A Study on Recognition of Requisite Part and Effectuation Part in Law Sentences

By Ngo Xuan Bach

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Akira Shimazu

March, 2011

A Study on Recognition of Requisite Part and Effectuation Part in Law Sentences

By Ngo Xuan Bach (0910021)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Akira Shimazu

and approved by
Professor Akira Shimazu
Associate Professor Kiyoaki Shirai
Professor Satoshi Tojo

February, 2011 (Submitted)

A Study on Recognition of Requisite Part and Effectuation Part in Law Sentences

by

Ngo Xuan Bach (0910021)

School of Information Science

Japan Advanced Institute of Science and Technology

February 08, 2011

Keywords: Law Sentences, Requisite Part, Effectuation Part, Sequence Learning, Semi-supervised Learning

Abstract

In recent years, a new research field called Legal Engineering has been proposed. Legal Engineering regards laws as a kind of software for our society. Specifically, laws such as pension law are specifications for information systems such as pension systems. To achieve a trustworthy society, laws need to be verified about their consistency and contradiction.

Analyzing the logical structure of legal texts is a key problem in Legal Engineering. The results of this process are helpful to not only lawyers but also people who want to understand the legal texts. This is a preliminary step to support other tasks in legal text processing (such as translating legal articles into logical and formal representations, legal article retrieval, legal text summarization, question answering in legal domains, etc) and serve to verify legal documents.

In this thesis, we focus on two tasks which analyze the logical structure of legal texts at the sentence level and the paragraph level, respectively: *Recognition of Requisite Part and Effectuation Part in Law Sentences* (or RRE task) and *Recognition of Requisite Parts and Effectuation Parts in Paragraphs Consisting of Multiple Sentences* (or RREP task). The goal of the RRE task is to recognize logical parts given a law sentence. The goal of the RREP task is to recognize logical parts and logical structures (a set of some related logical parts) given a legal paragraph.

For the RRE task, our approach is modeling the task as a sequence learning problem. We present several supervised learning models for the RRE task: word-based model (consider a law sentence as a sequence of words), Bunsetsu-based model (consider a law sentence as a sequence of Bunsetsus), and reranking model (use a linear score function to rerank N-best outputs of the Bunsetsu-based model with a variant of the perceptron algorithm). We also present a simple semi-supervised learning method for the RRE task using extra word features derived from Brown word clusters. Our results show that modeling based on Bunsetsu is better than modeling based on words, and the semi-supervised learning method outperforms supervised models. In this task, our best model achieves 88.84% in $F_{\beta=1}$ score on the Japanese National Pension Law corpus.

For the RREP task, we present a two-phase framework in which we recognize logical parts in the first phase and logical structures in the second phase. We divide logical parts in a law sentence into some layers and provide a multi-layer sequence learning model to recognize them. We consider the sub-task of recognizing logical structures as an optimization problem on a graph, where each node corresponds to a logical part and a sub-graph corresponds to a logical structure, and give a heuristic algorithm to solve it. Our models achieve 74.37% in recognizing logical parts, 75.89% in recognizing logical structures, and 51.12% in the whole task on the Japanese National Pension Law corpus. Our results provide a baseline for further researches on this interesting task.

Acknowledgments

First and foremost, I would like to sincerely thank my supervisor, Professor Akira Shimazu, who brought me to the interesting world of natural language processing and supported me during my study. Without his guidance and encouragement my work would not be done.

Second, I would like to say many thanks to Japanese Government Scholarship Program. Without their financial support I would not come to Japan to study.

I would like to thank Professor Kiyoaki Shirai. His valuable comments made this thesis better.

I would like to thank Dr. Nguyen Le Minh, who gave me advices when I was in troubles. He always listened to me, discussed with me, and supported me everything I need.

I am grateful to my colleagues in natural language processing laboratory for making JAIST an enjoyable place to my study and life.

I also would like to thank Kenji Takano and Yoshiko Oyama for analyzing law sentences and building the corpus, and Technical Communication Program Office of JAIST for editing this manuscript.

Finally, I would like to give special thanks to my family for their sacrifice, love, and understanding.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we first introduce our research tasks, *Recognition of Requisite Part and Effectuation Part in Law Sentences*, or RRE task (Section 1.1), and *Recognition of Requisite Parts and Effectuation Parts in Paragraphs Consisting of Multiple Sentences*, or RREP task (Section 1.2). Next, we describe some related works (Section 1.3). Finally, we present contributions and the outline of this thesis (Section 1.4).

1.1 RRE task

In recent years, a new research field called Legal Engineering has been proposed in the 21st Century COE Program, Verifiable and Evolvable e-Society [15, 16, 17]. Legal Engineering serves to exam and verify whether a law has been established appropriately according to its purpose, whether the law is consistent with related laws, and whether the law has been modified, added, and deleted consistently. There are two important goals of Legal Engineering. The first goal is to help experts make complete and consistent laws, and the other is to design an information system which works based on laws.

Legal Engineering regards laws as a kind of software for our society. Specifically, laws such as pension law are specifications for information systems such as pension systems. To achieve a trustworthy society, laws need to be verified about their consistency and contradiction.

Legal texts have some specific characteristics that make them different from other daily-use documents. Legal texts are usually long and complicated. They are composed by experts who spent a lot of time to write and check carefully.

One of the most important characteristics of legal texts is that legal texts usually have some specific structures. In most cases, a law sentence can roughly be divided into two logical parts: *requisite part* and *effectuation part* [30, 40]. For example, the Hiroshima city provision 13-2 *When the mayor designates a district for promoting beautification, s/he must in advance listen to opinions from the organizations and the administrative agencies* *which are recognized to be concerned with the district*, includes a requisite part (before the comma) and an effectuation part (after the comma) [30].

The requisite part and the effectuation part of a law sentence are composed from three parts: a *topic part*, an *antecedent part*, and a *consequent part*. There are four cases (illustrated in Figure 1.1) basing on where the topic part depends on: case 0 (no topic part), case 1 (the topic part depends on the antecedent part), case 2 (the topic part depends on the consequent part), and case 3 (the topic part depends on both the antecedent part and the consequent part). In case 0, the requisite part is the antecedent part and the effectuation part is the consequent part. In case 1, the requisite part is composed from the topic part and the antecedent part, while the effectuation part is the consequent part. In case 2, the requisite part is the antecedent part, while the effectuation part is composed from the topic part and the consequent part. In case 3, the requisite part is composed from the topic part and the antecedent part, while the effectuation part is composed from the topic part and the consequent part. Figure 1.2 gives examples of law sentences in four cases.

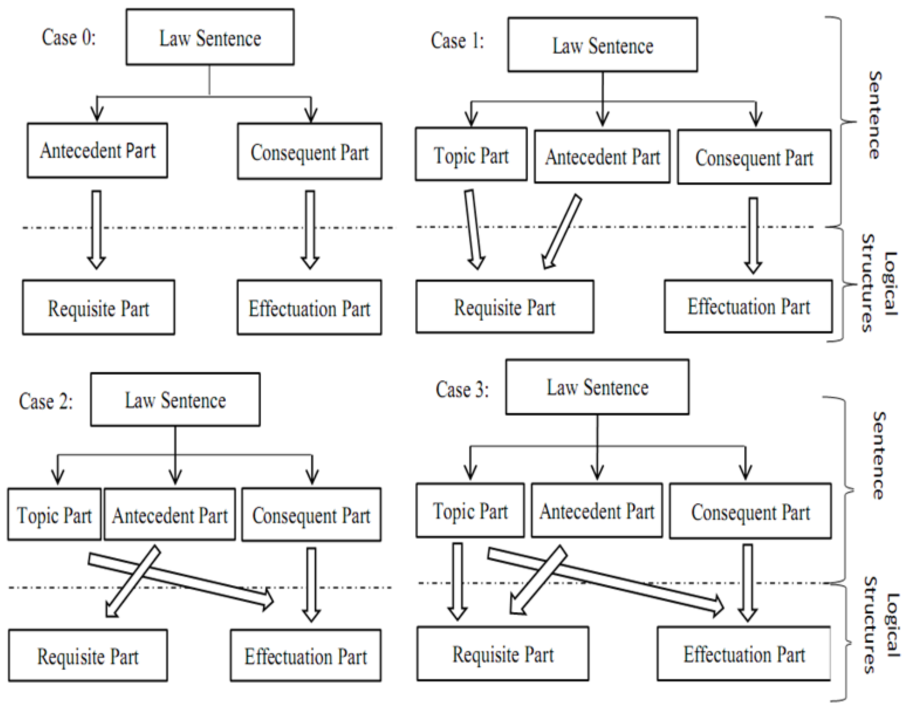
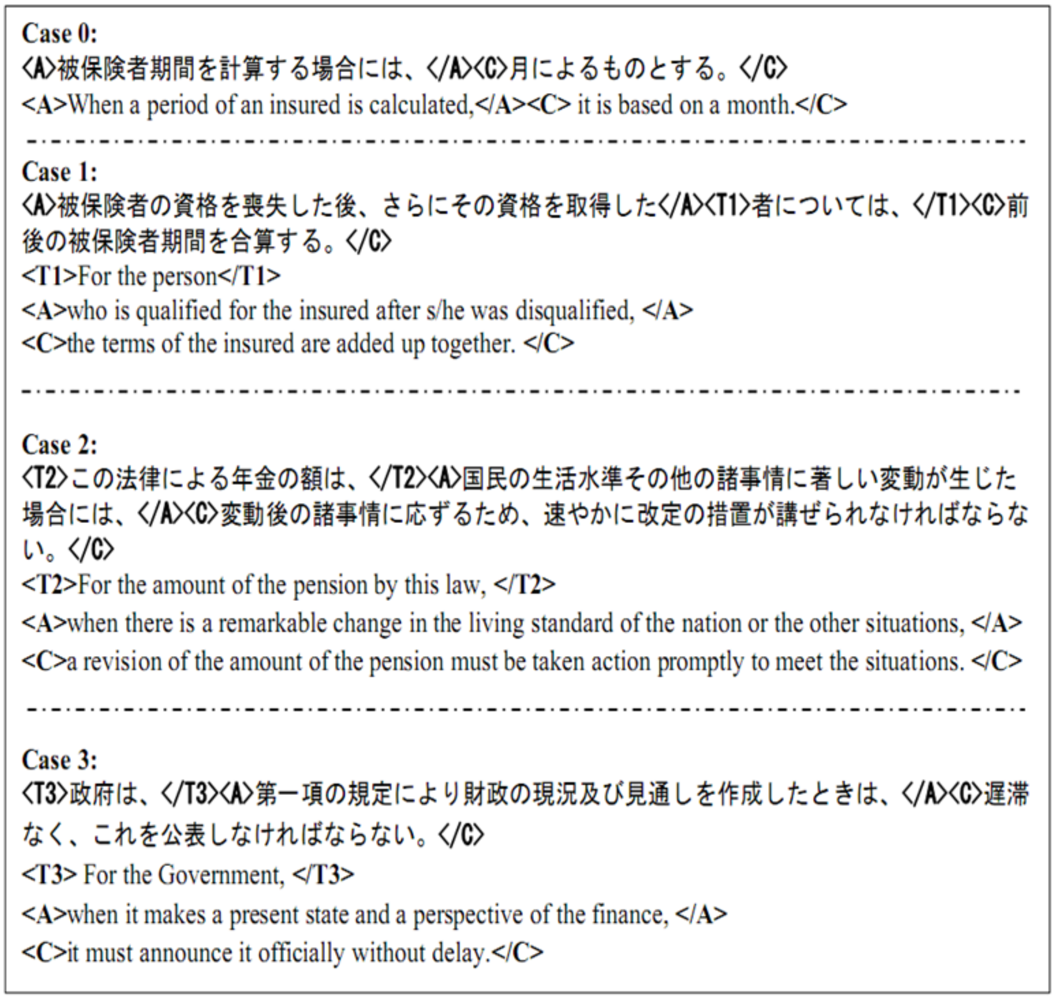


Figure 1.1: Four cases of the logical structure of a law sentence.

Recognition of Requisite Part and Effectuation Part in Law Sentences, or RRE, is the task of analyzing the logical structure of law sentences. The input of the task is a law sentence, and the output are *non-overlapping* and *non-embedded*[1] *logical parts* of the input

[1]This means that one word can only belong to one logical part, and there is no logical part that contains another logical part. These notions will be formally explained in Chapter 2.


