracy of this pattern significantly increases. Nonetheless, the structure does not tell us which sense of hit (out of 17 different senses of the verb hit defined in Wordnet) is required for this pattern to match, which limits its applicability. If the correct sense can be determined, we would be able to use its synonyms and hyponym as alternative event triggers, thus enabling extraction of more events. This, in turn, requires sense disambiguation to be performed on the event triggers.

In MUC evaluations, participating groups (Yangarber and R. Grishman 1998) often used human experts to decide the correct sense of event triggers and then manually added correct synonyms to generalize event patterns. Human-generated synonyms make accurate extraction patterns, but the process is very time consuming and the patterns are not portable to new domains. In BEAR, we navigate through Wordnet (Fellbaum 1998) to automatically establish the desired sense and appropriate synonyms of each instance of event trigger. Obviously, just looking up a trigger word in Wordnet is not sufficient; we also need to select the correct sense from the synset since not every sense of a word can function as a trigger for an event of the right type. One approach is simply to select the most likely sense, which is typically the first
Chapter 3

sense of a word defined in Wordnet (Ahn, 2006). This is possible because the synsets in Wordnet are ordered by frequency occurrence in a general English text corpus. Unfortunately, this approach is not very reliable, since the required sense of a trigger is not necessarily the most common one in the Wordnet synset. For example, Figure 3.7 shows the senses of the verb *launch* listed in Wordnet. The second sense best matches the LAUNCH event defined in MUC 7: this sense of launch may be less common in “ordinary” usage, but is more frequent in MUC military domain.

![Figure 3.7](image)

We developed a new approach for utilizing Wordnet to decide the desired sense of an event trigger. To determine the correct sense of the trigger word, we examine the immediate lexical context of other words occurring nearby. This will tell us, for instance, if the word *strike* means an attack *(is target, or missile nearby?)* or an agreement *(is deal, or sign nearby?)*? In order to automatically compile the list of desired senses for each trigger, we utilize both Wordnet and an annotated training text corpus, with a representative set of event mentions of a given type, such as ATTACK or INJURE. From Wordnet, we extract all senses for the trigger words in each of the event mention and then search for the shared senses among them. This process is described in the following steps:

<table>
<thead>
<tr>
<th>Sense</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>establish, set up, found, launch</strong></td>
<td>(set up or found; &quot;She set up a literacy program&quot;)</td>
</tr>
<tr>
<td>2. <strong>launch</strong></td>
<td>(propel with force; &quot;launch the space shuttle&quot;; &quot;Launch a ship&quot;)</td>
</tr>
<tr>
<td>3. <strong>launch</strong></td>
<td>(launch for the first time; launch on a maiden voyage; &quot;launch a ship&quot;)</td>
</tr>
<tr>
<td>4. <strong>plunge, launch</strong></td>
<td>(begin with vigor; &quot;He launched into a long diatribe&quot;; &quot;She plunged into a dangerous adventure&quot;)</td>
</tr>
<tr>
<td>5. <strong>launch, set in motion</strong></td>
<td>(get going; give impetus to; &quot;launch a career&quot;; &quot;Her actions set in motion a complicated judicial process&quot;)</td>
</tr>
<tr>
<td>6. <strong>launch</strong></td>
<td>(smoothen the surface of; &quot;float plaster&quot;)</td>
</tr>
</tbody>
</table>

*Figure 3.7* the senses of *launch* (as verb) listed in Wordnet
1) For each type of event, we first collect all triggers from the annotated event mentions in the training corpus. Each trigger specifies the lemma of a word used, its POS, the type of event, the sense of the trigger (initially empty).

2) We send this trigger list to Wordnet and obtain all possible senses for each trigger, taking into account its POS. For triggers with only one sense is returned, this sense is directly assigned to the trigger; for all other triggers, i.e., when multiple senses returned, we proceed to step 3.

3) We now order the trigger list by the trigger frequency \( TF(t, w_{\text{pos}}) \), which is calculated by dividing number of times each word \( w_{\text{pos}} \) is used as a trigger for the event of type \( t \) by the total number of times this word occurs in the training corpus. Clearly, the greater trigger frequency of a word, the more discriminative it is as a trigger for the given type of event.

4) From the top of the trigger list, we select the first trigger (Tr1) that does not have its sense assigned yet, and process step 5.

5) Again, beginning from the top of the trigger list, for every trigger Tr2 different than Tr1 we look for a pair of compatible senses between Tr1 and Tr2. To do so, we look up both words in Wordnet and then traverse Synonym, Hypernym, and Hyponym links starting from Tr1 and see whether there are paths which can reach the senses of Tr2. If such converging paths exist, they identify compatible senses of Tr1 and Tr2. These senses are assigned to their respective triggers and we go back to step 4. The objective of this step is to assign a unique sense to each trigger. This process is captured in the following pseudo-code:

```java
for (int j = 0; j < triggers.size(); j++) {
    Tr2 = triggers.get(j);
    if (!Tr1.senseDefined() && !Tr2.senseDefined()) {
        for (sc1 in trigger1.sense_candidates && sc2 in trigger2.sense_candidates) {
            if (Wordnet.isSynonym/isHyponym/isHypernym(sc1, sc2)) {
                trigger1.setSense(sc1);
            }
        }
    }
}
```

33 \( t \) – the type of the event, \( w_{\text{pos}} \) – the lemma of a word and its POS. So \( TF(t, w_{\text{pos}}) \) means the frequency of a word in POS used as a trigger in the type of event.
Chapter 3

```java
trigger2.setSense(sc2);
}
}
}
if (Tr2.senseDefined()) {
    for (sc1 in trigger1.sense_candidates) {
        if (Wordnet.isSynonym/isHyponym/isHypernym(sc1, Tr2.sense)) {
            trigger1.setSense(sc1);
        }
    }
}
}
```

6) After we loop through step 4 and 5, if any triggers remain in the list that still do not have a sense assigned, we select their most commonly used sense, which is the first sense listed in Wordnet.

Here is one example to illustrate how this algorithm works. The verb *hit* is a trigger of an attack event in “Military officials say a missile *hit* his warthog and he was forced to eject.”; while *attack* is a trigger in “Four U.S. Army soldiers were killed when a suicide bomber *attacked* a military checkpoint today in Najaf”. Our hypothesis is that since both sentences refer to an ATTACK event, these two trigger words, *attack* and *strike*, must be used in compatible senses, and that these are the senses we wish to identify in Wordnet. Indeed, going through the links in Wordnet, we find that *hit* in sense 9 is the hyponym of sense 1 of *attack* (Figure 3.8). By linking these two senses, we select sense 1 of *attack* (out of 6 possible senses) and sense 9 of *hit* (out of 17 possible senses) as verb triggers for the Attack event. These two senses can now be saved, along with the word and its part-of-speech and the event type invoked in the system knowledge base (Figure 3.9); they will be utilized to identify event triggers with high accuracy in the running text and also to find additional triggers for the same event type.
Sense 1
attack, assail

=> submarine
=> pepper, pelt
=> **strike, hit** (sense 9)
=> counterattack, counterstrike
=> gas
=> storm, surprise
=> blitz
=> invade, occupy
=> besiege, beleaguer, surround, hem in, circumvent
=> bombard, bomb
=> strafe
=> cannonade
=> torpedo
=> raid, bust

**Figure 3.8** the semantic relation between *attack* and *strike*

---

**content:** hit  
**POS:** V  
**type:** Conflict_Attack  
**sense number:** 9  
**sense is:**  
[Synset: [Offset: 1088878] [POS: verb] Words: strike, hit -- (make a strategic, offensive, assault against an enemy, opponent, or a target; "The Germans struck Poland on Sept. 1, 1939"; "We must strike the enemy's oil fields")]

**sense relationship:** hyponym

**sense relationship with:** attack

---

**content:** attack  
**POS:** V  
**type:** Conflict_Attack  
**sense number:** 1  
**sense is:**  
[Synset: [Offset: 1084194] [POS: verb] Words: attack, assail -- (launch an attack or assault on; begin hostilities or start warfare with; "Hitler attacked Poland on September 1, 1939 and started World War II"; "Serbian forces assailed Bosnian towns all week")]

**sense relationship:** hypernym

**sense relationship with:** strike

**Figure 3.9** the senses of two attack trigger words saved in BEAR KB
This algorithm assigns a single sense to every event trigger; from now on the trigger words will indicate event types in text only if they are used in their designated senses. Specifically, different senses of a homograph may trigger different event types. For example, *fire* can indicate either an Attack event or a Personnel End-Position event, depending upon which sense it is being used (figure 3.10).

<table>
<thead>
<tr>
<th>content: fire</th>
<th>POS: V</th>
<th>type: Conflict_Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>sense number: 1</td>
<td></td>
<td>sense is:</td>
</tr>
<tr>
<td>sense relationship: hypernym</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sense relationship with: strike</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>content: fire</th>
<th>POS: V</th>
<th>type: Personnel_End-Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>sense number: 4</td>
<td></td>
<td>sense is:</td>
</tr>
<tr>
<td>sense relationship: synonym</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sense relationship with: dismiss</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.10** different senses found for one phrase used as triggers in different type of events

Our approach makes the process of identifying references to tracked events significantly more accurate. Furthermore, finding new triggers (i.e., triggers not in the training corpus) by following synonym/hypernym/hyponym links from already known triggers is now more precise because we are following links between individual senses rather than allowing all the senses, as was done in the past. A detailed discussion of how to acquire additional triggers follows in Chapter 4.

### 3.3 Pattern generation
In section 3.2, we discussed how to represent an event using a syntactic structure with semantic features. In this section, we will explain how these features can be utilized to create event extraction patterns.

Usually, an event of a given type contains a fixed set of roles, i.e., arguments that play specific semantic function in the event, e.g. Attack event has Attacker, Target, Instrument, Time, and Place, but each role is not necessarily specified in every event mention. For example, the Attack event described in the sentence, “Monday, Israeli soldiers fired on four diplomatic vehicles in the northern Gaza town of Beit Hanoun”, has no Instrument specified. This means that while we can induce partial event patterns based on the examples (seeds) in the training corpus, we need to make allowances to adapt them later to yet unseen event mentions. Our approach is, therefore, to create patterns as collections of quasi-independent role sub-patterns that can be added and modified as needed, while an event-level set of constraints is observed. This way it is going to be easy to plug in or out sub-patterns in order to adapt event extraction to new event mentions in the test corpus. To create a role sub-pattern, we use a mechanism similar to the chain model (Sudo et al., 2001), that is, we map the dependency paths that connect the trigger to the role heads. The dependency path contains syntactic relations, such as POS and phrase dependencies, and semantic representations, such as entity type and sense selections for the trigger. In addition to the role sub-patterns, an event pattern also contains the following elements:

- Event Type and Subtype
- Trigger – an instance of the trigger must be found in text in order for the pattern to be considered. It contains word lemma, POS, and the assigned sense.
Project Accuracy score – reflecting the percentage of events that were correctly extracted by this pattern in the past

Context profile – additional features collected from the text around the event description. At this time, the context profile includes references of other types of events often found near this event (either in the same sentence, same paragraph, or adjacent paragraphs).

<table>
<thead>
<tr>
<th>Pattern id: 490</th>
<th>From: sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projected Accuracy: null</td>
<td>Type: Conflict</td>
</tr>
<tr>
<td>Subtype: Attack</td>
<td>Trigger: fire_V_1</td>
</tr>
</tbody>
</table>

**Attacker pattern:** <N = Attacker(subj, PER)> <V(fire): trigger>

**Place pattern:** <V(fire): trigger> <Prep> <N> <Prep(in)> <N(GPE): Place>

**Target pattern:** <V(fire): trigger> <Prep(on)> <N(VEH): Target>

**Time-Within pattern:** <N(time2): Time-Within> <E1> <V(fire): trigger>

Figure 3.11 A pattern derived from an annotated event mention

Figure 3.11 shows a pattern created from the Attack event description in Figure 3.6. In order to obtain sub-patterns we generalize trigger-role paths using the following criteria:

- If a node in the path is a trigger, then its lemma and POS will be used in the sub-pattern.

- If a node in the path is an event role, then its entity type, syntactic role (subject, object), and POS will be used in the sub-pattern.

- If a node in the path is a preposition head of a role phrase, then this preposition will be placed directly in the sub-pattern along with its POS. [NB: head prepositions are good role indicators, e.g. *from* indicates the Origin in a Movement event.]

- For all other nodes, only their POS’s will be used in the sub-pattern.
Some other pattern elements, including Projected Accuracy Score and Context, will be calculated at a later stage when the pattern accuracy is measured (A detailed description of this process follows in section 3.4.)

Once a complete pattern is defined, it can be applied to the event extraction task, at which time we require that all its sub-patterns match in order to count a new event mention. This is required so that our pattern can accurately distinguish events from other non-event information in text. This is one advantage of patterns created at the event level; however, the high precision of our patterns may easily be offset by diminished recall. A pattern may miss many relevant event mentions where only some roles are specified, because we require that all role sub-patterns are matched. For example, the sentence “In 2001, four known rockets were fired on Israeli targets inside the Gaza Strip” describes a Conflict-Attack event; however, it would not match the pattern in Figure 3.11, because the Attacker role in the pattern is not specified in text. In order to match the event in this sentence a new, slightly revised pattern is needed. It should be noted that we do not attempt to relax or further generalize the patterns, for example, by making some roles optional; rather, we aim to circumscribe each new example with a closely fitting pattern. This way we guard our system from over-generation that plagues most manually build patterns. One way to recognize that new patterns are needed is by allowing the role sub-patterns in the existing patterns to match independently of each other, without yielding positive event identification. Upon detecting a close partial match (a near miss), new patterns may be constructed automatically by removing or replacing unmatched sub-patterns until they fit the new syntactic form. Such “pattern mutation” techniques will be described in detail in the next chapter.
Chapter 3

3.4 Pattern validation

The patterns in our system are automatically generated from the training text corpus; it is therefore necessary to validate their projected accuracy before they can be used on a new data set. Basically, we apply all automatically generated patterns back to training corpus and measure the rate at which they can correctly extract event mentions.

Let’s consider the following example: “7:05 a.m., u.s. central command says another top baath party official has been captured”, which describes a Justice-Arrest-Jail event; the pattern generated from this example is shown in Figure 3.12. While this pattern seems reasonable given the source sentence, it is far too generic to be included with other patterns already in the system: if used, the resulting pattern set would produce the false alarm rate as high as 82%!

| Pattern id: 27 |
| from: sample |
| Projected Accuracy: null |
| Type: Justice |
| Subtype: Arrest-Jail |
| Trigger: capture |
| Person pattern is: <N(obj, PER): Person> <V(capture): trigger> |

The validation mechanism we use consists of two steps: the first step is to verify the quality of automatically generated patterns and the second (optional) step is to introduce contextual and entity type constraints in an attempt to repair patterns that do not pass the first validation step. The quality of each pattern is verified along the following dimensions: the projected accuracy of trigger matching; the projected accuracy of each role sub-pattern; and the projected accuracy of the whole pattern. We use the projected accuracy of the whole pattern as the

---

34 We only discuss validation through training corpus in chapter. The validation on test corpus will be discussed in Chapter 4.
criterion for retaining patterns in the system; the projected accuracy of triggers is calculated but will only be used in latter iterations to help finding new patterns; the projected accuracy of sub-patterns is also calculated and is used to improve projected accuracy of event pattern as well as to find new patterns in latter iterations. The projected accuracy of the pattern is a ratio of positive event matches to all matches against the validation corpus. Please note that in the supervised learning step the validation corpus is the initial training corpus, i.e., either human annotated or obtained using manually designed seed patterns; in subsequent, unsupervised learning steps, the validation corpus also includes portions of the automatically annotated test corpus.

After the first validation step, we remove the patterns that do not pass a predefined accuracy threshold. These rejected patterns will undergo the second step of our validation process, where we examine the linguistic context of the event mentions that led to creation of these patterns and then introduce context-based constraints in order to repair these patterns. Another repair in the second step is to apply entity constraints to improve precision of sub-patterns and thus the pattern’s projected accuracy should also be improved. After a pattern is repaired its projected accuracy score is recomputed as before. If this score now rises over the threshold, the revised pattern will be returned to the system. In the rest of this section, we explain the validation tests in detail.

3.4.1 Validation of the event trigger

When a pattern is applied back to the training corpus for validation, the first thing we need to know is that how selective its trigger word or phrase is. In many cases, particularly early in the
learning process, many new events will have syntactic and semantic structures that differ from what is captured in the patterns discovered up to that point; therefore the trigger would be the most important factor in spotting these new events. Ideally, we would like a trigger to be highly selective, that is, its presence should signal the correct type of event with high probability (i.e., high precision).

<table>
<thead>
<tr>
<th>Trigger phrase</th>
<th>Sense number</th>
<th>Overall count</th>
<th>Trigger count</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Conflict-Attack event</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>attack_N</td>
<td>1</td>
<td>120</td>
<td>114</td>
<td>0.95</td>
</tr>
<tr>
<td>attack_V</td>
<td>1</td>
<td>38</td>
<td>28</td>
<td>0.74</td>
</tr>
<tr>
<td>bombing_N</td>
<td>1</td>
<td>54</td>
<td>51</td>
<td>0.94</td>
</tr>
<tr>
<td>war_N</td>
<td>1</td>
<td>332</td>
<td>284</td>
<td>0.855</td>
</tr>
<tr>
<td>shoot_V</td>
<td>3</td>
<td>23</td>
<td>15</td>
<td>0.652</td>
</tr>
<tr>
<td>hit_V</td>
<td>9</td>
<td>49</td>
<td>23</td>
<td>0.469</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Life-Die</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>murder_V</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>0.909</td>
</tr>
<tr>
<td>murder_N</td>
<td>1</td>
<td>35</td>
<td>17</td>
<td>0.486</td>
</tr>
<tr>
<td>kill_V</td>
<td>1</td>
<td>183</td>
<td>164</td>
<td>0.896</td>
</tr>
<tr>
<td>Kill_N</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>0.833</td>
</tr>
<tr>
<td>casualty_N</td>
<td>3</td>
<td>20</td>
<td>9</td>
<td>0.45</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Movement-Transport</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>travel to_V</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>1.0</td>
</tr>
<tr>
<td>arrive_V</td>
<td>1</td>
<td>30</td>
<td>23</td>
<td>0.767</td>
</tr>
<tr>
<td>move in_V</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>0.667</td>
</tr>
<tr>
<td>deploy_V</td>
<td>1</td>
<td>17</td>
<td>9</td>
<td>0.529</td>
</tr>
<tr>
<td>send_V</td>
<td>1</td>
<td>79</td>
<td>24</td>
<td>0.304</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1 The sense and selectivity of event triggers

Table 3.1 lists some representative trigger phrases and their selectivity for three event types. The first column lists the trigger phrase (and its POS); the second column specifies its sense number (based on Wordnet); the third column is the number of occurrences in training corpus; the fourth column is the number of times the phrase was used as a trigger; and the fifth
column is its selectivity, which is the ratio of the counts in columns 3 and 4. From the table we can see that some triggers selectivity is very high. For example, 95% of attack_N occurring in the training corpus is contained within Attack event mentions. An occurrence of a trigger with such a high selectivity is a strong indicator of an event mention. For other trigger phrases their selectivity may be quite low. For example, only 30% of send_V occurrences points to a Transport event. For low selectivity triggers, additional contextual information is needed to decide whether an event is present.

### 3.4.2 Validation of sub-patterns

Validating sub-patterns is part of the overall pattern validation; however, we also use this process to identify high accurate sub-patterns that can be used to build new patterns in later iterations. To validate a sub-pattern we apply it to the training corpus and calculate its projected accuracy score by dividing the number of correctly matched roles by the total number of matches returned. The projected accuracy score will tell us how well a sub-pattern can distinguish a specific event role from other information, when used independently from other elements of the complete pattern.

<table>
<thead>
<tr>
<th>Victim pattern: &lt;N(obj, PER): Victim&gt; &lt;V(kill): trigger&gt;   (Life-Die)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projected Accuracy: 0.9390243902439024</td>
</tr>
<tr>
<td>Number of negative matches: 5</td>
</tr>
<tr>
<td>Number of Positive matches: 77</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attacker pattern: &lt;N(sub, GPE/PER/ORG): Attacker&gt; &lt;V&gt; &lt;V(use): trigger&gt; (Conflict-Attack)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projected Accuracy: 0.025210084033613446</td>
</tr>
<tr>
<td>Number of negative matches: 116</td>
</tr>
<tr>
<td>Number of positive matches: 3</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
**Chapter 3**

**Attacker pattern:** `<N(subj, GPE/PER): Attacker> <V(attack): trigger> (Conflict-Attack)`

**Projected Accuracy:** 0.4166666666666667

**Number of negative matches:** 7

**Number of positive matches:** 5

| Categories of positive matches | GPE: 4  
|                              | GPE_Nation: 4  
|                              | PER: 1  
|                              | PER_Individual: 1  |
| Categories of negative matches | GPE: 1  
|                              | GPE_Nation: 1  
|                              | PER: 6  
|                              | PER_Group: 1  
|                              | PER_Individual: 5  |

**Figure 3.13** sub-patterns with projected accuracy scores

Figure 3.13 shows three sub-pattern examples. The first sub-pattern extracts the *Victim* role in a *Life-Die* event with very high projected accuracy. This sub-pattern is also a good candidate for generations of additional patterns for this type of event, a process which we describe in Chapter 4. The second sub-pattern in Fig 3.13 was built to extract the *Attacker* role in *Conflict-Attack* events, but it has very low projected accuracy. This sub-pattern should not be used for extraction unless we can find contextual constraints that would limit its applicability. The third example shows another *Attacker* sub-pattern whose projected accuracy score is 0.417 after the first step in validation process. This is quite low; however, it can be repaired by constraining its entity type to GPE. This is because we note that with a GPE entity, the subpattern is 80% on target, while with PER entity it is 85% a false alarm. After this sub-pattern is restricted to GPE its projected accuracy becomes 0.8.

Table 3.2 lists example sub-patterns for which the projected accuracy increases significantly after adding more constrains. The first column lists the auto-generated sub-patterns; the second column is the projected accuracy after the first step of validation; the third column is the additional constraints applied to the sub-pattern by removing the entity types that cause...
the majority of false alarms; the forth column shows that the new projected accuracy with the entity type constraints. When the projected accuracy of a sub-pattern is improved, all patterns containing this sub-pattern will also improve their projected accuracy. If the adjusted projected accuracy rises above the predefined threshold, the repaired pattern will be saved.

<table>
<thead>
<tr>
<th>Sub-patterns</th>
<th>Projected Accuracy</th>
<th>Additional constraints</th>
<th>Revised Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Movement-Transport:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt;\text{N(obj, PER/VEH): Artifact}&gt; &lt;\text{V(send): trigger}&gt;)</td>
<td>0.475</td>
<td>removing PER</td>
<td>0.667</td>
</tr>
<tr>
<td>(&lt;\text{V(deploy): trigger}&gt; &lt;\text{N(obj, PER/VEH): Artifact}&gt;)</td>
<td>0.583</td>
<td>removing PER</td>
<td>0.714</td>
</tr>
<tr>
<td>(&lt;\text{V(bring): trigger}&gt; &lt;\text{N(obj)}&gt; &lt;\text{Prep = to}&gt; &lt;\text{N(FAC/GPE): Destination}&gt;)</td>
<td>0.375</td>
<td>removing GPE</td>
<td>1.0</td>
</tr>
<tr>
<td>(&lt;\text{V(take): trigger}&gt; &lt;\text{N(obj, VEH/PER/VEH): Artifact}&gt;)</td>
<td>0.345</td>
<td>removing VEH &amp; PER_Individual</td>
<td>0.474</td>
</tr>
<tr>
<td>(&lt;\text{N subj, ORG/PER/GPE: Agent}&gt; &lt;\text{V}&gt; &lt;\text{Prep}&gt; &lt;\text{V(send): trigger}&gt;)</td>
<td>0.235</td>
<td>removing PER &amp; GPE</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Conflict Attack:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt;\text{N(PER/ORG/GPE): Attacker}&gt; &lt;\text{N(attack): trigger}&gt;)</td>
<td>0.682</td>
<td>removing PER</td>
<td>0.8</td>
</tr>
<tr>
<td>(&lt;\text{N subj, GPE/PER): Attacker}&gt; &lt;\text{V(attack): trigger}&gt;)</td>
<td>0.417</td>
<td>removing GPE</td>
<td>0.8</td>
</tr>
<tr>
<td>(&lt;\text{N(obj, VEH/PER/FAC): Target}&gt; &lt;\text{V(target): trigger}&gt;)</td>
<td>0.364</td>
<td>removing PER_Individual</td>
<td>0.667</td>
</tr>
<tr>
<td>(&lt;\text{N subj, GPE/PER/ORG): Attacker}&gt; &lt;\text{V(fire): trigger}&gt;)</td>
<td>0.4</td>
<td>removing PER_Individual &amp; ORG</td>
<td>0.667</td>
</tr>
<tr>
<td>(&lt;\text{N subj, PER/GPE): Attacker}&gt; &lt;\text{V(destroy): trigger}&gt;)</td>
<td>0.182</td>
<td>removing GPE &amp; PER_Individual</td>
<td>0.429</td>
</tr>
</tbody>
</table>

Table 3.2 Sub-patterns which projected accuracy is significantly increased after noisy samples removed

3.4.3 Validation of the whole event pattern
We calculate pattern scores based on the ratio of fully and correctly extracted events in the training corpus. For each extracted event candidate, one of the following may obtain:

- **A Full match** is achieved when the event type is correctly identified and all its roles are correctly matched. A full credit is added to the pattern score.
- A Partial match is achieved when the event type is correctly identified but only a subset of roles is correctly extracted. A partial score, which is the ratio of the matched roles to the whole roles, is added.
- A False Alarm occurs when a wrong type of event is extracted (including when no event is present in text). No credit is added to the pattern score.

The projected accuracy score of a pattern is the sum of collected match credits divided by the total number of matches returned by this pattern. The first example in Figure 3.17 shows a pattern for the *Conflict-Attack* event. When this pattern is applied to our training corpus during the validation process, all events it extracts are Attack events, so the projected accuracy of this pattern is 1.0. However, the second pattern in Fig 3.14 has a very low projected accuracy; it is removed from the system because it would cause significant over-generation during extraction.

After the first step of validation, only the patterns with projected accuracy over a certain threshold will be retained for event extraction. This threshold can be defined depending upon the application, i.e., whether the focus is on precision or recall. If we want more precision we can set the threshold high; otherwise, we can lower the threshold to have more relevant events extracted, but at a lower precision.