Case 0:
<A>被保険者期間を計算する場合には、</A><C>月によるものとする。</C>
<A>When a period of an insured is calculated,</A><C> it is based on a month.</C>

Case 1:
<A>被保険者の資格を喪失した後、さらにその資格を取得した</A><T1>者については、</T1><C>前後の被保険者期間を合算する。</C>
<T1>For the person</T1>
<A>who is qualified for the insured after s/he was disqualified, </A>
<C>the terms of the insured are added up together. </C>

Case 2:
<T2>この法律による年金の額は、</T2><A>国民の生活水準その他の諸事情に著しい変動が生じた場合には、</A><C>変動後の諸事情に応ずるため、速やかに改定の措置が講ぜられなければならない。</C>
<T2>For the amount of the pension by this law, </T2>
<A>when there is a remarkable change in the living standard of the nation or the other situations, </A>
<C>a revision of the amount of the pension must be taken action promptly to meet the situations. </C>

Case 3:
<T3>政府は、</T3><A>第一項の規定により財政の現況及び見通しを作成したときは、</A><C>遅滞なく、これを公表しなければならない。</C>
<T3> For the Government, </T3>
<A>when it makes a present state and a perspective of the finance, </A>
<C>it must announce it officially without delay.</C>

Figure 1.2: Examples of four cases of the logical structure of a law sentence. *A* means *antecedent part*, *C* means *consequent part*, and *T1*, *T2*, *T3* mean *topic parts* which correspond to case 1, case 2, and case 3 (the translations keep the ordinal sentence structures).

sentence[2]. The RRE task is an important task which has been studied in research on Legal Engineering. This task is a preliminary step to support other tasks in legal text processing such as translating legal articles into logical and formal representations and verifying legal documents, legal article retrieval, legal text summarization, question answering in legal domains, etc [16, 30]. In a law sentence, the consequent part usually describes a law provision, and the antecedent part describes cases in which the law provision can be applied. The topic part describes subjects which are related to the law provision. Hence, the outputs of the RRE task will be very helpful to not only lawyers but also people who want to understand the law sentence. They can easily understand 1) what does a law sentence say? 2) in which cases the law sentence can be applied? and 3) which subjects

[2]As shown in Figure 1.2, the outputs are topic parts (together with their case numbers), antecedent parts, and consequent parts.

are related to the provision described in the law sentence?

1.2 RREP Task

In the previous section (Section 1.1), we have introduced the RRE task, in which we recognize logical parts of law sentences. In this section, we introduce the RREP task, *Recognition of Requisite Parts and Effectuation Parts in Paragraphs Consisting of Multiple Sentences*, in which we analyze the logical structure of legal text at the paragraph level. The input of this task is a paragraph in a legal article, and the outputs are *logical parts* in law sentences and *logical structures* between logical parts (each logical structure is a set of some related logical parts).

Figure 1.3 shows two cases of inputs and outputs of the task. In the first case, the input is a paragraph of two sentences, and the outputs are four logical parts, which are grouped into two logical structures. In the second case, the input is a paragraph consisting of four sentences, and the outputs are four logical parts, which are grouped into three logical structures.

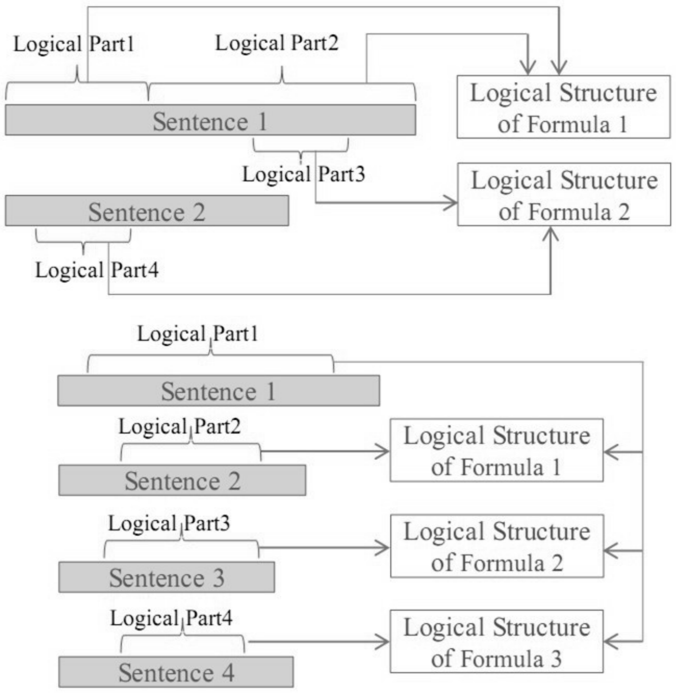


Figure 1.3: Two cases of inputs and outputs of the RREP task.

This task consists of two sub-tasks as follows:

1. Sub-task 1: *Recognition of Logical Parts*. The goal of this sub-task is recognizing (*non-overlapping*) logical parts of law sentences in the input paragraph.

2. Sub-task 2: *Recognition of Logical Structures.* The goal of this sub-task is grouping related logical parts (outputs of Sub-task 1) into some logical structures.

In the RRE task, the input is only one sentence, so we assume that all the logical parts of this sentence belong to the same logical structure. In this task, we consider more general cases, in which the input is a paragraph consisting of multiple sentences (maybe complete or non-complete) and logical parts of an input sentence can belong to different logical structures.

An example in natural language is presented in Figure 1.4. In this example, the input paragraph consists of four sentences[3]. The goal of the first sub-task is to recognize five logical parts including one topic part and one consequent part in the first sentence, and three antecedent parts in remainder sentences. In the second sub-task, the goal is to recognize three logical structures given five logical parts. Each logical structure consists of the topic part, the consequent part, and one antecedent part.

Each logical structure will form a logical formula[4]. The intuitive meaning of three logical formulas can be expressed as follows:

1. *The meaning of formula 1*: If a person in my company cannot complete his/her job, he/she will be sacked without warning.
2. *The meaning of formula 2*: If a person in my company goes to work late three times or more, he/she will be sacked without warning.
3. *The meaning of formula 3*: If a person in my company uses the Internet in working time, he/she will be sacked without warning.

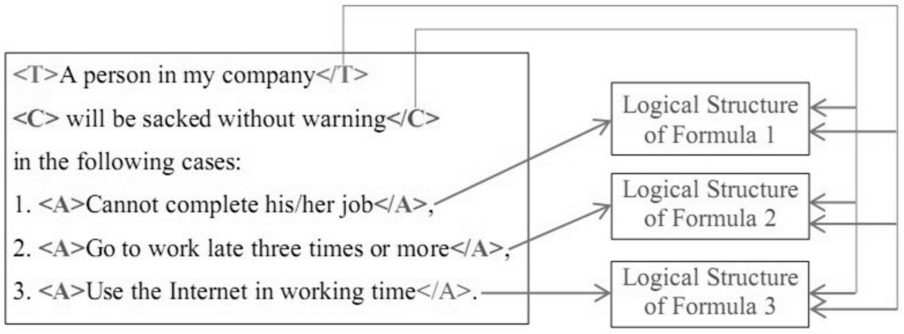

<T>A person in my company</T>
<C> will be sacked without warning</C>
in the following cases:
1. <A>Cannot complete his/her job</A>,
2. <A>Go to work late three times or more</A>,
3. <A>Use the Internet in working time</A>.
Logical Structure of Formula 1
Logical Structure of Formula 2
Logical Structure of Formula 3

Figure 1.4: An example of the RREP task in natural language (T means *Topic part*, A means *Antecedent part*, and C means *Consequent part*).

[3]Some sentences are non-complete.
[4]The task of building logical formulas is not covered in this thesis.

1.3 Related Works

This section presents some related works in Legal Engineering and some tasks in natural language processing (NLP) which are similar to our tasks.

There have been some studies on analyzing logical structures of legal texts. Nakamura et al. (in [30]) describe a rule-based system, which translates legal sentences into logical forms. They built the system based on linguistic analysis of Japanese legal documents. Their system achieved 78% in accuracy in terms of deriving predicates with bound variables. Kimura et al. (in [18]) present a study on how to deal with legal sentences including itemized and referential expressions. At the paragraph level, Takano et al. (in [39]) classify a legal paragraph into one of six predefined categories: A, B, C, D, E, and F. Among six types, Type A, B, and C correspond to cases in which the main sentence is the first sentence, and subordinate sentences are other sentences. In paragraphs of Type D, E, and F, the main sentence is the first or the second sentence, and a subordinate sentence is an embedded sentence in parentheses within the main sentence.

In the RRE task, we want to recognize some kinds of parts in an input sentence. There are some NLP tasks which are quite similar to our RRE task.

1. *Named entity recognition* [29, 35]. Named entities (NEs) are phrases that contain the names of persons, organizations, locations, expressions of time, percentages, quantities, etc. Named entity recognition (NER) is the task in which we locate named entities in texts and classify them into predefined categories[5]. For example:

   [PERSON Jim] bought [QUANTITY 300] shares of [ORGANIZATION Acme Corp.] in [TIME 2006][6].

   This sentence contains four named entities: *Jim* is a person, *300* is a quantity, *Acme Corp.* is an organization, and *2006* is an expression of time. Usually, the NER task is modeled as a sequence learning problem where its input is a sequence of words as follows:

   Jim/B-PERSON bought/O 300/B-QUANTITY shares/O of/O
   Acme/B-ORGANIZATION Corp./I-ORGANIZATION in/O 2006/B-TIME ./O[7]

2. *Text chunking* [33]. Chunks are phrases of syntactically related words. Text chunking is the task of dividing a text into non-overlapping phrases or chunks. This means that one word can only belong to one chunk. For example:

   [ADVP However] , [NP Mr. Dillow] [VP said] [NP he] [VP believes] [SBAR that] [NP a reduction] [PP in] [NP raw material stockbuilding] [PP by] [NP industry] [VP could lead] [PP to] [NP a sharp drop] [PP in] [NP imports] .[8]

[5]Each NER system only can recognize some predefined categories of named entities.

[6]This example is picked from Wikipedia.

[7]Words tagged with B-X begin an named entity of type X. Words tagged with I-X are inner an named entity of type X. Words tagged with O are outside named entities.

[8]This example is picked from CoNLL-2000 shared task data [33].

In text chunking task, we consider following types of chunks: NP (noun phrase), VP (verb phrase), PP (prepositional phrase), ADVP (adverb phrase), SBAR (subordinated clause), ADJP (adjective phrase), PRT (particles), CONJP (conjunction phrase), INTJ (interjection), LST (list marker), and UCP (unlike coordinated phrase). Usually, text chunking task is also modeled as a sequence labeling problem as follows:

However/B-ADVP ,/O Mr./B-NP Dillow/I-NP said/B-VP he/B-NP believes/B-VP that/B-SBAR a/B-NP reduction/I-NP in/B-PP raw/B-NP material/I-NP stockbuilding/I-NP by/B-PP industry/B-NP could/B-VP lead/I-VP to/B-PP a/B-NP sharp/I-NP drop/I-NP in/B-PP imports/B-NP ./O

3. *Clause identification* [10, 34]. Clauses are natural structures above chunks: *It is a hypothesis of the author's current clause-by-clause processing theory, that a unit corresponding to the basic clause is a stable and easily recognizable surface unit and that is also an important partial result and building block in the construction od a richer linguistic representation that encompasses syntax as well as semantics and discourse structure* ([10], page 220). For example:

   (Coach them in (handling complaints) (so that (they can resolve problems immediately)) .)[9]

   This sentence contains four clauses:

   - handling complaints
   - they can resolve problems immediately
   - so that they can resolve problems immediately
   - Coach them in handling complaints so that they can resolve problems immediately.

   Usually, clause identification task is divided into three sub-tasks:

   - *Start identification.* The goal of this sub-task is to detect starts of clauses. Usually, this sub-task is modeled as a sequence learning problem:

     Coach/S them/X in/X handling/S complaints/X so/S that/X they/S can/X resolve/X problems/X immediately/X ./X

     In this encoding scheme, words starting a clause are tagged with S, and other words are tagged with X.

   - *End identification.* The goal of this sub-task is to detect ends of clauses. Usually, this sub-task is modeled as a sequence learning problem:

     Coach/X them/X in/X handling/X complaints/E so/X that/X they/X can/X

[9]This example is picked from CoNLL-2001 shared task data [34].

resolve/X problems/X immediately/E ./E

In this encoding scheme, words ending a clause are tagged with E, and other words are tagged with X.