---

**Pattern id: 155**  
**From:** sample  
**Projected Accuracy:** 1.0  
**Type:** Conflict  
**Subtype:** Attack  
**Trigger:** hit_V_9

---

35 Event candidates are those matched by event patterns but still subject to verification.
Chapter 3

<table>
<thead>
<tr>
<th>Attacker pattern:</th>
<th>&lt;N(subj, WEA): Instrument&gt; &lt;V(hit): trigger&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target pattern:</td>
<td>&lt;V(hit): trigger&gt; &lt;N(obj, VEH): Target&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern id: 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>from: sample</td>
</tr>
<tr>
<td>Projected Accuracy: 0.1765</td>
</tr>
<tr>
<td>Type: Justice</td>
</tr>
<tr>
<td>Subtype: Arrest-Jail</td>
</tr>
<tr>
<td>Trigger: capture_V_3</td>
</tr>
</tbody>
</table>

Person pattern is:  
<N(obj, PER): Person> <V(capture): trigger>  

Figure 3.14 a pattern from sample event

For the patterns where the projected accuracy score falls under the cutoff threshold, we may still be able to make some “repairs” by taking into account their context profile. To do so we collect all the matches produced by such a failed pattern and create a list of all other events that occur in their immediate vicinity: in the same sentence, as well as the sentences before and after it. These other events, of different types and detected by different patterns, may be seen as co-occurring near the target event: these that co-occur near positive matches of our pattern will be added to the positive context support of this pattern; conversely, events co-occurring near negative (false) matches will be added to the negative context support for this pattern. By collecting such contextual information, we can find contextually-based indicators and dis-indicators for occurrence of event mentions. When these extra constraints are included in a previously failed pattern, its projected accuracy is expected to increase, in some cases above the threshold.

Event id: 27  
from: sample  
Projected Accuracy: 0.1765  
Adjusted Projected Accuracy: 0.91  
Type: Justice  
Subtype: Arrest-Jail  
Trigger is: capture  

---

36 If a known event is detected in the same sentence (sent_…), the same paragraph (para_…), or an adjacent paragraph (adj_para_…) as the candidate event, it becomes an element of the pattern context support. 

61
Person sub-pattern: <N(obj, PER): Person> <V(capture): trigger>

Co-occurrence ratio: {para_Conflict_Demonstrate=100%, ...}
Mutually exclusive ratio: {sent_Conflict_Attack=100%, para_Conflict_Attack=96.3%, ...}

Figure 3.15 An Arrest-Jail pattern with context profile information

For example, the pattern in Figure 3.15 has an initially low projected accuracy score; however, we find that positive matches of this pattern show a very high (100% in fact) degree of correlation with mentions of Demonstrate events. Therefore, limiting the application of this pattern to situations where a Justice-Arrest-Jail event is mentioned in a nearby text improves its projected accuracy to 91%, which is well above the required threshold.

3.5 Order in applying patterns

During pattern validation (as well as in later evaluation) an application order issue arises when multiple patterns match the same fragment of text, and the system must decide which pattern should be applied. To address this issue in BEAR we established the following criteria for pattern ranking:

1) [Use the More Specific Pattern] If one pattern contains all sub-patterns of another pattern, then we use the one that has more sub-patterns for extraction.

2) [Use the More Accurate Pattern] Otherwise, select the pattern with the higher projected accuracy score (if available).

3.6 Experimental Results – first iteration

Up until now, we described how to represent events in text, how to create patterns to extract them, and how to validate these patterns against a training data set. These steps comprise the first iteration of our bootstrapping system: i.e., building high-quality initial patterns (high
precision, but possibly low recall). These initial patterns can now be applied to an unannotated text corpus to extract a number of event mentions, from which the system can learn how to derive new patterns in the next iteration. Before we proceed we wanted to make sure that the patterns learned in the first iteration are indeed of high quality. The evaluation experiment described in this section was conducted to verify if the pattern validation process supports selection of sufficient quality patterns. We first describe the dataset and software components used in the experiment; then we discuss experiment mechanics; at finally we analyze the results.

### 3.6.1 Experimental dataset

For this experiment we used the data corpus created for the ACE 2005 evaluation. Table 3.3 shows the sources and size of ACE 2005 corpus. Currently, BEAR works only on English data. The English part of the ACE corpus includes references to 33 different event types from eight domains showed in Table 3.4. The corpus consists of a training set with 5482 annotated event mentions and a test set with 1390 annotated event mentions.

In order to evaluate how well our bootstrapping technique learns event extraction rules, we performed extraction experiments using ACE diagnostic tasks. This allows us to compare our performance to leading information extraction systems that participated in ACE evaluations.
Table 3.3 ACE training (left) and test (right) corpus

<table>
<thead>
<tr>
<th>Source</th>
<th>Training epoch</th>
<th>Approximate size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>English Resources</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Broadcast News</td>
<td>3/03 – 6/03</td>
<td>60,000 words</td>
</tr>
<tr>
<td>Broadcast Conversations</td>
<td>3/03 – 6/03</td>
<td>45,000 words</td>
</tr>
<tr>
<td>Newswire</td>
<td>3/03 – 6/03</td>
<td>60,000 words</td>
</tr>
<tr>
<td>Weblog</td>
<td>11/04 – 2/05</td>
<td>45,000 words</td>
</tr>
<tr>
<td>Usenet</td>
<td>11/04 – 2/05</td>
<td>45,000 words</td>
</tr>
<tr>
<td><strong>Conversational</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Telephone Speech</strong></td>
<td>11/04 – 12/04</td>
<td>45,000 words</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th><strong>Test epoch</strong></th>
<th>Approximate size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>English Resources</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Broadcast News</td>
<td>7/03 – 8/03</td>
<td>10,000 words</td>
</tr>
<tr>
<td>Broadcast Conversations</td>
<td>7/03 – 8/03</td>
<td>7,500 words</td>
</tr>
<tr>
<td>Newswire</td>
<td>7/03 – 8/03</td>
<td>10,000 words</td>
</tr>
<tr>
<td>Weblog</td>
<td>3/05 – 4/05</td>
<td>7,500 words</td>
</tr>
<tr>
<td>Usenet</td>
<td>3/05 – 4/05</td>
<td>7,500 words</td>
</tr>
<tr>
<td><strong>Conversational</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Telephone Speech</strong></td>
<td>11/04 – 12/04</td>
<td>7,500 words</td>
</tr>
</tbody>
</table>

Table 3.4 Types and Subtypes of events defined in ACE 2005

<table>
<thead>
<tr>
<th>Types</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life</td>
<td>Be-Born, Marry, Divorce, Injure, Die</td>
</tr>
<tr>
<td>Movement</td>
<td>Transport</td>
</tr>
<tr>
<td>Transaction</td>
<td>Transfer-Ownership, Transfer-Money</td>
</tr>
<tr>
<td>Business</td>
<td>Start-Org, Merge-Org, Declare-Bankruptcy, End-Org</td>
</tr>
<tr>
<td>Conflict</td>
<td>Attack, Demonstrate</td>
</tr>
<tr>
<td>Contact</td>
<td>Meet, Phone-Write</td>
</tr>
<tr>
<td>Personnel</td>
<td>Start-Position, End-Position, Nominate, Elect</td>
</tr>
<tr>
<td>Justice</td>
<td>Arrest-Jail, Release-Parole, Trial-Hearing, Charge-Indict, Sue, Convict, Sentence, Fine, Execute, Extradite, Acquit, Appeal, Pardon</td>
</tr>
</tbody>
</table>

3.6.2 Experiment metrics

BEAR is a self-learning system that makes multiple passes through the data to learn event extraction patterns that find as many event mentions as possible while maintaining the high precision from the initial supervised learning step and through subsequent unsupervised iterations. As we discussed, the patterns are first learned from the seed examples in the training corpus and then from the events automatically extracted after each iteration; therefore, maintaining high precision after each step is necessary to keep the bootstrapping process progressing in the right direction, i.e., to converge on the target event types. Our experiments
Chapter 3

examine the system performance at two levels, (1) the event detection level (discussed below) and (2) the argument extraction level (discussed in Chapter 5). The evaluation on the event level assesses how well the system can detect event types by spotting their mentions in raw text. The evaluation on the argument level measures how well the system extracts event arguments, including semantic roles.

We use Precision, Recall and F-measure standard metrics to evaluate the system performance. These metrics are defined as follows:

\[
Precision = \frac{|Extracted \ Events \cap \ Annotated \ Events|}{|Extracted \ Events|} \quad (3.1)
\]

\[
Recall = \frac{|Extracted \ Events \cap \ Annotated \ Events|}{|Annotated \ Events|} \quad (3.2)
\]

\[
F - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (3.3)
\]

3.6.3 Experiments and Evaluations

From the 5482 annotated event mentions in the training corpus, BEAR creates 1661 patterns and then calculates the projected accuracy scores of these patterns, sub-patterns, and triggers through validation back against the training corpus. Figure 3.16 shows the distribution of patterns based on their projected accuracy. On the X-Axis we group patterns into 10 accuracy “zones” from 0.9 to 0 based on the lower bound. The Y-Axis shows the number of patterns, which fall into each of the zones. More than 65% of BEAR initial patterns achieve high

---

37 Extracted events are the events extracted by BEAR; Annotated events are the events annotated by ACE annotators.
projected accuracy ($> 0.9$); these patterns will form the seeds for the bootstrapped learning in subsequent unsupervised iterations.

Figure 3.16 The Distribution of Patterns based on their projected accuracy

In order to test the actual performance of the retained patterns, we ran all generated patterns over the test corpus and extracted the total of 918 event mentions. Figure 3.17 shows the distribution of the extracted event mentions by the patterns in different projected accuracy zones. More than 72% of event mentions were extracted by patterns with either the highest ($\geq 0.9$) or the lowest ($< 0.1$) accuracy, while the patterns with mid-level accuracy (0.4 - 0.5) extract very few events. It is reasonable distribution, because the projected accuracy of more than half of initial patterns is greater or equal to 0.9 and this interval contributes a large portion of the correctly extracted events, while low score patterns extract mostly false alarms.
The events extracted from test corpus are checked against the human-annotated key to test the overall system performance. At this stage, we want to see whether our pattern validation process is predictive of pattern performance. We measured recall and precision at different cutoff thresholds based on the projected pattern accuracy. Figure 3.18 shows the trend of precision and recall while different pattern threshold is applied. The chart shows that most of the positive event mentions are extracted by the patterns generated with the projected accuracy above 0.5, and that further lowering of this threshold does not significantly increase recall. On the other hand, the overall precision steadily decreases while the threshold is lowered, and then drops off sharply as threshold is lowered below 0.1. From this chart it is not immediately clear what an optimal threshold should be. To see this, we created an F-score chart (Figure 3.19), which shows that the overall F-score is highest and quite stable between 0.6 and 0.1 thresholds. Based on this distribution, we select different projected pattern accuracy thresholds for the use in the unsupervised bootstrapping steps. This will let us observe how well BEAR performs under different start conditions. The bootstrapping process and the further evaluation experiments will be discussed in Chapter 4.
Chapter 3

Figure 3.18 The Precision and Recall of BEAR base run at different pattern projected accuracy thresholds

Figure 3.19 The F-score of BEAR base run at different pattern projected accuracy thresholds

3.7 Conclusions

In this chapter, we first discussed how to automatically create high-quality patterns from a set of annotated events. This procedure is applied to manually annotated data during the first supervised learning iteration discussed above. In Chapter 4 we will discuss how this procedure
can be also applied to automatically annotated data in subsequent unsupervised bootstrapping steps. We also introduced a novel technique to determine the correct sense of an event trigger using Wordnet traversal. Also we designed a new pattern structure, which consists of a trigger, a sequence of chain-based sub-patterns and a context profile that specifies additional constraints on pattern applicability based on co-occurrence of other events.

Another important issue we discussed in this chapter was the validation of event patterns, sub-patterns, and triggers. The validation process assigns an accuracy score to each pattern based on it extraction performance against the training corpus. When this score falls under a predetermined threshold, the pattern is removed from the system in order to maintain high precision of the overall extraction process. We also showed using evaluation against the test corpus that our validation process actually works well at estimating pattern performance. In the next Chapter we will show how this validation process can be generalized over the set of events extracted in subsequent bootstrapping iterations.

In the next Chapter, we discuss the process of automatically deriving new patterns from the events extracted in prior iterations. In this novel application of unsupervised bootstrapping the system automatically learns to adapt the existing patterns to unannotated data, as well as to derive entirely new patterns.
Chapter 4

UNSUPERVISED PATTERN LEARNING

As explained in previous chapters, the method underlying the BEAR system is to continue applying the patterns learned in previous iterations to a text corpus in order to extract additional event mentions from which to learn new patterns, and so forth until no new events are found or no new rules can be learned. The effect of this bootstrapping process is that the system rapidly learns, on its own, how to extract events from unannotated text corpora, including the corpora in new subject domains, without any human supervision. In this chapter, we discuss how BEAR achieves its learning objective following the derivation of seed patterns described in Chapter 3. We discuss two methods by which the system utilizes context features that help it to construct new patterns. The first method is to manipulate the structure of the already learned patterns in order to derive additional variants, or “mutated” patterns. The second method is to induce new patterns from previously extracted events by finding alternative ways to circumscribe the structure of these events, i.e., by exploiting structural redundancy in language. Finally, we describe evaluation experiments performed and discuss the results obtained.

4.1 New Patterns derived from known patterns

In this section we describe how new patterns can be automatically learned by refitting one or more existing patterns to previously unseen but structurally close variants of event mentions.