- *Clause identification.* The goal of this sub-task is to identify all clauses in the input text. This sub-task can also be modeled as a sequence learning problem:

  Coach/(S* them/* in/* handling/(S* complaints/*S) so/(S* that/* they/(S* can/* resolve/* problems/* immediately/*S)S) ./*S)

  In this encoding scheme, (S*, *S), and * denote a clause start, a clause end, and neither a clause start nor a clause end, respectively. (S* and *S) can be used in combination with each other. For example, (S*S) marks a word where a clause starts and ends, and *S)S) marks a word where two clauses end [34].

A comparison between our RRE task (and Sub-task 1 of the RREP task) and above tasks is shown in Table 1.1.

Table 1.1: A comparison between our RRE task (and Sub-task 1 of the RREP task) and some NLP tasks

Task	Input	Outputs	Overlapping	Embedded
RRE	A sentence	Logical parts	No	No
Sub-task 1 of RREP	A paragraph	Logical parts	No	Yes
NER	A sentence	NEs	No	No
Text Chunking	A sentence	Chunks	No	No
Clause Identification	A sentence	Clauses	No	Yes

In the RREP task, in addition to recognizing logical parts, we also recognize logical structures (Sub-task 2). This sub-task is quite similar to the *coreference resolution* task [13, 31]. Coreference is the phenomenon in which multiple expressions in a text refer to the same thing. For example:

Kaka has revealed **he** thought **he** would never play again after undergoing knee surgery last summer.[10]

In this sentence, **Kaka** and two words **he** are most likely referring to the same person. In coreference resolution task, we cluster mentions (each mention is a phrase, typically a noun phrase) in a text according to the underlying referent entity. Usually, we consider three types of mentions: proper (Kaka), nominal (football player), and pronominal (he).

However, our Sub-task 2 of the RREP task and coreference resolution task have some significant differences. The greatest difference is that our task accepts one logical part belonging to multiple logical structures. We describe these differences in Table 1.2.

[10]This sentence is picked from BBC News.

Table 1.2: A comparison between Sub-task 2 of the RREP task and coreference resolution task

Feature to be compared	Coreference resolution	Sub-task 2 of RREP
Input items	mentions in texts	logical parts in paragraphs
Outputs	groups of mentions	groups of logical parts
Relations between items	coreference to an entity	logical relation
Number of items in each group	possibly one mention	at least two logical parts
Items in multiple groups	no	yes

1.4 Contributions and the Outline of the Thesis

Our main contributions can be summarized in the following points:

1. Introducing two tasks to legal text processing:
   - *Recognition of Requisite Part and Effectuation Part in Law Sentences*, or RRE task.
   - *Recognition of Requisite Parts and Effectuation Parts in Paragraphs Consisting of Multiple Sentences*, or RREP task.
2. Presenting an annotated corpus for the tasks, the *Japanese National Pension Law corpus*:
   - Single sentences for the RRE task.
   - Paragraphs for the RREP task.
3. Proposing solutions to solve the tasks.
   - Word-based model, Bunsetsu-based model, reranking model, and simple semi-supervised method for the RRE task.
   - Two-phase framework, multi-layer sequence learning model for Sub-task 1, and graph-based model for Sub-task 2 of the RREP task.
4. Evaluating our solutions on the real annotated corpus.

The remainder of this thesis is organized as follows.

Chapter 2 presents how to model the RRE task as a sequence learning problem, gives a brief introduction to Conditional random fields (the learning method which we choose for the RRE task), and introduces our corpus and evaluation methods for the RRE task.

Chapter 3 describes our word-based model for the RRE task, feature investigation, and experiments on the word-based model. This chapter also presents an exploring on contributions of words to the RRE task.

Chapter 4 describes our Bunsetsu-based model for the RRE task and experiments on this model. This chapter also presents our reranking model, in which we use a variant of the perceptron algorithm to rerank N-best outputs of the Bunsetsu-based model.

Chapter 5 considers the RRE task in the view of semi-supervised learning. This chapter presents a simple semi-supervised learning method for the RRE task using extra word features derived from Brown word clusters.

Chapter 6 investigates the task of recognition of requisite parts and effectuation parts in paragraphs consisting of multiple sentences (the RREP task). This chapter begins with the formulation section. Next, the chapter presents our two-phase framework to solve two sub-tasks: multi-layer sequence learning model for Sub-task 1, and graph-based model for Sub-task 2. Then the chapter describes our experiments on the Japanese National Pension Law corpus. The chapter ends with some conclusions.

Chapter 7 summarizes this thesis and discusses future works.

Chapter 2

RRE as a Sequence Learning Problem

In this chapter, we first present how to formulate the RRE task as a sequence learning problem (Section 2.1). Then, we give an introduction to Conditional random fields (CRFs), the learning method which we choose for our task, and provide some explanations why we choose CRFs (Section 2.2). Finally, we describe our corpus, the Japanese National Pension Law (JNPL) corpus, and evaluation methods for the RRE task (Section 2.3). This corpus was used in all experiments on the RRE task.

2.1 Formulation

Let x be an input law sentence in a law sentence space X, then x can be represented by a sequence of elements $[w_1 w_2 \ldots w_n]$. An element can be a word or a Bunsetsu[1]. Let P be the set of predefined logical part categories. A logical part $p(s, e)$ is the sequence of consecutive elements spanning from element w_s to element w_e with category $p \in P$.

Let $p_1(s_1, e_1)$ and $p_2(s_2, e_2)$ be two different logical parts of one sentence x. We define two kinds of relationships between two logical parts: *overlapping* and *embedded*. We say that $p_1(s_1, e_1)$ and $p_2(s_2, e_2)$ are *overlapping* if and only if $s_1 < s_2 \leq e_1 < e_2$ or $s_2 < s_1 \leq e_2 < e_1$. We denote the *overlapping* relationship by $\sim$. We also say that $p_1(s_1, e_1)$ is *embedded* in $p_2(s_2, e_2)$ if and only if $s_2 \leq s_1 \leq e_1 \leq e_2$, and denote the *embedded* relationship by $\prec$.

Figure 2.1 illustrates an example of *overlapping* and *embedded* relationships. In this example, the law sentence consists of nine elements $w_1, \ldots, w_9$, and three logical parts $p_1(2, 5)$, $p_2(1, 7)$, and $p_3(7, 9)$. Among three logical parts, p_2 and p_3 are *overlapping*, and p_1 is *embedded* in p_2.

[1]In Japanese, Bunsetsu is a unit of language, which is similar to a chunk in English. We will explain more details about Bunsetsu in the next chapters.

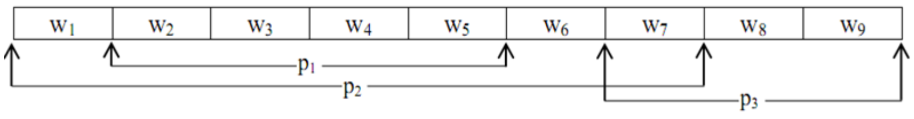


Figure 2.1: An example of *overlapping* and *embedded* relationships.

In the RRE task, we want to split a source sentence into some *non-overlapping* and *non-embedded* logical parts. Let S be the set of all possible logical parts:

$$S = \{p(s,e) | 1 \leq s \leq e \leq n, p \in P\}. \tag{2.1}$$

A *solution* of the RRE task is a subset $y \subseteq S$ which does not violate the *overlapping* relationship and the *embedded* relationship. Formally, the *solution space* can be described as follows:

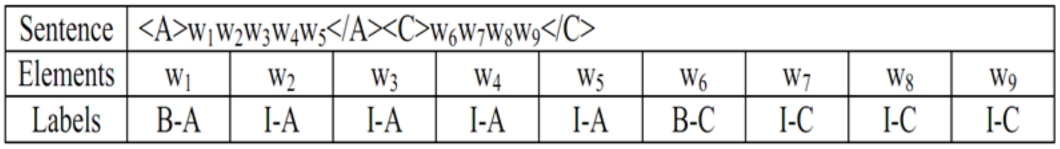
$$Y = \{y \subseteq S | \forall u, v \in y, u \nsim v, u \not\prec v\}. \tag{2.2}$$

The learning problem in the RRE task is to learn a function $R : X \to Y$ from a set of m training samples $\{(x^i, y^i) | x^i \in X, y^i \in Y, \forall i = 1, 2, \ldots, m\}$.

Our RRE task belongs to the class of phrase recognition problems [5]. The task is similar to some other tasks such as named entity recognition (NER) [35] and chunking [33] in the sense that it does not allow *overlapping* and *embedded* relationships. In this sense, it is different from the clause identification task [34] because that task allows the *embedded* relationship. One important characteristic of our task is that the input sentences are usually very long and complicated, so the logical parts are also long.

Sequence learning is a suitable model for phrase recognition problems which do not allow *overlapping* and *embedded* relationships. It has been applied successfully to many phrase recognition tasks such as word segmentation, chunking, and NER. So we choose the sequence learning model for the RRE task.

We model the RRE task as a sequence labeling problem, in which each sentence is a sequence of words or Bunsetsus. Figure 2.2 illustrates an example in IOB notation [26]. In this notation, the first element of a part is tagged by B, the other elements of the part are tagged by I, and an element not included in any part is tagged by O. This sentence consists of an antecedent part (tag A) and a consequent part (tag C).



Sentence	<A>w$_1$w$_2$w$_3$w$_4$w$_5$</A><C>w$_6$w$_7$w$_8$w$_9$</C>								
Elements	w$_1$	w$_2$	w$_3$	w$_4$	w$_5$	w$_6$	w$_7$	w$_8$	w$_9$
Labels	B-A	I-A	I-A	I-A	I-A	B-C	I-C	I-C	I-C

Figure 2.2: A sentence in sequence learning model with IOB notation.

In the RRE task, we consider two types of law sentences: *implication type* and *equivalence type*[2], and seven kinds of logical parts, as follows:

1. Implication sentences:
   - Antecedent part (A)
   - Consequent part (C)
   - Three kinds of topic parts T_1, T_2, T_3 (correspond to case 1, case 2, and case 3)
2. Equivalence sentences:
   - The equivalent left part (EL)
   - The equivalent right part (ER)

In the IOB notation, we will have 15 kinds of tags: B-A, I-A, B-C, I-C, B-T_1, I-T_1, B-T_2, I-T_2, B-T_3, I-T_3, B-EL, I-EL, B-ER, I-ER, and O[3]. For example, an element with tag B-A begins an antecedent part, while an element with tag B-C begins a consequent part.

2.2 Learning Method: Conditional Random Fields

This section provides an introduction to Conditional random fields (CRFs), the learning method which we choose for the RRE task, and explains why CRFs is suitable for the RRE task.

Conditional random fields (CRFs) [23, 38, 45] are undirected graphical models (see Figure 2.3), which define the probability of a label sequence y given an observation sequence x as a normalized product of potential functions. Each potential function has the following form:

$$exp(\sum_j \lambda_j t_j(y_{i-1}, y_i, x, i) + \sum_k \mu_k s_k(y_i, x, i)) \tag{2.3}$$

where $t_j(y_{i-1}, y_i, x, i)$ is a transition feature function (or edge feature), which is defined on the entire observation sequence x and the labels at positions i and $i-1$ in the label sequence y; $s_k(y_i, x, i)$ is a state feature function (or node feature), which is defined on the entire observation sequence x and the label at position i in the label sequence y; and λ_j and μ_k are parameters of the model, which are estimated in the training process.

Each feature function (edge feature and node feature) takes a real value. For example, in the part of speech (POS) tagging problem, the input is a sequence of words, and the

[2]Most of sentences belong to implication type.

[3]Tag O is used for an element not included in any part.

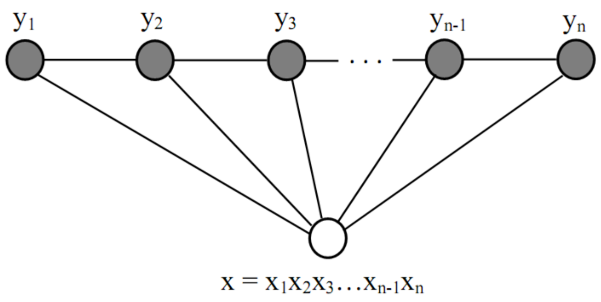


Figure 2.3: Graphical model of a CRFs for sequence learning (n is the length of sequence).

output is a sequence of POS tags. A state feature function may be $s_k(y_i, x, i) = 1$ if the word at the position i is *table* and the POS tag at the position i is NN, otherwise $s_k(y_i, x, i) = 0$.

The probability of a label sequence y given an observation sequence x then can be defined as follows:

$$p(y|x, \lambda, \mu) = \frac{1}{Z(x)} exp(\sum_j \lambda_j t_j(y_{i-1}, y_i, x, i) + \sum_k \mu_k s_k(y_i, x, i)) \tag{2.4}$$

where $Z(x)$ is a normalization factor,

$$Z(x) = \sum_y exp(\sum_j \lambda_j t_j(y_{i-1}, y_i, x, i) + \sum_k \mu_k s_k(y_i, x, i)). \tag{2.5}$$

Training CRFs is commonly performed by maximizing the likelihood function with respect to the training data using advanced convex optimization techniques like L-BFGS [4]. And inference in CRFs, i.e., searching the most likely output label sequence of an input observation sequence, can be done by using Viterbi algorithm [11].

In the RRE task, we choose CRFs as the learning method. There are some reasons why we choose CRFs. The first reason comes from the nature of the RRE task. The RRE task can be considered as a sequence learning problem, and CRFs is an efficient and powerful framework for sequence learning tasks. The second reason comes from the advantages of CRFs. CRFs is a discriminative method, it has all the advantages of Maximum entropy Markov models (MEMMs) [25] but does not suffer from the label bias problem [23]. The last reason is that CRFs has been applied successfully to many NLP tasks such as POS tagging, chunking, named entity recognition, syntax parsing, information retrieval, information extraction, etc [22, 23, 32, 37].

2.3 Corpus and Evaluation Methods

This section presents our corpus for the RRE task (Japanese National Pension Law corpus) and evaluation methods.

2.3.1 Corpus

The Japanese National Pension Law (JNPL) corpus includes 764 annotated Japanese law sentences[4]. Some statistics on the JNPL corpus are shown in Table 2.1. We have some remarks to make here. First, about 98.5% of sentences belong to the implication type, and only 1.5% of sentences belong to the equivalence type. Second, about 83.5% of topic parts are T_2, 15.2% of topic parts are T_3, and only 1.3% of topic parts are T_1. Finally, four main types of parts, antecedent parts (A), consequent parts (C), topic parts in case 2 (T_2), and topic parts in case 3 (T_3) make up more than 98.3% of all types.

Table 2.1: Statistics on the Japanese National Pension Law corpus

Sentence Type	Number	Part Type	Number
Equivalence	11	EL	11
		ER	11
Implication	753	**A**	**429**
		C	**745**
		T_1	9
		$\mathbf{T}_2$	**562**
		$\mathbf{T}_3$	**102**

In Legal Engineering, we mainly study the National Pension Law of Japan. But why do we choose National Pension Law for Legal Engineering? Our purpose is to study methodologies to design an information system based on laws, and methodologies to verify the consistency of laws, which are the base of that information system. National Pension Law is a procedural law, which provides administrative procedures of the national pension system. It can be regarded as the specification of an information system. Therefore, National Pension Law is suitable for studying methodologies in Legal Engineering.

2.3.2 Evaluation Methods

In all experiments on the RRE task, we divided the corpus into 10 sets and performed 10-fold cross-validation tests. Results were evaluated using precision, recall, and $F_{\beta=1}$ scores as follows:

[4]The corpus consists of mainly the first sentence of each article.

$$precision = \frac{\#\text{correct parts}}{\#\text{predicted parts}}, recall = \frac{\#\text{correct parts}}{\#\text{gold parts}} \tag{2.6}$$

$$F_{\beta=1} = \frac{2 * precision * recall}{precision + recall} \tag{2.7}$$

Note that results were evaluated based on logical parts, not based on labels of sequences. From the predicted label sequence, we extracted predicted logical parts, and compared predicted logical parts with annotated logical parts (or gold logical parts). A logical part is recognized correctly if and only if it has correct start element, correct end element, and correct part category (kind of logical part).

Figure 2.4 shows an example[5] of how to evaluate the RRE task. In this example, from the predicted label sequence, we can extract two predicted logical parts: $A(1,5)$ and $C(6,9)$. Suppose that gold logical parts are $T_1(1,2)$, $A(3,5)$, and $C(6,9)$, then *precision*, *recall*, and F_1 scores can be calculated as follows:

$$precision = \frac{\#\text{correct parts}}{\#\text{predicted parts}} = \frac{1}{2} = 0.50, recall = \frac{\#\text{correct parts}}{\#\text{gold parts}} = \frac{1}{3} = 0.33, \tag{2.8}$$

$$F_1 = \frac{2 * precision * recall}{precision + recall} = \frac{2 * 0.50 * 0.33}{0.50 + 0.33} = 0.40 \tag{2.9}$$

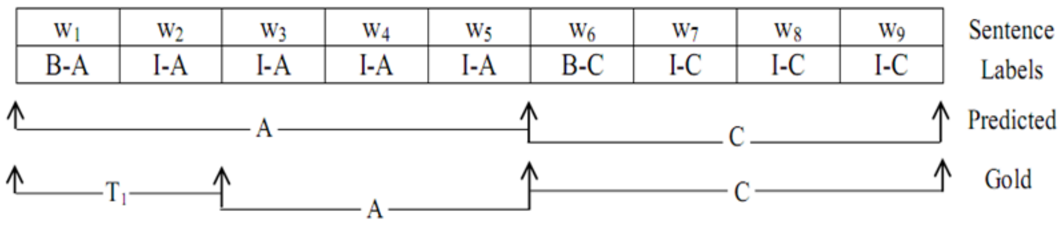


Figure 2.4: An example of evaluation method for the RRE task.

[5] A means antecedent part, C means consequent part, and T_1 means topic part in case 1.

Chapter 3

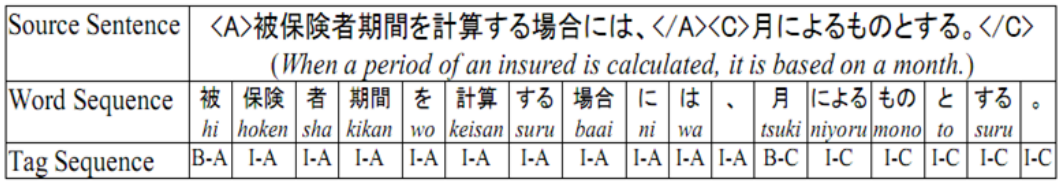
RRE Based on Words

In this chapter, we present a word-based model for the RRE task, in which each word is considered as an element in the sequence model (Section 3.1). We describe an investigation on linguistic features for the RRE task (Section 3.2). We present a study on RRE using head words and functional words (Section 3.3). We also present an investigation on contributions of words to RRE (Section 3.4). Finally, some conclusions are given (Section 3.5).

3.1 Word-based Model

In the word-based model, we consider a law sentence as a sequence of words. The recognition task now becomes a sequence learning task in which we assign a label sequence to an input word sequence.

Figure 3.1 shows an example of a law sentence in the word-based model. First, we segment the law sentence into a sequence of words. Then, we label the word sequence using IOB notation.



Source Sentence	<A>被保険者期間を計算する場合には、</A><C>月によるものとする。</C> (*When a period of an insured is calculated, it is based on a month.*)																
Word Sequence	被 *hi*	保険 *hoken*	者 *sha*	期間 *kikan*	を *wo*	計算 *keisan*	する *suru*	場合 *baai*	に *ni*	は *wa*	、	月 *tsuki*	による *niyoru*	もの *mono*	と *to*	する *suru*	。
Tag Sequence	B-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	B-C	I-C	I-C	I-C	I-C	I-C

Figure 3.1: A law sentence in the word-based model.

3.2 Feature Investigation

This section presents our experiments to evaluate the effects of features on the RRE task. All experiments were conducted using CRFs [23]. We used the implementation of Taku Kudo [21].

3.2.1 Feature Design

We designed five sets of features using the CaboCha[1] tool [20]. Each of these feature sets contains one kind of feature. With each kind of feature f, we obtained the following features in a window size 2:

- Uni-gram features (5): $f[-2]$, $f[-1]$, $f[0]$, $f[1]$, $f[2]$
- Bi-gram features (4): $f[-2]f[-1]$, $f[-1]f[0]$, $f[0]f[1]$, $f[1]f[2]$
- Tri-gram features (3): $f[-2]f[-1]f[0]$, $f[-1]f[0]f[1]$, $f[0]f[1]f[2]$.

For example, if f is word feature then $f[0]$ is the current word, $f[-1]$ is the preceding word, and $f[-1]f[0]$ is their co-occurrence. More details on feature sets are shown in Table 3.1.

Table 3.1: Feature design

Feature Set	**Kinds of Features**	**Window Size**	**#Features**
Set 1	Word	2	12
Set 2	POS tag	2	12
Set 3	Katakana and Stem of word	2	24
Set 4	Bunsetsu tag	2	12
Set 5	Named Entity tag	2	12

3.2.2 Baseline

We considered the model using only word features as the baseline model. The results of the baseline model are shown in Table 3.2. They are quite good, especially on four main kinds of parts, C, A, T_2, and T_3. This means that word features are very important to the RRE task.

[1]A Japanese morphological and dependency structure analyzer.

Table 3.2: Results of the baseline model

Tag	**Precision(%)**	**Recall(%)**	**F$_{\beta=1}$(%)**
C	90.25	91.95	91.09
EL	0.00	0.00	0.00
ER	0.00	0.00	0.00
A	89.29	85.55	87.38
T_1	100.00	22.22	36.36
T_2	85.02	89.86	87.37
T_3	60.00	38.24	46.71
Overall	**87.27**	**85.50**	**86.38**

3.2.3 Experiments on Feature Sets

To investigate the effects of features on the task, we conducted experiments on four other feature sets combined with the word features. The experimental results are shown in Table 3.3. Model 1 using only word features is the baseline model. Only Model 3 with *word* and *pos* features led to an improvement of 0.28% compared with the baseline model. Three other models yielded worse results. We can see that features other than word and pos features were not effective for our recognition task. In these experiments, Bunsetsu information was used as features of elements in sequences. In the next sections and the next chapters, we will present some other ways to use Bunsetsu information more efficiently.

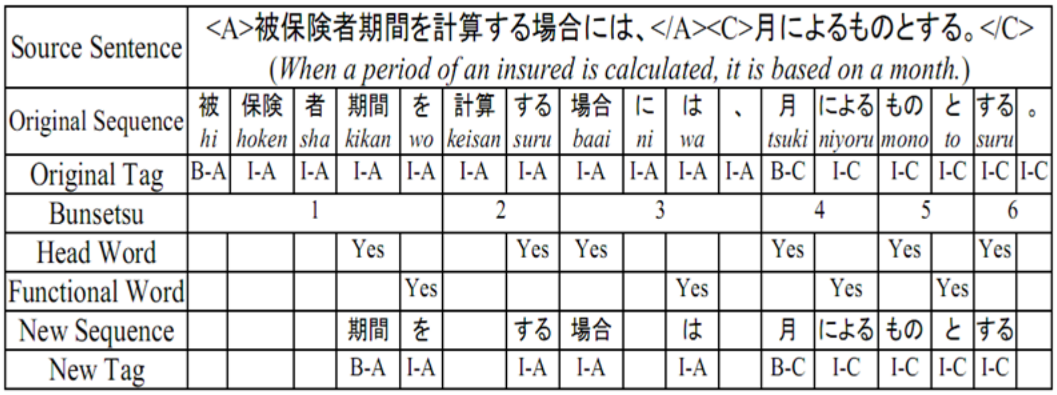
Table 3.3: Experiments with feature sets

Model	**Feature Sets**	**Precision(%)**	**Recall(%)**	**F$_{\beta=1}$(%)**
Model1	Word	87.27	85.50	86.38
Model2	Word + Katakana, Stem	87.02	85.39	86.20(-0.18)
Model3	Word + POS	**87.68**	**85.66**	**86.66(+0.28)**
Model4	Word + Bunsetsu	86.15	84.86	85.50(-0.88)
Model5	Word + NE	87.22	85.45	86.32(-0.06)

3.3 RRE Using Head Words and Functional Words

In Japanese, a sentence can be divided into some chunks called Bunsetsus. Each Bunsetsu includes one or more content words (noun, verb, adjective, etc) and may include some function words (case-marker, punctuation, etc). The head word of a Bunsetsu is the rightmost content word, and the functional word is the rightmost function word, except for punctuation [28]. The head word contributes the central meaning and the functional word plays a grammatical role. The couple of these two words is the core of a Bunsetsu, and other words play less important roles.

Our idea is to: first, reduce an original sentence to a reduced sentence which contains only head words and functional words; then, perform the recognition task on the new reduced sentence. From the recognition results of the reduced sentence, we can deduce the results of the original sentence. We illustrate this process in Figure 3.2. First, we split the original sentence into a sequence of Bunsetsu. Then, we mark words which are the head word or the functional word of a Bunsetsu. Only words with a mark remain in the reduced sentence.