4.1.1 Finding new triggers for known patterns
One way to adapt an existing pattern is to expand its range so that it would match more events, for example, by allowing the synonyms of the initial trigger, when used in an appropriate sense, to be added to the pattern. The same expansion may also be considered for hyponyms and hypernyms. As an example, let’s consider a Conflict-Attack event as described in the following sentence:

“*He was* **shooting** an **AK-47**”

In this event, “*He*”\(^{38}\) is the Attacker and “**AK-47**” is the Instrument. This event is only partially matched by one existing pattern shown in Figure 4.1, which has been derived from similar (but not identical) annotated examples in the training corpus. This pattern produces a mismatch on the trigger: the example above uses verb *shoot* while the pattern expects *fire* (sense 1). Nonetheless, there is a semantic link in Wordnet between “*shoot* (sense 3)” and “*fire* (sense 1)” (Figure 4.2). As a result “*shoot*” will be considered an alternative trigger for *Conflict-Attack* and added to the pattern (Figure 4.3). In general, pattern expansion can add any synonym or hyponym of the current trigger; as well as a hypernym, but we require it to be an already known trigger (in another pattern) to block arbitrary generalizations. For example, the hypernyms of *bombing_N* (trigger of Attack event) is *attack_N* which then links to *operation_N* then to *activity_N*, and finally to *act_N*, which has little to do with Attack events.

---

**Pattern id:** 256  
**From:** sample  
**Projected accuracy:** 0.83  
**Type:** Conflict  
**Subtype:** Attack  
**Trigger:** fire_V_1  
**Attacker pattern:** <N(subj, PER): Attacker> <V(fire): trigger>  
**Instrument pattern:** <V(fire): trigger> <N(obj, WEA): Instrument>

---

\(^{38}\) In this thesis, we won’t discuss pronoun resolution
Figure 4.1 A partially matching pattern for attack event with Fire trigger

**Shoot**
- => blaze away, blaze
- => overshoot
- => sharpshoot, snipe
- => open fire, **fire**
- => gun
- => pump

Figure 4.2 Hyponyms of shoot from Wordnet

<table>
<thead>
<tr>
<th>Pattern id:</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>From:</td>
<td>new_trigger</td>
</tr>
<tr>
<td>Projected accuracy:</td>
<td>null</td>
</tr>
<tr>
<td>Type:</td>
<td>Conflict</td>
</tr>
<tr>
<td>Subtype:</td>
<td>Attack</td>
</tr>
<tr>
<td>Trigger:</td>
<td>fire_V_1/shoot_V_3</td>
</tr>
<tr>
<td>Attacker pattern:</td>
<td>&lt;N(subj, PER): Attacker&gt; &lt;V(fire/shoot): trigger&gt;</td>
</tr>
</tbody>
</table>

Figure 4.3 A new pattern derived from the one in Fig. 4.1 after its trigger is replaced

4.1.2 Deriving new patterns through sub-pattern mutation

Since our patterns are derived automatically from data and it is possible that individual event mentions from which they were derived did not contain all possible roles or role combinations that may occur with these types of events, outside of the training set. This means that patterns derived from one data set are likely to have limited coverage when applied to a new data set, especially in an open domain, or with new topics. For example, existing patterns may only recognize subsets of roles within an event e.g., one pattern recognizing Attacker-Target-Time, while another pattern extracting Attacker-Instrument-Location, etc. In such situation we may be able to merge the partially matching patterns into a new composite pattern that would fit
the unmatched example. An algorithm for deriving new patterns by merging or exchanging (mutating) elements of other patterns is shown in Figure 4.4.

1. For a given type of event, apply all its currently know patterns to the new unprocessed test corpus, matching the sub-patterns independently.

2. Sub-patterns matching within the same clause may indicate presence of an event mention even if no currently known pattern matches fully at the same location.

3. If any two sub-patterns within the same clause assign two different roles to the same named entity, we shall remove one of the sub-patterns from further consideration: preferably the one with the lower projected accuracy.

4. New patterns shall be formed from the remaining set of sub-patterns within each clause as follows:
   a. If all the sub-patterns share the same trigger, a single new pattern is trivially formed from them.
   b. If the sub-patterns contain different triggers, then we will create one new pattern for each trigger, as follows. For any trigger T and role R if a subpattern linking T and R already exists then it will be added to the new pattern; otherwise a new subpattern between T and R will be derived from the dependency path between T and R.

**Figure 4.4** Method for sub-pattern mutation

In order to combine sub-patterns from different patterns into a new event pattern, we must first check whether the roles extracted by these sub-patterns in fact belong to the same event. If the role sub-patterns share the same trigger and match within the same clause, we will assume that their roles belong to the same event. Currently we do not combine subpatterns that extract roles from different clauses, because of an increased likelihood that the clauses describe different events. Figure 4.5a shows a sentence that has two different movement events, “Taylor’s *leaving*” and “US forces *going in*”. The dependency tree of this sentence (in Figure 4.5b), shows that each event mention belongs to a different clause “E2/C” and
“E1/C” (C means that the node is head of a clause). Under such circumstances, we assume that the roles extracted from different clauses belong to different events (as is the case here) and thus can’t be combined into a single pattern.

White house officials say it could be days or weeks or more before there is any decision as to whether president Taylor leaving the country means more U.S. forces going in.

Figure 4.5a Examples for multiple same type event mention(s) in one passage

![Dependency tree of part of passage in Figure 4.5a](image)

Figure 4.5b Dependency tree of part of passage in Figure 4.5a

In rare cases, it is possible that one entity is assigned multiple roles within the same event. For example, in the Die event, “Three people plus the bomber were killed…”, the “bomber” is both the Agent and the Victim. However, this type of occurrence is quite rare, only 0.8% of roles were so assigned. In BEAR, we make an assumption that an entity can only be assigned a single role in an event. Consequently, any sub-patterns that would assign an entity multiple roles are removed from the group before the new pattern is formed.
Let’s now walk through an example of how our pattern merging method works in practice.

Figure 4.6a describes a Transfer-Ownership event where “GE” is the Seller; “the French company, Thomson” is the Buyer, while “it” refers to Artifact and “almost 19 years ago” is Time.

“If what you are all after is RCA being a US company that has not been true since GE sold it to the French company, Thomson, almost 19 years ago.”

This event mention is partially matched by the patterns in Figures 4.6b and 4.6c; however, the first pattern can only extract the Buyer and the Artifact roles, while the second pattern only extracts the Seller and the Artifact roles. By combining these two partial patterns we obtain a more complete pattern as shown in Figure 4.6d.

Pattern id: 12
from: original
Projected accuracy: 1.0
Type: Transaction
Subtitle: Transfer-Ownership
Trigger: sell_V_1
Artifact pattern: <N(obj, ORG): Artifact> <V(sell): trigger>
Place Pattern: <V(sell): trigger> <SentHead> <N(GPE): Place>
Buyer pattern: <V(sell): trigger> <Prep = to> <N(ORG): Buyer>

Figure 4.6b A Pattern for Transfer-Ownership event that partially matches 4.6a

Pattern id: 75
from: sample
Projected accuracy: 1.0
Type: Transaction
Subtitle: Transfer-Ownership
Trigger: sell_V_1
Artifact pattern: <V(sell): trigger> <N(obj, ORG): Artifact>
Seller pattern: <N(subj, ORG): Seller> <V(sell): trigger>

The role Artifact in the Transfer-Ownership event is an item or an ORGANIZATION that was bought or sold. Please note that while the pronoun “it” most likely resolves to RCA, this requires an additional step of anaphor resolution.
Chapter 4

**Price pattern:** <V(sell):trigger> <N(obj, FAC)> <Prep> <N> <V> <V> <N(Money): Price>

*Figure 4.6c* Another pattern that partially matches the event mention in 4.6a

<table>
<thead>
<tr>
<th>Pattern id:</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>from:</td>
<td>sub-patterns_mutation</td>
</tr>
<tr>
<td>Projected accuracy:</td>
<td>null</td>
</tr>
<tr>
<td>Type:</td>
<td>Transaction</td>
</tr>
<tr>
<td>Subtype:</td>
<td>Transfer-Ownership</td>
</tr>
<tr>
<td>Trigger:</td>
<td>sell_V_1</td>
</tr>
<tr>
<td>Seller pattern:</td>
<td>&lt;N = Seller(subj, ORG)&gt; &lt;V(sell): trigger&gt;</td>
</tr>
<tr>
<td>Artifact pattern:</td>
<td>&lt;V(sell): trigger&gt; &lt;N(obj, ORG): Artifact&gt;</td>
</tr>
<tr>
<td>Buyer pattern:</td>
<td>&lt;V(sell): trigger&gt; &lt;Prep(to)&gt; &lt;N(ORG): Buyer&gt;</td>
</tr>
</tbody>
</table>

*Figure 4.6d* A new pattern obtained by fusing sub-patterns from 4.6b and 4.6c

4.2 **Creating new patterns from the linguistic context of the extracted events**

The techniques in Section 4.1 are based on fusion of event patterns that were learned from the training examples. These transformations, while leading to an increased event recall, still do not guarantee that the system can learn significantly different forms of event reference that are semantically equivalent but not comparable to any already learned structures. In this section, we discuss additional pattern learning methods that can be used to discover new event structures by examining the linguistic context surrounding the event mentions, which we already know how to find. In other words, we will now attempt to model the event context rather than its internal structure.

4.2.1 **Forming new patterns around alternative triggers**

By definition, an event trigger invokes a specific event type, which in turn activates appropriate extraction patterns in order to identify roles and other elements of an event. In some cases, multiple triggers may exist for the same event. This means that entirely different pattern
structures, anchored on these alternative triggers may also exist. Such structural duality may be
exploited to learn additional patterns for event extraction that cannot be obtained using the
pattern transformations described in the previous section. As an example, let’s consider the
pattern in Figure 4.7b that uses “bombing” as a trigger and the associated role patterns in
order to extract an Attack event mention from the sentence in Figure 4.7a.

Samudra is accused of plotting the terror bombing of two Bali nightspots in an attack that killed 202
people on this Indonesian resort island last October.

**Figure 4.7a** A sentence with an Attack event

<table>
<thead>
<tr>
<th>Pattern id: 1207</th>
</tr>
</thead>
<tbody>
<tr>
<td>from: sub-patterns_mutation</td>
</tr>
<tr>
<td>Projected accuracy: 0.8</td>
</tr>
<tr>
<td>Type: Conflict</td>
</tr>
<tr>
<td>Subtype: Attack</td>
</tr>
<tr>
<td>Trigger: bombing_N_1</td>
</tr>
</tbody>
</table>

**Target pattern:** <N(bombing): trigger> <Prep(of)> <N(FAC): Target>

**Attacker pattern:** <N(PER): Attacker> <V> <N(bombing): trigger>

**Time-Within pattern:** <N(bombing): trigger> <Prep> <N> <Prep> <N> <E0> <V> <N(time2): Time-within>

**Figure 4.7b** A pattern that extracts the event mention from the sentence in 4.7a

Figure 4.7c shows the dependency tree for the sentence in 4.7a. The nodes highlighted in italic
font mark the dependency paths that connect the event roles, Attacker, Target, and Time-
Within, and event trigger, “bombing”, corresponding to the role sub-patterns in Fig. 4.7b. It
turns out that the event mention in 4.7a has also another trigger, “attack”, which is already a
known trigger for Attack events. Clearly, the sub-patterns linking the “attack” trigger to the
event roles will be different than those linking the “bombing” trigger to the same roles. In
other words, if we use “attack” as the trigger (Figure 4.7d), we will need a whole new set of
role sub-patterns, as shown in Figure 4.7e.
Chapter 4

Figure 4.7c The dependency tree of Figure 4.7a and dependency structure (italic part) of the pattern in Figure 4.7b

Figure 4.7d The dependency structure of the new pattern in Figure 4.7c
Figure 4.7e A new pattern obtained by defining an alternative trigger

How do we know if an event mention may contain an alternative trigger? In order to find an alternative trigger for any event mention occurring in text, we consider every word and phrase in the sentence (this may be limited to nouns, verbs, and adjectives), which is not part of the existing trigger phrase, and:

- Is already a known trigger for this event type (i.e., used in other patterns) or a synonym or hyponym of one (in the correct sense), and

- Is found in the same clause as the original trigger, (e.g., such as “bombing” (Figure 4.7c) and “attack” (Figure 4.7d).)

If such alternative trigger can be found in the event mention, BEAR will build a new pattern based on the dependency paths between the roles in the mention and the alternative trigger and subject it for validation process.
4.2.2 Finding new events through trigger and role coreference

In (Gale, Church, and Yarowsky 1992), the authors argued that if a polysemous word is used multiple times within a single document, there is a very high probability that it is consistently used in the same sense (“one sense per discourse”). What it means for us is that if we extract an event mention (of type T) with trigger \( t \) (i.e., phrase \( t \) used in sense \( s \)) in one part of a document, and then find that \( t \) occurs in another part of the same document, then we may assume that this second occurrence of \( t \) has the same sense as the first; in other words, \( t \) occurs again. Since \( t \) is a trigger for an event of type T, we can hypothesize its subsequent occurrences indicate additional mentions of type T events that were not extracted by any of the existing patterns. Heng (2008) exploited this phenomenon for improved detection of event triggers in NYU event extraction system. Our objective is to exploit these unextracted mentions to automatically generate additional event patterns.

To verify our hypothesis we ran several experiments on ACE training corpus, in which we compared the triggers and the roles across all event mentions within each document. The purpose of this experiment was to determine whether different event mentions sharing the same trigger and perhaps some common roles are indeed of the same type, and if so, under what circumstances we can assume that this assumption is reliable enough for effective pattern learning.

\(^{40}\) In the experiment described by Gale et al., the context is a 100-words window around a polysemous word. This is just about the size of average document in ACE data set (it contains 4160 documents and around 350,000 words total).
The following is the procedure how to find correlations between a sentence that contains an extracted event mention and every other sentence in the same document. The event mention, EVM1 \((t; E_{n_1}, \ldots, E_{n_m})\) where \(t\) is the trigger and \(E_{n_i}\) are the role entities is compared against every other sentence, SE \((w_1, \ldots, w_k; E_{n_1}, \ldots, E_{n_p})\) where \(w_j\) are words and \(E_{n_j}\) are entities found in SE. If there is no \(w_j\) in SE such that \(w_j = t\) then we return FALSE and stop. Otherwise, we count the number, \(c\), of co-referential entities between EVM1 and SE and return the ratio \(c/m\) (Figure 4.8). The output values are then correlated with the training data of annotated event mentions to chart the probability that SE contains a mention of EVM2 of the same type as EVM1 (Figure 4.9a and Figure 4.9b).

In Figure 4.9a, the X-axis is the percentage of entities coreferred between the EVM and the SE; while the Y-axis shows the probability that the SE contains a mention that is the same type
as the EVM. This Figure shows that when the trigger projected accuracy is 0.5 or higher, each
of its occurrences within the document indicates an event mention of the same type with a
very high probability (mostly > 0.9). For triggers with lower projected accuracy, this high
probability is only achieved when the two mentions share at least 60% of their roles, in
addition to having a common trigger.

![Graph](image.png)

**Figure 4.9b** The same analysis as in Fig 4.9a but allowing for SHD forms of the trigger to
locate additional event mentions.

Documents may also contain synonyms, hyponyms, and derived (SHD) forms of the first
mention trigger, in subsequent mentions of the same type events. Accordingly, we expanded
our experiment to cover such situations where a SHD form of a trigger appears in another
sentence along with the same entities (Figure 4.9b). We found that if we accept trigger variants
as indicators of additional event mentions the probability of finding an event of the same type
is still over 80% as long as the projected accuracy of the original event trigger is above 0.5.
However, if the projected accuracy of the trigger is below 0.5, its variants will be less likely to
indicate additional event mentions and the probability of finding an event of the same type will
fall under 0.8. Thus in BEAR, we would use the SHD forms of a trigger only if the projected accuracy of the trigger is over 0.5.

One remaining issue is to determine the correct assignment of roles within each of the newly discovered event mentions. Our statistical analysis shows that the entities shared between the event mentions of the same type are assigned the same roles in each mention with 84% probability. Thus, if the same entities appear in multiple event mentions of the same type, we will assign them the same roles as already assigned in the extracted mentions. For any new arguments, we will use other techniques to assign roles, including prepositional analysis (section 4.2.3) and sub-pattern mutation, as appropriate.

It may be interesting to note that among the collected pairs of same-type events (EVM1, EVM2) only 28% were co-referential, i.e., both mentions were of the same event. However, if the two mentions shared at least 50% of their roles, then the coreference rate increased to 71%.

| A. **Capek** was forced to **resign** from **UBS** because he had discussed the stock with a client… |
| B. **Howard G. Capek** resigned under pressure last week when **UBS** executives discovered the first e-mail message. |
| C. **Li Feng** assaulted **19 students** in his class… |
| D. **Li** was charged with rape and sexual assault on **all 19 students** |

**Figure 4.10** Two examples that the event mentions identified by the common trigger and co-referential arguments (in both cases here, the event mentions are also co-referential)

| Pattern id: -1 |
| from: propagate |
| **Projected accuracy:** null |
| **Type:** Personnel |
| **Subtype:** End-Position |
| **Trigger:** resign_V_1 |
Figure 4.10 shows two examples where an additional event mention is found by BEAR because it has the same trigger and co-referential arguments with an extracted event mention within the same document. In the first example, an End-Position event is extracted from A, with “resign” as the trigger, and “Capek” and “UBS” assigned Person and Entity roles, respectively. The sentence B, taken from the same document, contains the same trigger word, “resigned’ and also the same entities, “Howard G. Capek” and “UBS”. The projected accuracy of resign_V as an End-Position trigger is 0.88. With 100% argument overlap rate, we estimate the probability that B contains an event mention of the same type as A (and in fact co-referential mention) at 97%. Thus a new pattern for End-Position is automatically derived from B, as shown in Figure 4.11 (the first pattern). If this pattern passes the subsequent validation step, it will be added to the set of patterns to be used in the next system iteration.

In the second example in Fig. 4.10, an Attack event mention is extracted from C, where “assaulted” (verb) is the trigger. The same trigger is found in another sentence (D) later in the same document, although in a derived form, “assault” (noun), which is a known Attack

---

41 Entity is the employer in the event
42 fire_V means the POS of fire is verb.
trigger with the projected accuracy of 0.67. When combined with the 100% entity overlap, the estimated possibility that sentence D contains an Attack event is 90%. Therefore, a new Attack pattern will be built out of sentence D (second raw in Figure 4.11).

4.2.3 Prepositions as role indicators

Prepositions are well-known indicators of semantic roles for the noun phrases that follow them. For example, “near” followed by a GPE noun phrase often indicates Location, while “to” may indicate Destination in some events. Fillmore’s (1968) Case Grammar and Schank’s (1973) Conceptual Dependency Theory exploit these properties of prepositions to derive semantic representation of natural language sentences.

In BEAR, when a new event mention is encountered such that its roles cannot be determined by any existing sub-patterns, we can fall back on prepositional indicators to find new dependency paths and to define new role sub-patterns. But just how reliable such indicators are? Many prepositions are ambiguous in their usage, and defining a symbolic rule system, such as those based on Case Grammar, may be impractical. Also, the manually defined rules usually lack sufficient coverage for many possible variants that may be encountered when processing a large text corpus. So, instead of defining rules by hand, we may attempt to automatically learn which prepositions are reliable role predictors within specific event types. To do so, we calculate the rate of co-occurrence between prepositions, event types, and roles. For each sentence that contains a mention of a specific event type, we collect all prepositions that are heads of named entities within the same clause as the event trigger.
Troops from the U.S. Army’s 101st Airborne Division went to the site on Friday, finding a number of large drums buried in bunkers.

Figure 4.12 co-relations between preposition and event roles

Figure 4.12 shows a dependency structure for a sentence containing a mention of Movement-Transport event (“went” is the trigger; “Troop” is the Artifact; “site” is the Destination, and “Friday” is the Time) and the dependency tree of the passage. We collect 5 prepositions: “from”, “to”, “on”, “of”, and “in”. From the parse tree, we note that “of” and “in” are within a subordinate clause (anchored at “finding”), and thus do not belong to the clause with the trigger. This leaves 3 prepositions (from, to, on) for which we build preposition-entity pairs as follows: “from-ORG (Division)”, “to-FAC (site)”, and “on-TIME (Friday)”. Since “site” and “Friday” are annotated as the roles, Destination and Time-Within, respectively, we increase the
“to-FAC” count as indicator of Destination role and the “on-Time” count as indicator of Time-Within role in Movement. On the other hand, since “Division” doesn’t have any role in this Movement event, only the number of occurrence of “from-ORG” is increased.

After applying this process to all event mentions in the training corpus, we obtain 527 preposition-entity pairs that are associated with specific event types and roles within these events. Table 4.1 lists a sample list of correlations. The first 4 columns in this table are self-explanatory. The fifth column shows the number of times that the preposition-entity type pair occurs as part of a mention of this type of event. The sixth column is the number of times that the entity is assigned this role. The seventh column is the ratio between the numbers in columns 5 and 6. It is easy to see that, when paired with certain types of entities, some prepositions are remarkably good role indicators, although this may vary between event types, e.g., from-GPE is a good indicator (0.9) of Origin in Movement events, while in-VEH is a weak indicator of Destination (0.2) in Attack events.

<table>
<thead>
<tr>
<th>Preposition</th>
<th>Entity type</th>
<th>Event type</th>
<th>Event role</th>
<th>(prep,EntType) occurrences</th>
<th># of times as role indicators</th>
<th>Correlation rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>against</td>
<td>ORG/PER</td>
<td>Sue</td>
<td>Defendant</td>
<td>7</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>at</td>
<td>ORG</td>
<td>End-Position</td>
<td>Entity</td>
<td>7</td>
<td>6</td>
<td>0.86</td>
</tr>
<tr>
<td>for</td>
<td>Crime</td>
<td>Sentence</td>
<td>Crime</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>from</td>
<td>GPE</td>
<td>Transport</td>
<td>Origin</td>
<td>20</td>
<td>18</td>
<td>0.9</td>
</tr>
<tr>
<td>from</td>
<td>GPE</td>
<td>Transport</td>
<td>Destination</td>
<td>20</td>
<td>1</td>
<td>0.05</td>
</tr>
<tr>
<td>in</td>
<td>VEH</td>
<td>Attack</td>
<td>Target</td>
<td>5</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>into</td>
<td>GPE</td>
<td>Transport</td>
<td>Destination</td>
<td>25</td>
<td>24</td>
<td>0.96</td>
</tr>
<tr>
<td>with</td>
<td>Crime</td>
<td>Charge-Indict</td>
<td>Crime</td>
<td>9</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>with</td>
<td>ORG</td>
<td>Attack</td>
<td>Attacker</td>
<td>5</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>with</td>
<td>WEA</td>
<td>Attack</td>
<td>Instrument</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

... Table 4.1 Correlation rates between events types, and prepositions as role indicators
The correlations we are interested in can be expressed as a relation \( \text{PER: prep} \times \text{entity-type} \times \text{event-type} \rightarrow \text{role} \), where the values of the first 3 arguments can predict the fourth, or \( \text{PER(p, E, EV)} = r \). In BEAR, we select only the arguments for which this prediction rate is over 0.8 and the prediction has at least 3 correct predicts in training data (3 is the average number of correct predicts at training corpus of all predictions). In order to use PER relations to locate event roles that were unmatched by existing patterns, we perform the following steps:

1. Find all prepositions that are in the same clause as the event trigger \( t \) for event type \( T \), but retain only those whose immediate children\(^{43}\) are named entities.

2. Remove the prepositions whose children have already event roles assigned.

3. For any remaining preposition \( p \) which is the head of a prepositional phrase containing an entity \( e \) of type \( E \), find if there exists a PER relation such that \( \text{PER}(p, E, T) = r \) and assign role \( r \) to \( e \). Create a new role sub-pattern between \( e \) and \( t \).

We illustrate this method with the following example. The sentence in Figure 4.13 contains a mention of an Attack event, which we discussed before in section 4.3.2, where we showed how to detect this event and match two of its roles: Attacker (“MP”) and Target (“taxi”). The third role, Instrument (“M-16 rifle”) was not extracted because we lacked an appropriate subpattern. Now, however, using the list of prepositional indicators we can locate the Instrument role, based on the high prediction rate for \( \text{PER(with, WEA, Attack)} = \text{Instrument} \).

\(^{43}\) There is no other node in the dependency path between a node and its direct child.
Chapter 4

An MP in the rear vehicle fired into the grill of the taxi with an M-16 rifle.

Figure 4.13 A dependency tree for an Attack event mention with an Instrument role

Of the 3 prepositional phrases in this sentence, “of + (the) taxi” is already assigned a role (Target), which leaves us with “in-VEH” and “with-WEA”. There is no known value for PER(in, VEH, Attack); however, PER(with, WEA, Attack) = Instrument has 100% prediction rate. Thus, we say that “the M-16 rifle” is the instrument of this event and we add an appropriate role sub-pattern into the previously derived partial pattern (Figure 4.14).

Pattern id: -1
from: propagate+rl-ind
Projected accuracy: null
Type: Conflict
Subtype: Attack
Trigger: fire

44 “into” in the sentence is not considered because grill is not tagged as an entity
4.3 Relaxing pattern constraints

There is one important consideration that we have not discussed thus far, which is pattern over-fitting. Since we require that all roles are matched before an event is extracted, many abridged event mentions may go un-extracted because some of their roles are not present, unless, of course a right pattern exists. To address this issue, we allow pattern relaxation, so that only a subset of roles can be matched. This amounts to essentially creating a new pattern, which is a subset of elements of the original pattern. However, just like with any new pattern, we need to perform a validation process before the pattern is adopted into the system (see Chapter 3 for details of the validation process.) If the new pattern doesn’t receive a high enough projected accuracy score, BEAR will not learn it.

Figure 4.14 Adding a role sub-pattern based on a strong prepositional indicator

![Pattern Table]

<table>
<thead>
<tr>
<th>Pattern id: 160</th>
</tr>
</thead>
<tbody>
<tr>
<td>from: sample</td>
</tr>
<tr>
<td>Projected accuracy: 1.0</td>
</tr>
<tr>
<td>Type: Movement</td>
</tr>
<tr>
<td>Subtype: Transport</td>
</tr>
<tr>
<td>Trigger: leave</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Artifact pattern:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;N(subj, PER): Artifact&gt; &lt;V(leave): trigger&gt; 0.26</td>
</tr>
</tbody>
</table>

| Origin pattern: |
|-----------------
| <V(leave): trigger> <N(obj, GPE): Origin> 0.83 |

<table>
<thead>
<tr>
<th>Time-Within pattern:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;V(leave): trigger&gt; &lt;N&gt; &lt;Prep&gt; &lt;N(time2): Time-Within&gt; 1.0</td>
</tr>
</tbody>
</table>

Figure 4.15 A pattern with high projected accuracy sub-patterns
White House officials say it could be days or weeks or more before there is any decision as to whether president Taylor leaving the country means more U.S. forces going in.