Source Sentence	<A>被保険者期間を計算する場合には、</A><C>月によるものとする。</C> *(When a period of an insured is calculated, it is based on a month.)*																
Original Sequence	被 *hi*	保険 *hoken*	者 *sha*	期間 *kikan*	を *wo*	計算 *keisan*	する *suru*	場合 *baai*	に *ni*	は *wa*	、	月 *tsuki*	による *niyoru*	もの *mono*	と *to*	する *suru*	。
Original Tag	B-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	B-C	I-C	I-C	I-C	I-C	I-C
Bunsetsu	1					2		3				4		5		6	
Head Word				Yes			Yes	Yes				Yes		Yes		Yes	
Functional Word					Yes					Yes			Yes		Yes		
New Sequence				期間	を		する	場合		は		月	による	もの	と	する	
New Tag				B-A	I-A		I-A	I-A		I-A		B-C	I-C	I-C	I-C	I-C	

Figure 3.2: Sentence reduction.

Experimental results on the new reduced sentences are shown in Table 3.4. In the HFW (Head-Functional-Word) model, we only use head and functional words, while in the HFWP (Head-Functional-Word-POS) model, we use head and functional words and their POS tags. In both models, we retain punctuation marks, which are important signals in a sentence.

Table 3.4: Experiments on reduced sentences

Model	**Sentence**	**Feature**	**Precision(%)**	**Recall(%)**	**F$_{\beta=1}$(%)**
Baseline	Original	Word	87.27	85.50	86.38
HFW	Reduction	Word	**88.09**	**86.30**	**87.19(+0.81)**
HFWP	Reduction	Word + POS	87.74	86.52	87.12(+0.74)

Experimental results show that using reduced sentences gives better results than using full sentences. This demonstrates the importance of head and functional words in the RRE task. The HFW model improves by 0.81% in the $F_{\beta=1}$ score (5.9% in error rate) compared with the baseline model.

Reduced sentences only contains important words of the original sentences (head words, functional words, and punctuation marks). These words are significant to convey the main meaning of the sentence. Hence, the model based on reduced sentences only process on a subset of words compared with the word-based model. It can disregard other unimportant words. It is the reason why the model using head words and functional words is better than the word-based model.

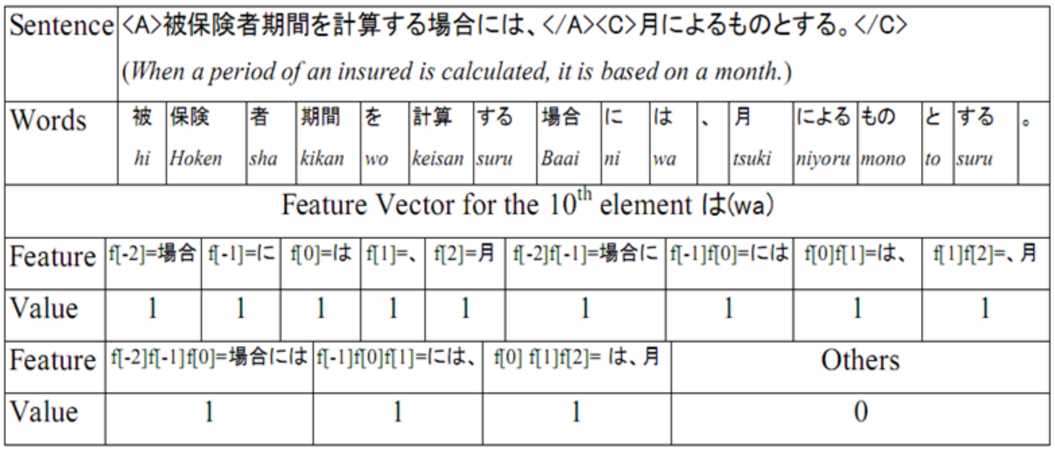
3.4 Exploring Contributions of Words to RRE

In the previous sections, we have applied statistical machine learning methods to the RRE task. Experimental results show that word features are very important. However, applying machine learning methods to NLP tasks is considered as a black-box process, in which it is too difficult to understand the behavior of models. In the RRE task, we do not know the contributions of each word, which words are good, and which words are bad. An interesting question is that *whether machine learning methods use the same words as human do in recognizing logical parts of law sentences.* This section presents an investigation on the contributions of words to the RRE task, and finds an answer to this question.

3.4.1 Method

In sequence learning models (such as CRFs model), a feature vector for an element is a vector in which each factor usually is an indicator function. For example, indicator function $f[0] = play$ will returns 1 if the current word is *play*, otherwise it returns 0. Indicator function $f[0]f[1] = play\ tennis$ will returns 1 if the current word is *play* and the next word is *tennis*, otherwise it returns 0.

Figure 3.3 shows an example of a feature vector of an element in sequence learning models. This vector includes features extracted in a window size 2. In this vector, most of feature values are zero, only features that map with the context of the element have non-zero value (1 in the case of indicator functions).



Sentence	<A>被保険者期間を計算する場合には、</A><C>月によるものとする。</C> *(When a period of an insured is calculated, it is based on a month.)*

Words	被	保険	者	期間	を	計算	する	場合	に	は	、	月	による	もの	と	する	。
	hi	*Hoken*	*sha*	*kikan*	*wo*	*keisan*	*suru*	*Baai*	*ni*	*wa*		*tsuki*	*niyoru*	*mono*	*to*	*suru*	

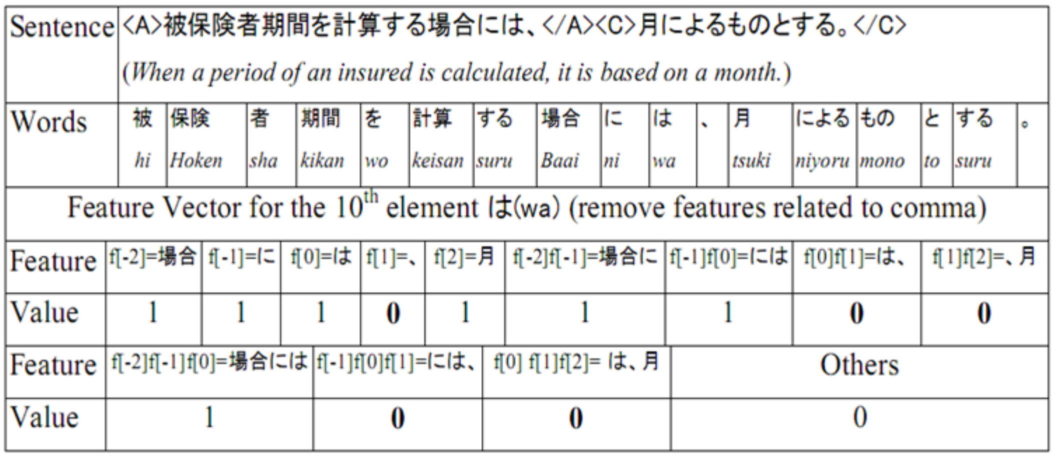
Feature Vector for the 10th element は(wa)

Feature	f[-2]=場合	f[-1]=に	f[0]=は	f[1]=、	f[2]=月	f[-2]f[-1]=場合に	f[-1]f[0]=には	f[0]f[1]=は、	f[1]f[2]=、月
Value	1	1	1	1	1	1	1	1	1

Feature	f[-2]f[-1]f[0]=場合には	f[-1]f[0]f[1]=には、	f[0] f[1]f[2]= は、月	Others
Value	1	1	1	0

Figure 3.3: A feature vector in sequence learning models.

In our method, to investigate contributions of a word w, we remove all features related to w, and compare the performance of the system before and after removing features. A decrease in the performance means that word w is important to the task. Figure 3.4 illustrates the feature vector in Figure 3.3 after removing all the features related to comma. In this vector, all values are the same with the previous vector, except for values of five features related to comma (they are changed from 1 to 0).



Sentence	<A>被保険者期間を計算する場合には、</A><C>月によるものとする。</C>
	(When a period of an insured is calculated, it is based on a month.)

Words	被	保険	者	期間	を	計算	する	場合	に	は	、	月	による	もの	と	する	。
	hi	*Hoken*	*sha*	*kikan*	*wo*	*keisan*	*suru*	*Baai*	*ni*	*wa*		*tsuki*	*niyoru*	*mono*	*to*	*suru*	

Feature Vector for the 10th element は(wa) (remove features related to comma)

Feature	f[-2]=場合	f[-1]=に	f[0]=は	f[1]=、	f[2]=月	f[-2]f[-1]=場合に	f[-1]f[0]=には	f[0]f[1]=は、	f[1]f[2]=、月
Value	1	1	1	**0**	1	1	1	**0**	**0**

Feature	f[-2]f[-1]f[0]=場合には	f[-1]f[0]f[1]=には、	f[0] f[1]f[2]= は、月	Others
Value	1	**0**	**0**	0

Figure 3.4: A feature vector after removing features related to comma.

Let f_1 be the $F_{\beta=1}$ score of the system when we use all the features, and $f_1 w$ be the $F_{\beta=1}$ score of the system when we remove all the features related to a word w. Errors in two cases are computed as follows:

$$error = 1 - f_1, \tag{3.1}$$

$$error_w = 1 - f_1 w. \tag{3.2}$$

We define an *errorRate* score of a word w, the percentage of the change in the error, as follows:

$$errorRate_w = (error_w - error)/error. \tag{3.3}$$

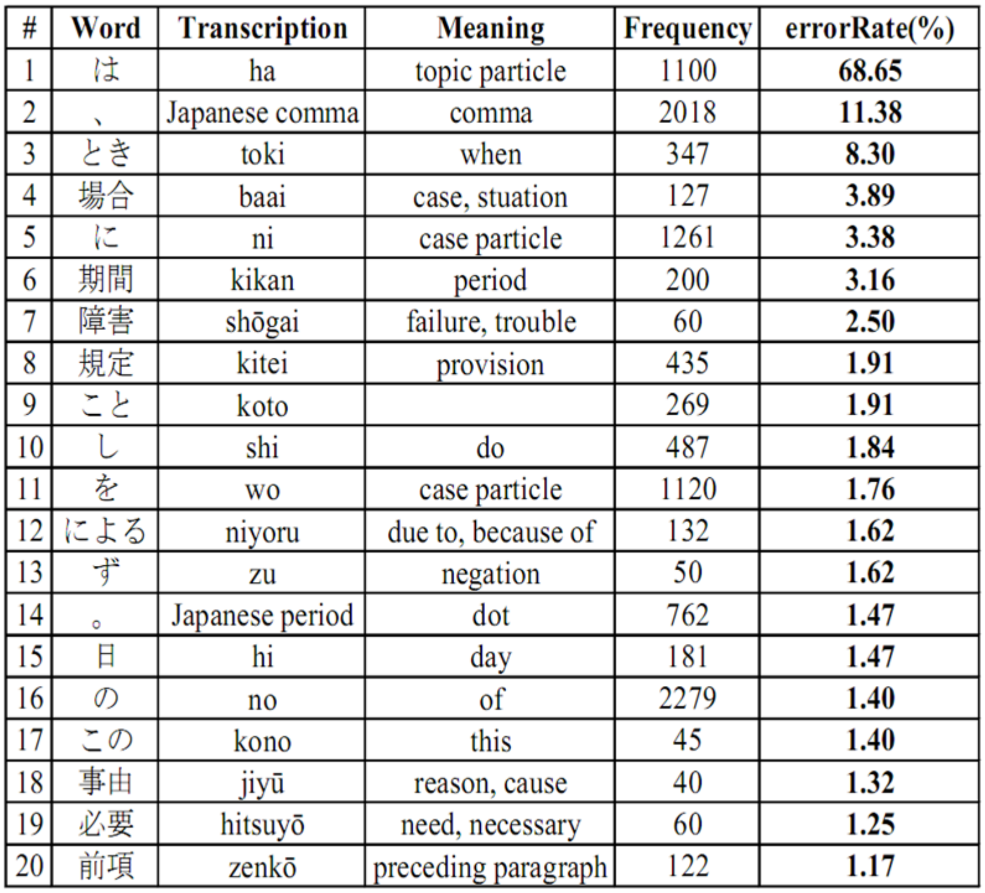
We use the *errorRate* score of a word w to evaluate the importance of w. The lager *errorRate* is, the more important w is. This is reasonable because the performance of the system decreases when we remove all the features related to w.

3.4.2 Experimental Results

We conducted experiments with top 100 most frequency words[2] of the JNPL corpus. Figure 3.5 and Figure 3.6 show experimental results of top 20 highest *errorRate* words. Most of these words have strong relations to the logical structure of law sentences.