<table>
<thead>
<tr>
<th>Pattern id: -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>from: relaxation</td>
</tr>
<tr>
<td>Projected accuracy: null</td>
</tr>
<tr>
<td>Type: Movement</td>
</tr>
<tr>
<td>Subtype: Transport</td>
</tr>
<tr>
<td>Trigger: leave</td>
</tr>
</tbody>
</table>

**Artifact pattern:** \(<\text{N(subj, PER): Artifact}> <\text{V(leave): trigger}> 0.26\)

**Origin pattern:** \(<\text{V(leave): trigger}> <\text{N(obj, GPE): Origin}> 0.83\)

Figure 4.16 A new “relaxed” pattern obtained by removing the Time-Within sub-pattern

We use an example to illustrate this method. Figure 4.15 is a pattern generated from samples in training corpus. In this pattern, three sub-patterns, Artifact, Origin, and Time-Within, are defined. When this pattern is used to match the sentence in Figure 4.16, the Artifact, “Taylor”, and the Origin, “the country” are matched; however, the Time-Within role defined in this pattern is not matched, because there is no time mentioned for this sentence. In order to allow this partial match to count, we need perform pattern relaxation by removing the Time-Within sub-pattern from the original pattern and create a new pattern (Figure 4.16). Basically, we can generate new patterns from any partially matched event mention, because all these candidate patterns have to pass our pattern validation process before they are used for event extraction.

### 4.4 Pattern validation using both training and test corpora

The pattern validation process described in Chapter 3 depends entirely on the hand-annotated training set. In order to validate new patterns found by the bootstrapping methods introduced in this chapter, we need to extend the process so that it can work off of both the training data and the automatically extracted events in test set. We can’t quite use the whole test corpus
because it does not provide a complete annotation set, especially in early iterations. Instead, we will only utilize the set of event mentions that were annotated in prior bootstrapping iterations, and use these to test for precision of any new patterns.

The process of pattern validation against the extended corpus is the same as the process introduced in Chapter 3 by counting the ratio of positive matches and false alarms for each pattern. When a pattern detects a new event mention in the test corpus where there is already an event mention extracted in the same sentence by a previous iteration, we will count as follows:

1. If the previously extracted event has the same type as the newly detected event mention and they both have the same roles, then we count a positive match for the pattern.

2. If the previously extracted event has a different type than the newly detected event mention, we will count a false alarm for the pattern.

After the validation, the projected accuracy of new patterns, sub-patterns, and triggers will be calculated by dividing the number of positive matches by the total number of matches in already annotated data. If the projected accuracy of a pattern passes a specific threshold, it will be added to the system; otherwise it will be discarded.

4.5 Experiments and Evaluation
In this section, we discuss the experiments we conducted to evaluate the performance of our approach: (1) how effectively the system can learn event patterns and (2) how accurate these patterns are for extraction of events from a new, unannotated text data. We test the system learning effectiveness by comparing its extraction performance immediately after the first learning iteration (i.e., using rules derived from the training data) with its performance after 10 cycles of unsupervised learning. The evaluation is conducted at two levels: the accuracy of detection of event mentions and the accuracy of resolving event co-reference, i.e., combining mentions of the same event into a comprehensive event representation. In this chapter we discuss the accuracy of extracting event mentions. In Chapter 5, we discuss the evaluation of event coreference resolution and compare BEAR performance to the IE systems that participated in ACE 2005.

4.5.1 BEAR performance and analysis

In this section we discuss BEAR’s performance of the event mention extraction task against the ACE-2005 test corpus.

4.5.1.1 BEAR performance at various pattern thresholds

During each learning cycle, BEAR calculates the projected accuracy scores for all newly discovered patterns, sub-patterns, and triggers, but it learns only these patterns whose projected accuracy is over a threshold, which we shall call the learning threshold. This learning threshold can be adjusted to allow BEAR learn more patterns at the expense of their decreased precision. If an application requires more precision, the threshold can be set higher; if more recall is needed the threshold can be set lower. Figure 4.17 shows the learning behavior of BEAR when different learning thresholds are used.
In Figure 4.17, X-axis shows values of the learning threshold (in descending order), while Y-axis is the average F-score achieved by the automatically learned patterns for all types of events against the ACE test corpus. The red (lower) line represents BEAR’s base run immediately after the first iteration (supervised learning step); the blue (upper) line represents BEAR’s performance after an additional 10 unsupervised learning cycles are completed. We note that the final performance of the bootstrapped system steadily increases as the learning threshold is lowered, peaking at about 0.5 threshold value, and then declines as the threshold value is further decreased, although it remains solidly above the base run. Analyzing more closely a few selected points on this chart we note, for example, that the base run at threshold of 0 has F-score of 38%, which represents 31% recall, 47% precision. On the other end of the curve, at the threshold of 0.9, the base run precision is 93% but recall at only 21%, which produces F-score of 34%. It is interesting to observe that at neither of these two extremes the system

---

45 The learning process for one type of event will stop when no new patterns can be generated, so the number of learning cycles for each type event is different. The longest learning cycles is 10 and shortest ones is 2.
learning effectiveness is particularly good, and is significantly less than at the median threshold of 0.5 (based on the experiments conducted thus far). At the low end, the initial pattern precision is insufficient to sustain an effective bootstrapping; at the high end too few patterns are learned. On the other hand, learning effectiveness at the median threshold is quite impressive, improving the system performance in extracting event mentions from 44% to 69% F-score, which represents 82% precision 59% recall. Figure 4.18 shows the learning curve of BEAR while threshold is at 0.5. Most learning is done before the first 4 steps and then fewer and fewer resources can be used for system to learn. After 10 iterations, no more new events can be found. This is better than any currently available event extraction technology; nonetheless, we believe there is more room for improvement in future research.

![Figure 4.18 BEAR Learning Curve while Threshold is at 0.5](image)

4.5.1.2 Failure Analysis of BEAR’s performance

In this section, we perform failure analysis in order to understand why BEAR still misses some event mentions or why it extracts some false positives.
One source of error is inconsistency among annotators as well as annotator mistakes. Figure 4.19 (Weischedel, 2006) shows the inter-annotator agreement on each type of event on a subset of ACE 2005 dataset (called Pilot 1). The agreement is calculated along the following two dimensions: 1) whether an event mention is present in a specific sentence; and if so 2) what is the type of the event mentioned. Weischedel finds that the level of agreement in 1) is quite low and averages only 66%. Our analysis shows that at least 19% of the false alarms recorded by BEAR fall onto cases where the annotators disagree.

Also, we found that many errors are simply the result of using less than perfect patterns: more than 80% BEAR patterns have the projected accuracy of less than 100%. When we analyzed the event mentions that BEAR failed to extract, we found that the high degree of variability among these event mentions was one of the key factors involved. For example, among the Personnel event mentions that BEAR could not extract 31% used syntactic constructs radically different from the examples in the training corpus.
Figure 4.20 shows BEAR’s average precision after the 10th learning iterations. The average precision of BEAR across all ACE-defined event types is 83.7% and there are no significant differences in precision for different types of events, which shows that our pattern validation process is quite robust.

A detailed analysis of the 16.3% of false alarms reveals that a significant portion of these (about a fifth) were introduced by a handful of underperforming patterns whose actual precision on test corpus was significantly below their projected accuracy obtained through the validation process. As an example, let’s consider the pattern for the Conflict-Attack event shown in Figure 4.21.
This pattern was derived from the training sentences that refer to events in which bombs were dropped from fighter aircraft during the Iraq war. This pattern has a very high projected accuracy in training set; however, when used on the test data, it only extracted two false alarms. These two false alarms were structurally similar but semantically very different event mentions, in which the police ask someone “to drop his gun”. Since the training corpus did not contain such negative samples, the acquired pattern could not be refined to make a semantic distinction between “drop a bomb” and “drop a gun”.

After a closer look at the underperforming patterns, we found that as many as 80% of them have been derived from fewer than 10 supporting text examples (Figure 4.22). One main reason that the patterns having less 10 supporting samples contribute to the most part of overestimated problem is that more than 69% of total patterns having less 10 supporting samples.
samples. So, we calculated the ratio of the overestimated patterns against all patterns used for event extraction (Figure 4.23). It shows that 19% of the patterns with less 10 supporting samples are overestimated; as for the patterns with 10 to 19 supporting samples, 13% are overestimated. Only 6% of the patterns with more than 30 supporting samples are overestimated (we don’t have patterns having 20 to 29 supporting samples). The results show clearly that with less training data, the system is more likely to make wrong estimation.

![Figure 4.23 The distribution of the overestimated patterns on all used patterns](image)

**Figure 4.23** The distribution of the overestimated patterns on all used patterns

![Figure 4.24 BEAR’s average recall after the bootstrapping process is complete (at learning threshold of 0.5)](image)

**Figure 4.24** BEAR’s average recall after the bootstrapping process is complete (at learning threshold of 0.5)
Chapter 4

Figure 4.24 is the recall on each type of ACE event. The overall recall (leftmost bar) is 58%. Failure analysis shows that one source of error is the automatic parsing. BEAR uses Minipar dependency parser to extract syntactic structures that are used in making of patterns. The average F-score for Minipar on the SUSANNE corpus\(^\text{46}\) is 83.8%. Based on our analysis, at least 6% of the seed patterns obtained from the training corpus come from sentences that have incomplete or incorrect parse trees; the patterns generated from these seeds are therefore of questionable value. Similarly, we found that in the test corpus more than 4% of event patterns are derived from the sentences with incorrect or incomplete parse trees; naturally, these patterns will contribute to performance errors in the final evaluation.

Figure 4.24 shows that BEAR final recall varies greatly for individual types of events. The highest recall (66%) is for Conflict events and the lowest recall (35%) is for Personnel events. After further analysis, we found two major reasons for this variance in recall besides the structural variability between mentions: (1) differences in size of the initial learning samples for different event types and (2) pattern under-specification.

The first reason for the uneven performance is the number of instances of each event in the training set – events that have fewer positive examples in the training corpus give rise to fewer extraction patterns which translates into less recall, because more samples provide more structural variety, and thus increase the probability of generating patterns that find more events in the test corpus. Figure 4.25 shows the number of seed examples of each type of event in the training corpus. The number of Conflict seeds is about 3 times the number of Personnel seeds. With more seeds, BEAR learns more initial patterns for this type of event and thus is expected to extract more (and more precisely) new events from the test corpus.

\(^{46}\) Please refer [http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/susanne/susanne.doc](http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/susanne/susanne.doc) for more detail information about this corpus.
Nonetheless, the size of the seed sample is not the only reason for recall shortfall as may be noted by comparing the charts in figures 4.24 and 4.25. For example, the Personnel event type has at least 25% more seed samples in the training corpus than the Contact event type, at yet the recall of Personnel is 10% lower. The second reason for recall variability that we identified has to do with the patterns projected accuracy. If the projected accuracy of patterns (as well as sub-patterns and triggers) derived from the training examples is low, they will not pass the validation threshold, thus effectively cutting the size of the learning data sample. To illustrate this effect, we can compare the number of patterns generated for Personnel and Life event types. Figure 4.26 shows the percentage of patterns, sub-patterns, and triggers for these two event types that passed the learning threshold. We note that there are 28% more patterns, 21% more sub-patterns, and 61% more triggers generated for Life events than for Personnel events. From this example, we can see that BEAR has less qualified samples in Personnel events to learn and thus couldn’t find as many events as it does in other types of events.
4.5.2 BEAR performance with various combination of learning methods

We described six learning methods that BEAR uses to discover new event extraction rules. Three of the six methods work by manipulating already learned patterns in order to adapt them to new data – and thus creating new slightly altered patterns (section 4.2 and 4.4). The remaining three methods (section 4.3) work by exploiting a broader linguistic context in which the extracted events occur. These methods look for structural duality in the sentences containing event mentions in order to discover alternative extraction patterns. In this section, we report the results of running BEAR with each of these two groups of learning methods separately and then in combination to see how they contribute to the end performance.

For ease of reference, we shall call the group of learning methods working off of the existing patterns the *Pattern-Based methods* (PBM), and the group of learning methods working off of the context of extracted events the *Context-Based methods* (CBM); the combined set of all methods...
Chapter 4

will be referred to as Combined. Table 4.2 shows BEAR performance under these three learning options, after 10 learning iterations. In each iteration, new extraction rules are learned based on the rules learned in previous iterations and the events extracted by these rules. All new rules are adopted only after they are tested for their projected accuracy (and specifically their potential for creating false alarms), so that the overall precision of the resulting rule set is maintained at a high level relative to the Base Run.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
<th>Annotated</th>
<th>New Extracted</th>
<th>Correct</th>
<th>False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Run (after 1st iteration)</td>
<td>0.89</td>
<td>0.29</td>
<td>0.44</td>
<td>1390</td>
<td>447</td>
<td>400</td>
<td>47</td>
</tr>
<tr>
<td>Combined run after 10th iteration</td>
<td>0.82</td>
<td>0.59</td>
<td>0.69</td>
<td>1390</td>
<td>1008</td>
<td>823</td>
<td>185</td>
</tr>
<tr>
<td>Pattern-Based run after 10th iteration</td>
<td>0.83</td>
<td>0.52</td>
<td>0.64</td>
<td>1390</td>
<td>863</td>
<td>719</td>
<td>144</td>
</tr>
<tr>
<td>Context-Based run after 10th iteration</td>
<td>0.84</td>
<td>0.44</td>
<td>0.58</td>
<td>1390</td>
<td>729</td>
<td>614</td>
<td>115</td>
</tr>
</tbody>
</table>

Table 4.2 Performance of BEAR by applying learning methods either from PBM, or CBM, or the combination of PBM and CBM

The combined set of methods is significantly more effective than either of the two groups considered separately. With the combined methods the system learns enough to deliver significantly improved extraction performance with the F-score increasing by more than 56% over the Base Run.

4.6 Conclusions

In this chapter, we presented several methods by which BEAR can automatically learn new event extraction patterns from unannotated text. These methods fill the significant gaps in the event extraction capability that may be obtained through purely supervised methods (or from manually designed rules). The eight methods described here are based on structural
manipulation of already learned extraction patterns and on exploiting the linguistic and structural context of extracted event mentions in order to learn alternative extraction rules. We control the quality of the learning process by setting up a strict pattern validation process, which requires that all generated patterns have the projected accuracy score above a preset learning threshold.