In many cases, the word *wa* separates a *topic part* from other parts. Statistics on our JNPL corpus show that, among 673 topic parts (including T_1, T_2, and T_3), 655 cases

[2]Top 100 most frequency words are listed in Figure 3.8 at the end of this chapter.



#	Word	Transcription	Meaning	Frequency	errorRate(%)
1	は	ha	topic particle	1100	**68.65**
2	、	Japanese comma	comma	2018	**11.38**
3	とき	toki	when	347	**8.30**
4	場合	baai	case, stuation	127	**3.89**
5	に	ni	case particle	1261	**3.38**
6	期間	kikan	period	200	**3.16**
7	障害	shōgai	failure, trouble	60	**2.50**
8	規定	kitei	provision	435	**1.91**
9	こと	koto		269	**1.91**
10	し	shi	do	487	**1.84**
11	を	wo	case particle	1120	**1.76**
12	による	niyoru	due to, because of	132	**1.62**
13	ず	zu	negation	50	**1.62**
14	。	Japanese period	dot	762	**1.47**
15	日	hi	day	181	**1.47**
16	の	no	of	2279	**1.40**
17	この	kono	this	45	**1.40**
18	事由	jiyū	reason, cause	40	**1.32**
19	必要	hitsuyō	need, necessary	60	**1.25**
20	前項	zenkō	preceding paragraph	122	**1.17**

Figure 3.5: Experimental results.

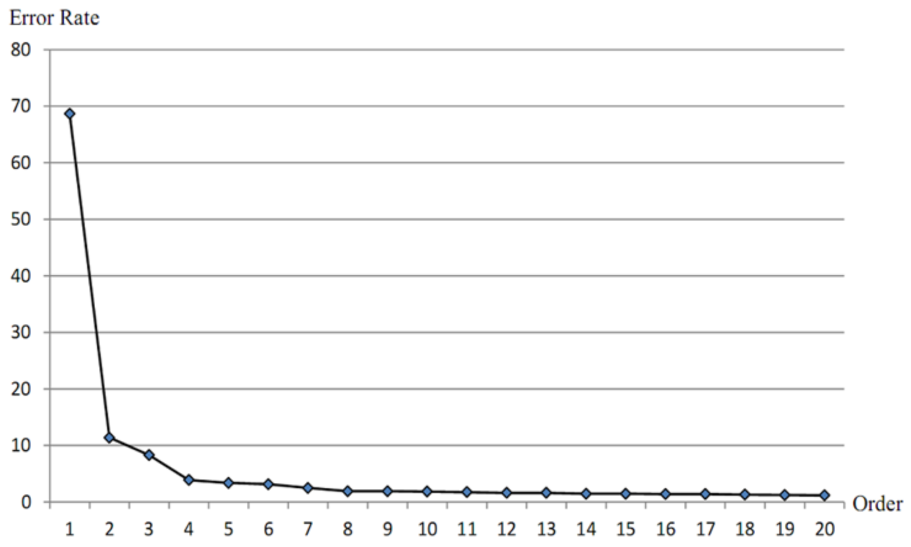


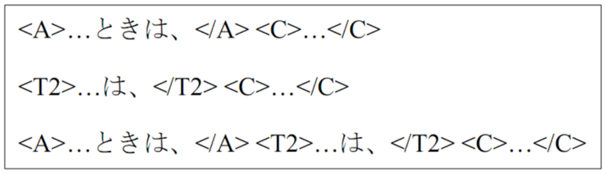
Figure 3.6: Error rates by orders of words.

(more than 97%) end with the word *wa* followed by a *comma*. A logical part usually ends with a punctuation mark (comma or dot). Among 1869 logical parts in the JNPL corpus, 1070 parts (more than 57%) end with a *comma* and 745 parts (about 40%) end with a *dot*. Hence, about 97% logical parts end with a punctuation mark.

Words *toki* (when) and *baai* (case, situation) are clear signals of an *antecedent part*. In the JNPL corpus, the word *toki* appears 347 times, in which it belongs to an antecedent part 343 times (about 99%). Only 4 times it belongs to a consequent part (about 1%). The word *baai* appears 127 times in the corpus. It belongs to an antecedent part 115 times (about 90%), a consequent part 4 times (about 3%), and a topic part 8 times (about 7%). Words *niyoru* (due to) and *jiyū* (reason, cause) realate to if-then strutures. Words *kikan* (period), *shōgai* (failure, trouble), *kitei* (provision), *hitsuyō* (need, necessary), and *zenkō* (preceding paragraph) are characteristics of legal texts.

We can see that, the top three words *wa* (68.65%), *comma* (11.38%), and *toki* (8.30%) are very important to the RRE task. They are significant signals for recognizing logical structures of law sentences.

Figure 3.7 presents some common templates (built from three words *wa*, *comma*, and *toki*) of law sentences[3]. In the first template, a law sentence consists of an *antecedent part* and a *consequent part*. In the second template, a law sentence consists of a *topic part* and a *consequent part*. In the last template, a law consists of an *antecedent part*, a *topic part*, and a *consequent part*. In all the cases, antecedent parts end with the phrase of words *toki*,*wa*, and *comma*, and topic parts end with the word *wa* followed by a *comma*.


<A>…ときは、</A> <C>…</C>

<T2>…は、</T2> <C>…</C>

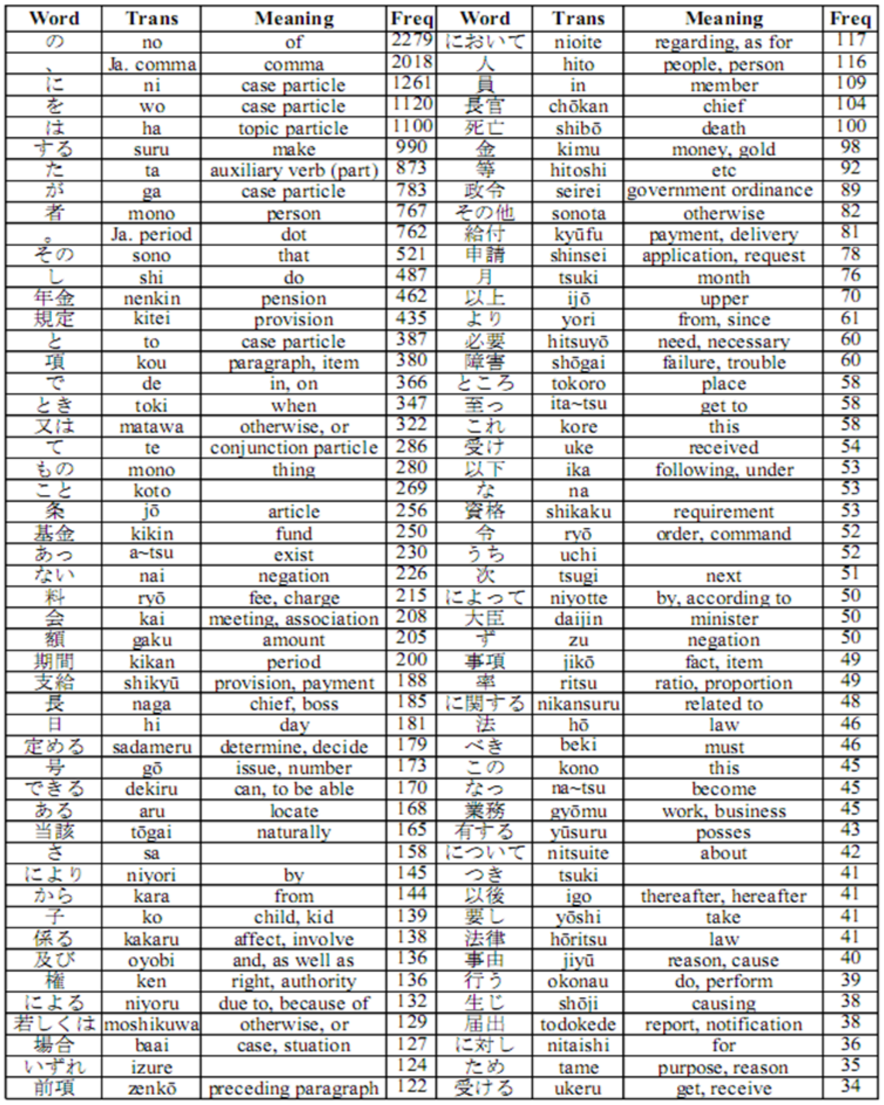
<A>…ときは、</A> <T2>…は、</T2> <C>…</C>

Figure 3.7: Some common templates of law sentences.

3.5 Conclusions

In this chapter, we presented our word-based model for the RRE task, in which we consider a law sentence as a sequence of words. We investigated various kinds of features including words, katakana and stem of words, part-of-speech (POS) tags, NE tags, and Bunsetsu tags. We found that only word and POS tag features are effective to the RRE task. We showed that using only functional words and head words is better than using all words. We also investigated contributions of words to the RRE task. We found that words that are important to human in recognizing logical structures of law sentences are also important to our statistical machine learning models. This is meaningful in the linguistic aspect, because statistical machine learning models are usually considered as black boxes.

[3]A means an *antecedent part*, C means a *consequent part*, and $T2$ means a *topic part* in case 2.



Word	Trans	Meaning	Freq	Word	Trans	Meaning	Freq
の	no	of	2279	において	nioite	regarding, as for	117
、	Ja. comma	comma	2018	人	hito	people, person	116
に	ni	case particle	1261	員	in	member	109
を	wo	case particle	1120	長官	chōkan	chief	104
は	ha	topic particle	1100	死亡	shibō	death	100
する	suru	make	990	金	kimu	money, gold	98
た	ta	auxiliary verb (part)	873	等	hitoshi	etc	92
が	ga	case particle	783	政令	seirei	government ordinance	89
者	mono	person	767	その他	sonota	otherwise	82
。	Ja. period	dot	762	給付	kyūfu	payment, delivery	81
その	sono	that	521	申請	shinsei	application, request	78
し	shi	do	487	月	tsuki	month	76
年金	nenkin	pension	462	以上	ijō	upper	70
規定	kitei	provision	435	より	yori	from, since	61
と	to	case particle	387	必要	hitsuyō	need, necessary	60
項	kou	paragraph, item	380	障害	shōgai	failure, trouble	60
で	de	in, on	366	ところ	tokoro	place	58
とき	toki	when	347	至っ	ita~tsu	get to	58
又は	matawa	otherwise, or	322	これ	kore	this	58
て	te	conjunction particle	286	受け	uke	received	54
もの	mono	thing	280	以下	ika	following, under	53
こと	koto		269	な	na		53
条	jō	article	256	資格	shikaku	requirement	53
基金	kikin	fund	250	令	ryō	order, command	52
あっ	a~tsu	exist	230	うち	uchi		52
ない	nai	negation	226	次	tsugi	next	51
料	ryō	fee, charge	215	によって	niyotte	by, according to	50
会	kai	meeting, association	208	大臣	daijin	minister	50
額	gaku	amount	205	ず	zu	negation	50
期間	kikan	period	200	事項	jikō	fact, item	49
支給	shikyū	provision, payment	188	率	ritsu	ratio, proportion	49
長	naga	chief, boss	185	に関する	nikansuru	related to	48
日	hi	day	181	法	hō	law	46
定める	sadameru	determine, decide	179	べき	beki	must	46
号	gō	issue, number	173	この	kono	this	45
できる	dekiru	can, to be able	170	なっ	na~tsu	become	45
ある	aru	locate	168	業務	gyōmu	work, business	45
当該	tōgai	naturally	165	有する	yūsuru	posses	43
さ	sa		158	について	nitsuite	about	42
により	niyori	by	145	つき	tsuki		41
から	kara	from	144	以後	igo	thereafter, hereafter	41
子	ko	child, kid	139	要し	yōshi	take	41
係る	kakaru	affect, involve	138	法律	hōritsu	law	41
及び	oyobi	and, as well as	136	事由	jiyū	reason, cause	40
権	ken	right, authority	136	行う	okonau	do, perform	39
による	niyoru	due to, because of	132	生じ	shōji	causing	38
若しくは	moshikuwa	otherwise, or	129	届出	todokede	report, notification	38
場合	baai	case, stuation	127	に対し	nitaishi	for	36
いずれ	izure		124	ため	tame	purpose, reason	35
前項	zenkō	preceding paragraph	122	受ける	ukeru	get, receive	34

Figure 3.8: Top 100 most frequency words in the JNPL corpus.