We also reported on performance evaluation at the event mention level. We used the base run of BEAR on the ACE test corpus as the baseline. After 10 iterations, when no more new events could be generated, the performance of BEAR was evaluated and we noted a significant improvement over the baseline. We tested system learning using two groups of methods: pattern based and context based, and the results showed that each group contributed to the final performance of BEAR. We also performed failure analysis in order to determine the sources of errors, both false alarms and misses. We found several major reasons for errors: the insufficient training data, imperfections of the learned patterns, perplexity of event triggers and sub-patterns, parser failures, and finally inconsistencies in the training data introduced by disagreement among the annotators.

In the next chapter, we discuss event coreference resolution at the document level and describe further evaluation experiments. We also compare the results obtained with BEAR against those obtained by other event extraction systems that participated in the ACE 2005 evaluations.
EVENT COREFERENCE

In this chapter, we discuss the issues of finding co-referential event mentions within the same document. We start by providing a definition of event coreference\(^\text{47}\) and how we utilize it in this thesis. We then discuss some previous approaches to coreference resolution, which are relevant to our task, and explain the novelty of our approach. Finally, we discuss and analyze the results of a formal evaluation conducted against ACE-2005 event extraction task. We also outline some directions for future work.

5.1. Event mention coreference

Two event mentions are co-referential if they both refer to the same event. For example, the events mentioned in the following passages are co-referential, because they both refer to the same historical event:

“Maddux was killed in Philadelphia.

…. Einhorn is accused of killing Maddux.”

Both passages mention a Life-Die type event. In the first message, the Victim is “Maddux”, and the Place is “Philadelphia”; in the second message, the Agent is “Einhorn”, and the Victim is “Maddux”. Each mention provides only partial information about the event; after the two mentions are combined, a more complete picture of the event emerges.

\(^{47}\) The definition refers ACE event annotation guide, http://projects.ldc.upenn.edu/ace/docs/English-Events-Guidelines_v5.4.3.pdf
Event coreference is the final step of the event extraction task in the ACE program. In the next section, we discuss some background research in event coreference, especially as related to the systems participating the ACE program.

5.2. Prior research in event coreference

Event coreference in MUCs was part of the template filling process, and was implemented using hand-crafted rules, alongside of those designed to fill in the template slots. In the ACE program, event extraction was defined at two levels: (1) the level of event mentions, which are considered independently of one another, and (2) the level of complete events that combine all co-referential mentions into a complete event representation. In ACE, the event coreference is thus explicitly separated from the event extraction process. This design has two advantages: one is that some of the research can focus entirely on issues related to coreference; the other is that coreference effectiveness in systems can be evaluated independently from event extraction.

Grishman at el. (2005) and Ahn (2006) proposed using machine learning methods to automatically detect coreference between event mentions. Both systems trained a maximum entropy based classifier for event co-reference. The features for training are collected from co-referential event mention pairs in the training corpus. Figure 5.1 shows the features collected by the NYU system and Figure 5.2 shows the features used in Ahn’s system.

- Event type and subtype
- Whether the two mentions have the same trigger
- The distance between the two mentions
The roles for which the two mentions have the same values assigned
The roles for which the two mentions have different values assigned

Figure 5.1 Features used in training for event coreference in the NYU system

- The triggers that are used by the two mentions
- The event types of the two mentions
- Depth of triggers of the two mentions in their parse trees
- Distance between two mentions, measured in sentences
- The roles for which the two mentions have the same values assigned
- The roles for which the two mentions have different values assigned
- The roles in the earlier mention that are not included in the later one
- The roles in the later mention that are not included in the earlier one
- The shared arguments in both mentions that are assigned different roles

Figure 5.2 Features used for event coreference in Ahn system

From the features collected by the two systems, we can see that their focus is quite similar. The only real difference between them is that Ahn’s system uses more detailed set of features, e.g., the actual triggers vs. a binary relation of “sameness”, while the NYU’s system uses fewer but more general features. In what follows, we will use the combined set of features from the two systems as the baseline for comparing our own system performance.

5.3. Features selected for computing event coreference in BEAR

In this section we describe the set of features used in computing coreference between event mentions in BEAR. We will demonstrate that our set of features performs better than the features used by other systems. We used two different classifiers: Maximum Entropy (ME) and Support Vector Machine (SVM). One of our objectives was to find which classification
method is more effective for predicting event coreference with the extended set of features in BEAR.

In what follows, we use the MENTION1 to denote an event mention that appears earlier in a document; another mention of the same event occurring later in the document shall be called MENTION2. For each pair of event mentions in text we shall decide if they form a coreferential pair, based on a list of features collected by the system. The features collected by BEAR come from four sources: the types of events in the pair (TP); their triggers (TR); their roles (RL); and the linguistic context surrounding each mention (CT).

Table 5.1 shows the list of features used in BEAR to determine coreference between any pair of event mentions. Many of the features are self-explanatory; for those that are not, we provide additional explanation and examples below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Features Description</th>
<th>Feature value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP1</td>
<td>Are the mentions of the same type of event</td>
<td>True/false</td>
</tr>
<tr>
<td>TR1</td>
<td>Do the mentions share the same trigger</td>
<td>True/false</td>
</tr>
<tr>
<td>TR2</td>
<td>Actual values of their triggers</td>
<td>Lemma_POS of the triggers</td>
</tr>
<tr>
<td>TR3</td>
<td>Semantic relation between the two trigger senses</td>
<td>Identical/Hyponym/Hypernym/Synonyms/Derived/None</td>
</tr>
<tr>
<td>TR4</td>
<td>Does the trigger of MENTION2 has a definite determiner</td>
<td>the/this/that</td>
</tr>
<tr>
<td>RL1</td>
<td>Do both mentions include the same entities in the same roles?</td>
<td>True/false</td>
</tr>
<tr>
<td>RL2</td>
<td>Name of the roles in RL1</td>
<td>Role_Name, e.g. Attacker</td>
</tr>
<tr>
<td>RL3</td>
<td>Do both mentions include the same entity in different roles</td>
<td>True/false</td>
</tr>
<tr>
<td>RL3</td>
<td>The list of the role pairs in RL3</td>
<td>MENTION1_rl_type-MENTION2_rl_type, e.g. Attacker_Target</td>
</tr>
<tr>
<td>RL4</td>
<td>Do both mentions contain different entities in the same roles?</td>
<td>True/false</td>
</tr>
<tr>
<td>RL5</td>
<td>Types of the roles in RL4</td>
<td>Role_type</td>
</tr>
<tr>
<td>RL6</td>
<td>The percentage of roles shared between the two mentions</td>
<td>The percentage falls into one of ten folders from 0.1 to 1.0</td>
</tr>
<tr>
<td>CT1</td>
<td>Distance between MENTION1 and MENTION2</td>
<td>In same sentence/same</td>
</tr>
</tbody>
</table>
TR3 is a new feature set which captures the semantic relation between the trigger senses. Most previous approaches to event coreference only checked whether the triggers from the two event mentions were the identical lexemes. In our system, we also check if the triggers are used in the same sense or if their senses are compatible, i.e., if they can be linked through semantic relations in Wordnet, including synonym, hypernym, hyponym, or derived forms.

We use two examples in Figure 5.3 and 5.4 to show what we try to get from TR3. In the two mentions in Figure 5.3, the triggers, {	extit{bombed}.verb} and {	extit{bombings}.noun}, are different and have different POS, but they are morphological variants and specifically Wordnet links the noun to the verb in sense #1 (Figure 5.5) as having the same meaning. Similarly, the triggers in the two
co-referential mentions in Figure 5.4 are related because *hang* is a hyponym of *kill* in Wordnet. More generally, such semantic correlations, e.g., sense-preserving morphological variation, hyponymy, etc. will be used as positive indicators that coreference exists between two event mentions.

<table>
<thead>
<tr>
<th><em>bom<strong>b</strong>ing</em>, bombardment (noun)</th>
<th>RELATED TO- &gt; (verb) *bom<strong>b</strong>#1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>=&gt; bombard, bomb</td>
</tr>
</tbody>
</table>

| **hang**, string up                | => execute, put to death          |
|                                   | => *kill*                          |

**Figure 5.5** Hypernym and morphological variant links between word senses in WordNet

Another new feature is TR4, which is set whenever the trigger of a later of the two event mentions is a definite noun phrase, headed by a definite determiner, such as *the, this, that*, etc. If this feature is present, it may suggest that the same event was already mentioned earlier in the document because the definite NP may be an anaphor. For example, in the second of the two event mentions shown in Figure 5.6, “*the*” in front of *attacks* is a strong indicator of coreference with the first mention.

> “it's been a year and a half since a series of anthrax **attacks** killed five people.”
> “well, the fbi can drain the pacific ocean but they won't find any evidence that he was involved with the **attacks** …”

**Figure 5.6** Two co-referential event mentions where the later one used a definite NP anaphor trigger

CT2 and CT3 are also new features. One of them is set whenever there is another event mention M3 between MENTION1 and MENTION2, of the same type as MENTION2. If M3 exists, we check whether it is co-referential with MENTION1. We collect this true/false feature based on an assumption that if M3 does not corefer with MENTION1, it may indicate
that a topic change occurred in the document, and thus MENTION2 may not be coreferential with MENTION1.

5.4. Experiments and discussion

5.4.1. Corpora

We use the ACE corpus already described in chapters 3 and 4 and work on event pairs from each document in the corpus, which forms another layer of annotation over the event mentions. The followings are the statistics showing the number of event pairs generated and the number of features collected from the training and test corpora.

- Training corpus

  From each document of training set, we create event pairs by connecting any two annotated event mentions. We collected 32,972 pairs of event mentions in the training corpus. This set contains 3,270 positive instances of coreference, which are identified by ACE annotators, and the remaining 29,702 negative instances. We extracted the total of 7,010 feature values from these pairs.

- Test corpus

  We applied the same process to the test corpus and found 8,259 event mention pairs including 551 positive instances of co-referential pairs as identified by ACE annotators and 7708 negative instances.
5.4.2 ACE event scoring system

In order to compare our results against those obtained by other IE systems participating in ACE, we score BEAR runs using the ACE value metric, which combines credits for complete and partial matches against the human prepared answer key with penalties for false alarms and wrongly matched events. In this session, we discuss the value metric in detail.

Table 5.2 ACE value based scoring

<table>
<thead>
<tr>
<th>Step</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>[ V_{DR_Value}^{sys} = \frac{\sum_{i} value_of_sys_token_i}{\sum_{j} value_of_ref_token_j} ]</td>
</tr>
<tr>
<td>2.</td>
<td>[ Value(token) = Element_Value(token) \cdot Arguments_Value(token) ]</td>
</tr>
<tr>
<td>3.</td>
<td>[ Element_Value(sys) = \begin{cases} \min \left( \prod_{attribute=type,modality} AttrValue(attribute_{sys}) \right) \cdot \prod_{attribute=type,subtype,modality,genericty,polarity,issue} W_{err-attribute} \text{ if mapped} \ AttrValue(attribute_{sys}) \cdot W_{EA} \text{ if not mapped} \end{cases} ]</td>
</tr>
<tr>
<td>4.</td>
<td>[ Arguments_Value(sys) = \begin{cases} \sum_{all<del>arg_{sys}} \sum_{all</del>docs<del>that</del>mention<del>the</del>relation} MAV_{doc}(arg_{sys},arg_{ref}) \text{ if arithmetic averaging} \ \prod_{all<del>arg_{sys}} \sum_{all</del>docs<del>that</del>mention<del>the</del>relation} MAV_{doc}(arg_{sys},arg_{ref}) \text{ if geometric averaging} \end{cases} ]</td>
</tr>
<tr>
<td>5.</td>
<td>[ MAV_{doc}(arg_{sys}) = Value_{doc}(arg_{sys},arg_{ref}) \cdot W_{err-rect} - (Value_{doc}(arg_{sys},arg_{ref}) - Value_{doc}(arg_{sys},arg_{ref})).W_{EA} ]</td>
</tr>
</tbody>
</table>

Table 5.2 gives the formulas\(^{48}\) that are used to calculate the value of a system. The overall value for a system is defined to be the sum of the values of all of the system’s (sys) output tokens (extracted events), normalized by the sum of the values of all reference (ref) tokens (annotated events) (Formula 1.). The maximum possible VDR (The ACE Event Detection and Recognition task) value score of a system is 100 percent of human annotations. The value of each system output token is based on combination of Element_Value and Arguments_Value.