In this chapter, Bunsetsu information was used as features of the word-based model. In the next chapter, we will present a model that uses Bunsetsu information in a more effective way.

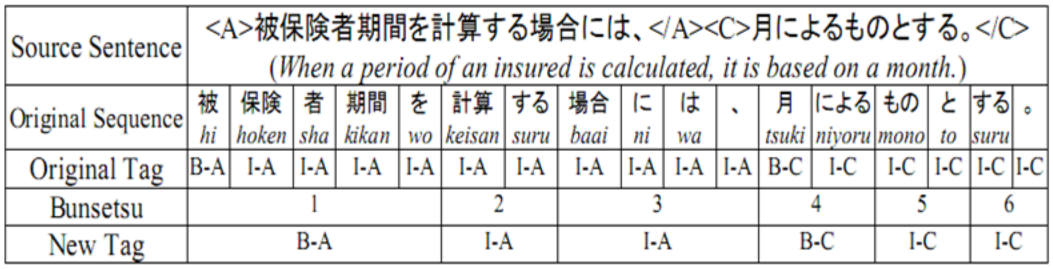
Chapter 4

RRE Based on Bunsetsu

In this chapter, we first present a Bunsetsu-based model, in which each Bunsetsu is considered as an element in the sequence model (Section 4.1). We then present some experimental results on the Bunsetsu-based model (Section 4.2), and some experiments with different tag settings on this model (Section 4.3). We next describe a reranking model for the RRE task using a variant of the perceptron algorithm (Section 4.4). Finally, we give some conclusions (Section 4.5).

4.1 Bunsetsu-based Model

In Chapter 3, we model the RRE task as a sequence labeling problem in which elements of sequences are words. Because a sentence may contain many words, the length of a sequence becomes large. In the Bunsetsu-based model, instead of considering words as elements, we consider each Bunsetsu as an element. This can be done because no Bunsetsu can belong to two different parts in this task. By doing this, we can reduce the length of sequences significantly. The recognition task now becomes a sequence learning task in which we assign a label sequence to an input Bunsetsu sequence. An example of a law sentence in the Bunsetsu-based model is illustrated in Figure 4.1.



Source Sentence	<A>被保険者期間を計算する場合には、</A><C>月によるものとする。</C> *(When a period of an insured is calculated, it is based on a month.)*																
Original Sequence	被 *hi*	保険 *hoken*	者 *sha*	期間 *kikan*	を *wo*	計算 *keisan*	する *suru*	場合 *baai*	に *ni*	は *wa*	、	月 *tsuki*	による *niyoru*	もの *mono*	と *to*	する *suru*	。
Original Tag	B-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	I-A	B-C	I-C	I-C	I-C	I-C	I-C
Bunsetsu	1					2		3				4		5		6	
New Tag	B-A					I-A		I-A				B-C		I-C		I-C	

Figure 4.1: A law sentence in the Bunsetsu-based model.

In this example, the length of the sequence is reduced from 17 (words) to 6 (Bunsetsus). On average, in the Japanese National Pension Law corpus, the length of a sequence with the old setting (words) is **47.3**, while only **17.6** with the new setting (Bunsetsus).

4.2 Experiments on Bunsetsu-based Model

We use features about head words, functional words, punctuations, and the co-occurrence of head words and functional words in a window size 1. A window size 1 in this model will cover three Bunsetsu. So, it is much longer than a window size 2 (which covers five words) in a model based on words. This is the reason why a window size 1 is sufficient in this model. Experimental results with the new setting are shown in Table 4.1, in which the BC model (Based-on-Chunks) is the model with the new setting. The results show that modeling based on Bunsetsu, an important unit in Japanese sentences, is suitable for the RRE task.

Table 4.1: Experiments on the Bunsetsu-based model

Model	**Precision(%)**	**Recall(%)**	$\mathbf{F}_{\beta=1}$**(%)**
Baseline	87.27	85.50	86.38
HFW	88.09	86.30	87.19(+0.81)
BC	**88.75**	**86.52**	**87.62(+1.24)**

A detailed comparison between the BC model and the baseline model is shown in Table 4.2. The BC model improves by 1.24% in the $F_{\beta=1}$ score (9.1% in error rate), compared with the baseline model.

There are two reasons may explain why the Bunsetsu-based model is better than the word-based model. The first reason is that Bunsetsus are basic units in analyzing Japanese(in fact, dependency parsing of Japanese based on Bunsetsus, not words). Bunsetsus convey the meaning of a sentence better than words. In the Bunsetsu-based model, we only use head words and functional words to represent a Bunsetsu. Hence, the Bunsetsu-based model also take advantages of the model using head words and functional words. The second reason is that the Bunsetsu-based model reduces the length of sequences significantly compared with the word-based models. It helps the Bunsetsu-based model so much in the learning process.

4.3 Experiments with Different Tag Settings

4.3.1 Four Kinds of Tag Settings

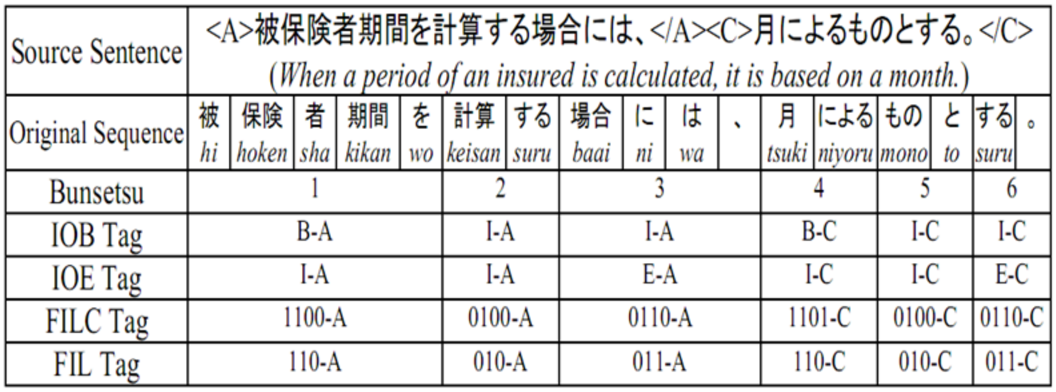
We describe briefly four kinds of tag settings which are normally used in sequence labeling problems [26].

Table 4.2: Comparison between the baseline model and the BC model

Tag	Baseline model			BC model		
	Precision(%)	Recall(%)	$F_{\beta=1}$	Precision(%)	Recall(%)	$F_{\beta=1}$
C	90.25	91.95	91.09	91.39	92.62	92.00(+0.91)
EL	0.00	0.00	0.00	0.00	0.00	0.00
ER	0.00	0.00	0.00	0.00	0.00	0.00
A	89.29	85.55	87.38	89.43	84.85	87.08(-0.30)
T_1	100.00	22.22	36.36	100.00	22.22	36.36
T_2	85.02	89.86	87.37	85.62	91.10	88.28(+0.91)
T_3	60.00	38.24	46.71	81.67	48.04	60.49(+13.78)
Overall	87.27	85.50	86.38	88.75	86.52	87.62(+1.24)

1. IOB: The first element of a part is tagged by B, the other elements of a part are tagged by I, and an element not included in any part is tagged by O.
2. IOE: The last element of a part is tagged by E, the other elements of a part are tagged by I, and an element not included in any part is tagged by O.
3. FILC: Four boolean categories are used: whether the element is the first in a part (F), inside a part (I), last in a part (L), or first in a consecutive part (C).
4. FIL: This is similar to FILC, except that we only use three boolean categories, F, I, and L.

An example that illustrates these tag settings for the RRE task is shown in Figure 4.2.



Source Sentence	<A>被保険者期間を計算する場合には、</A><C>月によるものとする。</C> (*When a period of an insured is calculated, it is based on a month.*)																
Original Sequence	被 *hi*	保険 *hoken*	者 *sha*	期間 *kikan*	を *wo*	計算 *keisan*	する *suru*	場合 *baai*	に *ni*	は *wa*	、	月 *tsuki*	による *niyoru*	もの *mono*	と *to*	する *suru*	。
Bunsetsu	1					2		3				4		5		6	
IOB Tag	B-A					I-A		I-A				B-C		I-C		I-C	
IOE Tag	I-A					I-A		E-A				I-C		I-C		E-C	
FILC Tag	1100-A					0100-A		0110-A				1101-C		0100-C		0110-C	
FIL Tag	110-A					010-A		011-A				110-C		010-C		011-C	

Figure 4.2: Four kinds of tag settings.

4.3.2 Experiments

To investigate the RRE task with different tag settings, we conducted experiments using the BC model with all four kinds of tag settings. In all experiments (see Table 4.3), we use the same feature set (head words, functional words, punctuations, and the co-occurrence of head words and functional words in a window size 1).

The best tag setting is IOE (it improves by 1.80% in the $F_{\beta=1}$ score, 13.2% in error rate compared with the baseline model). In the IOE setting, the model determines the last element of a part, while in the IOB setting, the model determines the first one. There are two reasons which may explain why the IOE setting is suitable for the RRE task. First, in Japanese, important words usually occur at the end of a phrase, and important Bunsetsu usually occur at the end of a sentence. A Bunsetsu always depends on another Bunsetsu which stands to its right. Second, in the RRE task, a part tends to finish at a punctuation mark (comma or period). So a chunk containing a comma or period has a high probability of being the last element of a part.

Table 4.3: Experiments with four kinds of tag settings

Model	**Element**	**Setting**	**Precision(%)**	**Recall(%)**	**$\mathbf{F}_{\beta=1}$(%)**
Baseline	Word	IOB	87.27	85.50	86.38
BC-IOB	Bunsetsu	IOB	88.75	86.52	87.62(+1.24)
BC-IOE	Bunsetsu	IOE	**89.35**	**87.05**	**88.18(+1.80)**
BC-FILC	Bunsetsu	FILC	88.75	86.09	87.40(+1.02)
BC-FIL	Bunsetsu	FIL	88.87	86.30	87.57(+1.19)

4.4 A Reranking Model for RRE

4.4.1 Why Reranking?

Discriminative reranking with linear models has been used successfully in some NLP tasks such as POS tagging, chunking, and statistical parsing [8]. The advantage of the reranking method is that it can exploit the output of a base model to learn. Based on the output of a base model, we can extract long-distance non-local features, which are impossible in sequence learning Markov models such as the Hidden Markov model, the Maximum entropy Markov model, and CRFs. In the RRE task, we use the reranking method to utilize features (probability, tag sequence, part sequence, etc) extracted from the output of a base model which is learned using CRFs.

4.4.2 Discriminative Reranking with Linear Models

In this method, first, a set of candidates is generated using a component GEN. GEN can be any model for the task. For example, in the POS tagging problem, GEN may be a model that generates all possible POS tags for a word based on a dictionary. Then, candidates are reranked using a linear score function:

$$score(y) = \Phi(y) \cdot W \tag{4.1}$$

where y is a candidate, $\Phi(y)$ is the feature vector of candidate y, and W is a parameter vector. The output of the reranking model will be the candidate with the highest score:

$$F(x) = argmax_{y \in GEN(x)} score(y) = argmax_{y \in GEN(x)} \Phi(y) \cdot W. \tag{4.2}$$

In this method, we must consider three questions:

1. How to choose the GEN component?
2. How to represent the feature vector Φ?
3. How to learn the parameter vector W?

In the next sub-sections, we will discuss how to represent a feature vector in the RRE task. We then present some variants of the perceptron algorithm, which will be used to learn the parameter vector. The choice of GEN component will be presented in the experiment sub-section.

4.4.3 Feature Representation

For a candidate, we extract a tag sequence and a part sequence. The tag sequence is the output of the candidate after removing the second tag if there are two adjacent same tags. In both sequences, we insert two special symbols, START at the beginning, and END at the end. For example:

- Candidate: I-A I-A I-A E-A I-T_2 I-T_2 I-T_2 E-T_2 I-C I-C I-C E-C
- Tag sequence: START I-A E-A I-T_2 E-T_2 I-C E-C END
- Part sequence: START A T_2 C END

For a candidate, we extract the following features (each of them corresponds to an element in a feature vector):

1. **Prob**: probability of the candidate output by GEN.
2. **Unigram**: For example: the number of times that the tag I-A (part A) appears in the tag sequence (part sequence).
3. **Bigram**: For example: the number of times that two tags, E-A, END, (two parts, A, END) co-appear in the tag sequence (part sequence).
4. **Trigram**: For example: the number of times that three tags, I-A, E-A, END, (three parts, A,C,END) co-appear in the tag sequence (part sequence).
5. **Num-of-parts**: the number of logical parts in the candidate.