(Formula 2). The Element Value (Formula 3) for scoring attributes of an event is to check whether the attributes (type and modality) of a system extracted event and the reference event can be matched. If an extracted event and a reference event overlap at least one argument (mapped), the first option in 3 will be used to calculate the element value. AttrValue is defined in the first row in Table 5.3. If any of attribute matching is wrong, the final Element Value will multiply a penalty, $W_{\text{err-attribute}}$ for the wrong match (the fourth row in Table 5.3). If a system extracted event could not be mapped by any reference event, the extracted event is a false alarm and the second option in Formula 3 will be used. $W_{\text{FA}}$ is defined in the fourth row in Table 5.3. The Argument Value (Formula 4) is for scoring the selection of event arguments. It uses two different averaging options: arithmetic averaging and geometric averaging. Arithmetic averaging sums up each result of mutual argument value ($MAV_{\text{doc}}$) and geometric averaging multiplies results of $MAV_{\text{doc}}$. ACE evaluation uses Arithmetic averaging formula (the sixth row in Table 5.3). $MAV_{\text{doc}}$ (Formula 5) is the formula to calculate the score of each event argument extracted by system. If an extracted argument is matched by the corresponding reference event, then the first part of Formula 5 is used while the second part becomes zero. However, if the system assigns a wrong role to an argument, a penalty, $W_{\text{err-role}}$ is assessed (the first column at the bottom row of Table 5.3). If an extracted argument has no corresponding role in the reference event, it is counted as a false alarm and the formula turns into $Value_{\text{doc}}(\text{arg}_{\text{sys}}, \text{arg}_{\text{ref}}) * W_{\text{A-FA}}$ where $W_{\text{A-FA}}$ is the false alarm penalty factor (the second column at bottom row of Table 5.3)

---

49 The event attributes are event type, modality, polarity, etc. In ACE scoring system, only event type and modality are evaluated.
5.4.3 Experiment strategies

We designed two experimental evaluations. The first experiment was to evaluate effectiveness of our features and classifiers at detecting coreference between event mentions. We tried different combinations of features in order to obtain optimal performance. The best performing classifier was then used in BEAR to complete the end-to-end task of event extraction and coreference, as required by ACE evaluation.

In the second experiment, we used the classifier selected in the first experiment to identify corefering event mentions and merge them into composite event representation. Then we ran the ACE evaluation program on so obtained composite events and compared our performance to other systems that participated in ACE 2005 competition.

5.4.4 Tuning the coreference classifier on the human-annotated training corpus

The purpose of this section is to describe in more detail the effect of features we selected for training event coreference resolution.

5.4.4.1 Event coreference resolution baseline used
To access the performance of event coreference resolution process in BEAR, we created a baseline classifier using the features described in (Ahn, 2006) and (Grishman, 2005), which represent the current state of the art. Since the features in (Ahn, 2006) are described in detail, we use them to train the baseline system first and then add selected features from NYU’s system at a later stage to see if they affect the performance. In order to train the baseline system we use megam maximum entropy classifier over the ACE training corpus. We then run the trained system against the test corpus. The results are shown in Table 5.4.

<table>
<thead>
<tr>
<th>System</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>EvCr-BaseLine1</td>
<td>0.6496</td>
<td>0.3230</td>
<td>0.4314</td>
</tr>
<tr>
<td>EvCr-BaseLine2</td>
<td>0.6793</td>
<td>0.3575</td>
<td>0.4685</td>
</tr>
</tbody>
</table>

Table 5.4 BEAR baseline

The first entry in Table 5.4 shows performance of the baseline system (EvCr-BaseLine1) using the set of features from Ahn’s system. The second entry (EvCr-BaseLine2) shows performance of the baseline system when selected features from the NYU system are added. EvCr-BaseLine2 has somewhat better performance because the more generic NYU features provide some fallback against data sparseness in the training corpus. BEAR performance improvement in event extraction will be measured against EvCr-Baseline2.

5.4.4.2 Find best features for BEAR coreference

In order to maximize the accuracy of event coreference resolution in BEAR, we experimented with various combinations of features identified in Table 5.1, and also using two types of classifiers: a Support Vector Machine classifier (SVM\[^{51}\]) and a Maximum Entropy classifier (megam), which was also used to train the baseline system. We divided our

---

\[^{50}\] Please refer to http://www.cs.utah.edu/~hal/megam/

\[^{51}\] Please refer to http://svmlight.joachims.org/
set of features into the four categories as discussed in section 5.3 and trained both classifiers using various combinations of these four groups of features. Our objective was to the combination of features and classifies under which the system performance was maximized.

<table>
<thead>
<tr>
<th>Features</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP + TR</td>
<td>0.4255</td>
<td>0.0363</td>
<td>0.0669</td>
</tr>
<tr>
<td>TP + RL</td>
<td>0.6376</td>
<td>0.1597</td>
<td>0.2554</td>
</tr>
<tr>
<td>TP + CT</td>
<td>0.4705</td>
<td>0.0290</td>
<td>0.0547</td>
</tr>
<tr>
<td>TP + TR + RL</td>
<td><strong>0.7025</strong></td>
<td>0.3085</td>
<td>0.4288</td>
</tr>
<tr>
<td>TP + TR + CT</td>
<td>0.4887</td>
<td>0.2359</td>
<td>0.3182</td>
</tr>
<tr>
<td>TP + RL + CT</td>
<td>0.7005</td>
<td>0.2505</td>
<td>0.3690</td>
</tr>
<tr>
<td>TP + TR + RL + CT</td>
<td>0.6437</td>
<td><strong>0.5082</strong></td>
<td><strong>0.5680</strong></td>
</tr>
</tbody>
</table>

Table 5.5 Performance of the Maximum Entropy classifier with feature groups

Table 5.5 shows the performance of various combinations of features with the maximum entropy classifier. TP (Type), TR (Trigger), RL (Role), and CT (Context) are the four groups of features (see Table 5.1). Since TP has only one feature defined, we always combine it with other groups. We did not anticipate that features from only one group would work well on their own; therefore we start by combing two groups, then three groups and finally all four groups together. The highest precision is reached when the classifier is trained on all the features from TP, TR, and RL groups. The highest recall is obtained when all 4 feature sets are used, with recall increasing an additional 20% and the F-score improving by 14 points over TP+TR+RL. Based on these results we decided to use the complete set of features to train coreference resolution module in BEAR. Similar experiments with the SVM classifier also showed that the combined set of features achieves the best performance, although the overall performance was not as good as with the ME classifier (see Table 5.6).

<table>
<thead>
<tr>
<th>Classifiers</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>EvCr-BaseLine2 (ME)</td>
<td><strong>0.6793</strong></td>
<td>0.3575</td>
<td>0.4685</td>
</tr>
<tr>
<td>BEAR-EvCr1 (ME)</td>
<td>0.6325</td>
<td><strong>0.5082</strong></td>
<td><strong>0.5680</strong></td>
</tr>
<tr>
<td>BEAR-EvCr2 (SVM)</td>
<td></td>
<td>0.4029</td>
<td>0.4922</td>
</tr>
</tbody>
</table>
Table 5.6 The performance of the complete feature set under two different classifiers compared to the baseline.

5.4.5 Comparing BEAR performance against ACE systems

After building an event coreference model from the training corpus (using the full set of features and the ME classifier), we apply it to the event mentions that were extracted from the test corpus using the set of event patterns obtained after 10 unsupervised learning iterations (Chapter 4). This way, we combine the event mention extraction module and the coreference resolution module into the BEAR system so that it can be tested against ACE 2005 diagnostic dataset. Using the same experimental setup as described in Chapter 4, we run BEAR using various pattern learning accuracy thresholds at 0.1 decrements (staring from 0.9 to 0). Figure 5.7 shows the final ACE value score for end-to-end runs using different pattern learning thresholds. The red curve represents BEAR overall Value score with only the first supervised learning step (BEAR base run) followed by event coreference resolution step. The blue line represents BEAR’s overall value score with the additional 10 unsupervised learning iterations followed by the event coreference resolution. When the pattern learning threshold is lowered from 0.9 to 0.4, the shape of the Value curve is similar as the F-score curve (measuring extraction of event mentions, see Chapter 4). The Value curve drops off sharply after pattern learning threshold moves below 0.4. The main reason for this sharp decline is that the number of false alarms increases dramatically when the threshold drops below 0.4 and is additionally amplified by false event coreferences. In addition, ACE Value score (Table 5.2) penalizes false alarms more heavily than the standard F-score.

---

52 The diagnostic task provides annotated entities/time/values in both training and test data.
Table 5.7 shows the best performance obtained with the current version of BEAR on the ACE 2005 test set, and it is compared favorably against the top three ACE systems. The second column of the table shows the overall results of the systems. It should be noted that
Chapter 5

our bootstrapped learning process helps BEAR to learn event extraction patterns so rapidly that it can outperform the best current IE systems.

From column 3 to 8, the table shows the system performance on each sub-corpus which is from different data sources. BEAR gets better score at 5 sub-corpuses and only one lower score comparing with the best system. We also run the Wilcoxon Signed Ranks test\textsuperscript{53} at sub-corpus level and it shows a significant difference at p\textless 0.05 between BEAR and the best system.

5.5. Conclusion and Future plans

In this chapter, we described the event coreference resolution process in BEAR. We selected a comprehensive set of features and run experiments with two classifiers using different subsets of the feature set. We found that the Maximum Entropy classifier works better for this task than the SVM classifier, and that its performance peaks when all available features are used.

Finally, we compared BEAR with other systems that participated in ACE 2005 diagnostic task. This evaluation demonstrated the effectiveness of our bootstrapping techniques: our system performs better than any contemporary IE system.

In this thesis we presented a semi-supervised machine learning process by which BEAR automatically learns event extraction from un-annotated text following an initial supervised learning step. Unlike traditional supervised machine learning methods, our techniques can adapt to very different event structures even quite unlike these encountered in the training

\textsuperscript{53} http://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test
data. Thus, our techniques can be easily applied to different data set and new domains. Specifically, our contributions include:

- **Flexible Pattern structure**
  For ease of pattern manipulation, each BEAR event pattern contains a list of chain-based sub-patterns, which are built based on the structure linking the event trigger to its roles. We explained how BEAR patterns can be easily modified to refit them to new syntactic structures encountered outside of the training set (Chapter 3).

- **A novel method for sense disambiguation of event triggers**
  We introduced a novel technique to determine the correct sense of an event trigger through Wordnet traversal. (Chapter 3)

- **Pattern validation methods**
  We developed a novel pattern validation method computes the projected accuracy of a pattern and whenever possible repairs patterns by inserting contextual constraints into it. We also described how to validate event triggers and sub-pattern, so that they can be used to derive new patterns. (Chapter 3 and 4)

- **Techniques to learn new patterns**
  We presented multiple methods that manipulate the structure of previously learned patterns in order to learn alternative extraction patterns.
Several follow up research directions may be considered:

- **Improving pattern discovery techniques**
  
  Additional pattern learning techniques are needed to recognize event descriptions that still evade the methods described in this thesis, especially for the events defined in *Personnel*, *Business*, and *Transactions* where recall is relatively lower than for other types of events.

- **Finding more role indicators**

  In this thesis, we used prepositions that frequently co-occur with specific roles as indicators of unseeded event roles. Other similar indicators may exist. One possible role indicator is that when one entity acts multiple roles in different events, one of its roles indicates another role it acting. For example, in the following sentence, “*Trial begins for Piscataway man accused of killing 12-year-old girl*”, two events, Life-Die event and Justice-Charge-Indict event are mentioned. “*Piscataway man*” plays different roles in these events: he is Agent in the Life-Die event and Defendant in the Justice-Charge-Indict event. If such role co-occurrence happens frequently enough, the Defendant of the Charge-Indict event and Agent can be the indicator of the other one. So our future research will utilize statistical methods to find more role indicators from event contexts.

- **Expand pattern leaning to cross-document event coreference**

  One technique used in this thesis learns additional event structures by exploiting referential consistency within a single document. However, it is also common that the
same news story is reported by multiple sources, so in our future studies, we would like to find a way to exploit such cross-document co-references to learn more patterns.

- **Adapting the bootstrapping method to different languages**

  We’d like to see how robust our bootstrapping approach is when applied to data in a language other than English. One intriguing possibility is to have BEAR learn event extraction from a corpus in one language and then “translate” the patterns onto the corpus in another language (e.g., a parallel corpus, or same-content corpus, or via a cross-lingual information retrieval technique) in order to establish seeds in the target language corpus. The bootstrapping process could then be applied over the partially seeded target language corpus, thus yielding event extraction in a new language essentially without any human intervention.

- **Link multiple events in one document to form a scenario**

  Another future direction is to expand BEAR to extract larger “scenario events”, i.e., groups of correlated events that form coherent “stories” often described in larger sections of text. For example, there are five different types of events described in Figure 5.8, including two Attack event and six Justice events in 1999 and 2002. If all these events can be extracted together, we could gain much better understanding of the Toefting character than reading any single event mention in which he was involved.

  "Toefting was **convicted** in October 2002 of **assaulting** a pair of restaurant workers during a night out with national squad teammates in the capital, Copenhagen. He was **sentenced** to four months in prison, but **appealed**. The **hearing** was scheduled for April 10.

  

54 Of course NLP tools will still be required for the target language.
Toefting has been convicted before. In 1999 he was given a 20-day suspended sentence for assaulting a fan who berated him for playing with German club Duisburg.”

Figure 5.8 A scenario about Toefting
References:


Grishman, Ralph. 1995. The NYU system for MUC-6 or where's the syntax? In Proceedings of the 6th conference on Message understanding, 167-175, Columbia, Maryland: Association for Computational Linguistics

Grishman, Ralph, and Beth Sundheim. 1995. Design of the MUC-6 evaluation. 1-11, Columbia, Maryland: Association for Computational Linguistics

Grishman, Ralph, and Beth Sundheim. 1996. Message Understanding Conference-6: a brief history. 466-471, Copenhagen, Denmark: Association for Computational Linguistics


Krupka, George, Paul Jacobs, Lisa Rau, and Lucja Iwańska. 1991. GE: description of the NLTooLSET system as used for MUC-3. 144-149, San Diego, California: Association for Computational Linguistics


Sundheim, Beth M. 1992. Overview of the fourth message understanding evaluation and conference. 3-21, McLean, Virginia: Association for Computational Linguistics

Swanson, Reid and Andrew S. Gordon. 2006. A Comparison of Alternative Parse Tree Paths for Labeling Semantic Roles. Accepted Posters, COLING/ACL.


Yangarber, R., and R. Grishman. 1998. NYU: Description of the Proteus/PET System as Used for MUC-7 ST. In *Proceedings of the 7th conference on Message understanding*.