4.4.4 The Perceptron Algorithm and Some Variants

The perceptron algorithm is one of the oldest algorithms used in machine learning. It is an online algorithm for learning a linear threshold function. Suppose that $D = \{(x^i, y^i) | x^i \in R^n, y^i \in \{+1, -1\}, \forall i = 1, 2, \ldots, m\}$ is a set of m training samples, the set of positive samples D^+ and the set of negative samples D^- are defined as follows:

$$D^+ = \{(x^i, y^i) | (x^i, y^i) \in D, y^i = +1\}, \tag{4.3}$$

$$D^- = \{(x^i, y^i) | (x^i, y^i) \in D, y^i = -1\}. \tag{4.4}$$

The purpose of the perceptron algorithm is to learn a parameter vector $w \in R^n$ so that:

$$w \cdot x^i = \sum_{j=1}^{n} w_j * x^i_j > 0, \forall (x^i, y^i) \in D^+, and \tag{4.5}$$

$$w \cdot x^i = \sum_{j=1}^{n} w_j * x^i_j \leq 0, \forall (x^i, y^i) \in D^-. \tag{4.6}$$

The perceptron algorithm is presented as Algorithm 1, where T is the number of iterations. The main step of the algorithm is checking whether the perceptron (with current value of the parameter vector) predicts samples correctly or not. If there is a mistake, it will update the parameter vector (line 6 in the algorithm). Since $y^i \in \{+1, -1\}$, the update formula can be expressed in two cases:

$$w = w + x^i, for\ y^i = +1 \tag{4.7}$$

and

$$w = w - x^i, for\ y^i = -1. \tag{4.8}$$

Algorithm 1 The perceptron algorithm

```
1: Initialize: $w \leftarrow 0$
2: for $t = 1, 2, \ldots, T$ do
3:     for $i = 1, 2, \ldots, m$ do
4:         Predict $z^i = +1$ if $w \cdot x^i > 0$, otherwise $z^i = -1$
5:         if $z^i \neq y^i$ then
6:             Update: $w \leftarrow w + y^i * x^i$
7:         end if
8:     end for
9: end for
```

Now we describe some variants of the perceptron algorithm for reranking[1]. Suppose that $D = \{(x^i, y^i) | y^i \in C, \forall i = 1, 2, \ldots, m\}$ is a set of m training samples, where C is a set of predefined categories. We define a feature function $\Phi(y)$, $\Phi(y) \in R^n$. We want to learn a parameter vector w so that:

$$F(x^i) = argmax_{y \in GEN(x^i)} \Phi(y) \cdot w = y^i, \forall i = 1, 2, \ldots, m, \tag{4.9}$$

where $GEN(x)$ is the GEN component in the reranking model, $GEN(x) \subseteq C$.

A variant of the perceptron algorithm for reranking is shown as Algorithm 2, where T is also the number of iterations. At the beginning, the parameter vector is initialized zero values. In each iteration, the algorithm predicts the label of each instance using the current value of the parameter vector. The predicted label is the label that maximizes the score function $score(y) = \Phi(y) \cdot w$. If there is a mistake (the predicted label is different from the gold label), the values of the parameter vector are updated as follows:

$$w = w + \Phi(y^i) - \Phi(z^i). \tag{4.10}$$

Algorithm 2 A variant of the perceptron algorithm for reranking

```
1: **Inputs**: Training set $\{(x^i, y^i) | x^i \in R^n, y^i \in C, \forall i = 1, 2, \ldots, m\}$
2: **Initialize**: $w \leftarrow 0$
3: **Define**: $F(x) = argmax_{y \in GEN(x)} \Phi(y) \cdot w$
4: for $t = 1, 2, \ldots, T$ do
5:     for $i = 1, 2, \ldots, m$ do
6:         $z^i \leftarrow F(x^i)$
7:         if $z^i \neq y^i$ then
8:             Update: $w \leftarrow w + \Phi(y^i) - \Phi(z^i)$
9:         end if
10:    end for
11: end for
12: **Output**: Parameter vector $w$.
```

Another variant of the perceptron algorithm for reranking, the average perceptron algorithm, is shown as Algorithm 3. In the average perceptron algorithm, in addition to parameter vector w, we store an average parameter vector w_{avg}. At the beginning, the average parameter vector is also initialized zero values. In each iteration and for each sample, we update the average parameter vector as follows[2]:

$$w_{avg} = w_{avg} + w. \tag{4.11}$$

[1] These algorithms were presented in Collins' papers [7, 8].

[2] The goal of this step is to calculate the total of parameters in all iterations.

Note that, we always update w_{avg} for each sample (even if w is not updated). Finally, we calculate the average values:

$$w_{avg} = w_{avg}/(mT), \tag{4.12}$$

where T is the number of iterations and m is the number of training samples. Collins (in [7]) showed that the average perceptron algorithm performs sinificantly better than the final parameter perceptron algorithm (Algorithm 2) in some tasks. In our model, we choose the average perceptron algorithm (Algorithm 3) to learn the parameter vector.

Algorithm 3 Average perceptron algorithm for reranking

```
1: Inputs: Training set $\{(x^i, y^i)|x^i \in R^n, y^i \in C, \forall i = 1, 2, \ldots, m\}$
2: Initialize: $w \leftarrow 0, w_{avg} \leftarrow 0$
3: Define: $F(x) = argmax_{y \in GEN(x)} \Phi(y) \cdot w$
4: for $t = 1, 2, \ldots, T$ do
5:    for $i = 1, 2, \ldots, m$ do
6:       $z^i \leftarrow F(x^i)$
7:       if $z^i \neq y^i$ then
8:          $w \leftarrow w + \Phi(y^i) - \Phi(z^i)$
9:       end if
10:      $w_{avg} \leftarrow w_{avg} + w$
11:   end for
12: end for
13: $w_{avg} \leftarrow w_{avg}/(mT)$
14: Output: Parameter vector $w_{avg}$.
```

4.4.5 Experiments on Reranking Model

The architecture of our reranking model for the RRE task is illustrated in Figure 4.3. First, the annotated corpus was divided into three parts: the training set (80%), the development set (10%), and the test set (10%). The training set was used for training a BC-IOE model. Then, this model was tested on the development set to learn a parameter vector using the average Perceptron algorithm [7, 8]. We also trained another BC-IOE model (using both the training set and the development set), and used this model as the GEN component of the reranking model. With each sample, we chose 20-best outputs as candidates.

In the decoding phase, we only performed reranking on samples for which the probabilities output by GEN are less than a threshold (Algorithm 4).

Experimental results on the Japanese National Pension Law corpus are shown in Table 4.4, in which the iteration number is set to 10 and the threshold is set to 0.5. The reranking model improves by 0.40% in the $F_{\beta=1}$ score (3.4% in error rate) compared with

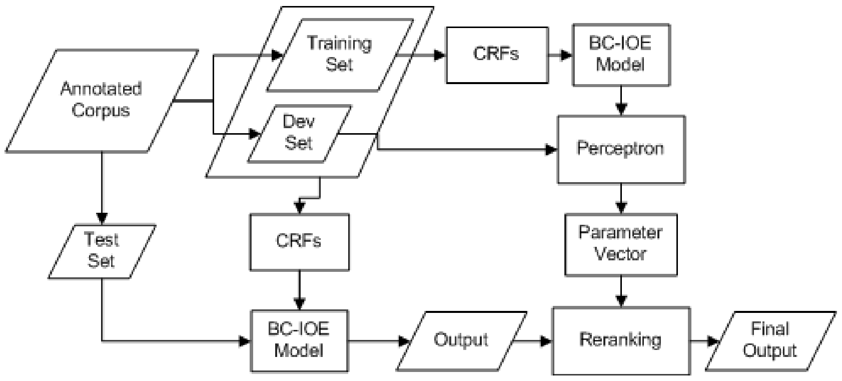


Figure 4.3: Reranking model.

Algorithm 4 Decoding in the reranking model

```
1: for each sample $x$ do
2:     if the highest probability output by GEN is greater than a threshold then
3:         $y$ is the output with the highest probability of GEN
4:     else
5:         $F(x) = argmax_{y \in GEN(x)} score(y) = argmax_{y \in GEN(x)} \Phi(y) \cdot W$
6:     end if
7: end for
```

the best model before (BC-IOE), and by 2.2% in the $F_{\beta=1}$ score (15.9% in error rate) compared with the baseline model.

Table 4.4: Results of the reranking model

Tag	**Precision(%)**	**Recall(%)**	**F$_{\beta=1}$(%)**
C	91.50	92.48	**91.99**
EL	0.00	0.00	0.00
ER	0.00	0.00	0.00
A	91.28	87.88	**89.55**
T_1	100.00	22.22	36.36
T_2	87.90	91.81	**89.82**
T_3	74.67	54.90	**63.28**
Overall	**89.42**	**87.75**	**88.58**

4.4.6 Error Analysis

In this sub-section, we discuss some cases in which our model fails. First, we could not analyze sentences of equivalence type (EL and ER parts). This is understandable, because in our corpus only 1.5% of sentences belong to this type, while 98.5% of sentences belong to the implication type. Part T_1 is a similar case ($F_{\beta=1} = 36.36\%$). Another case is when we discriminated between T_3 and T_2. This is a difficult situation, because we need semantic information to recognize T_3 correctly (there is a slight difference between T_3 and T_2). In some other situations, we fail when the input sentence is too long and complicated. In these cases, our model usually splits a correct part into two parts or merges two correct parts into one part.

In statistical machine learning, when the number of instances of a type X is small (both in the training set and in the test set), the model rarely predicts the label X. For this reason, the recall of a rare type is usually very low. Even the model predicts type X in a small number of times, if it predicts correctly in the cases it learned in the training set, the precision will be high. It is the case of T_1 in the RRE task for the corpus we used (low recall but high precision). If the model predicts wrongly, the precision will be low. It is the case of EL and ER in the RRE task (low recall and low precision). In this case, data is insufficient for the model to learn. Whether the precision may be low or high, the $F_{\beta=1}$ score is often low because the recall is very low.

4.5 Conclusions

In this chapter, we presented our Bunsetsu-based model for the RRE task, in which we consider a law sentence as a sequence of Bunsetsus. Experimental results on the JNPL corpus showed that the Bunsetsu-based model outperforms the word-based model. This is reasonable, because the Bunsetsu-based model has some advantages in comparison with the word-based model. The Bunsetsu-based model reduces the length of the input sequence by three times compared with the word-based model. Bunsetsu-based model does not need to concentrate on words in the middle of a Bunsetsu. Hence, it can reduce significantly the search space. We also showed that using a linear score function to rerank N-best outputs can improve the result of the Bunsetsu-based model.

In Chapter 3 and Chapter 4 (this chapter), we investigated the RRE task using supervised learning methods. Supervised methods using labeled data which are expensive and time consuming to obtain. In the next chapter, we will study how to exploit unlabeled data (easy to collect) to improve the RRE task.

Chapter 5

A Simple Semi-supervised Learning Method for RRE

In this chapter, we present a simple semi-supervised learning method for the RRE task using extra word features. First, we give an overview of semi-supervised learning in NLP (Section 5.1). Next, we present Brown clustering method, a simple and efficient method for word representation (Section 5.2). Then, we describe how to exploit extra word features extracted from Brown clusters to improve the RRE task (Section 5.3). Finally, some conclusions are given (Section 5.4).

5.1 Semi-supervised Learning in NLP

In NLP tasks, one natural question is how to utilize unlabeled data to improve the performance of a system with a fixed amount of labeled data. This problem is known as semi-supervised learning. Several semi-supervised learning approaches have been proposed. One traditional approach is using both labeled data and unlabeled data to train a system. Self-training is a popular method belonging to this approach [12, 47]. In self-training, a model is first trained with labeled data, and then used to label the unlabeled data. Usually, the most confident unlabeled samples, together with their predicted labels, are added to the training set. This procedure is repeated until a stop condition is satisfied.

Co-training [2, 47] is another well-known method in semi-supervised learning. In co-training, we assume that some assumptions on data are satisfied: the feature set can be divided into two sub-sets; each sub-set is sufficient to train a good classifier; and the two sub-sets are conditionally independent given the class. First, two classifiers are train on labeled data, each classifier using one sub-set of features. Then, each classifier labels unlabeled data, and most confident unlabeled instances (together with their predicted labels) are used to train the other classifier. Like in self-training, the second step is also repeated until a stop condition is satisfied.

A new approach in semi-supervised learning bases on the idea of *transductive inference* [43]. There are two kinds of settings for learning in Artificial Intelligence (AI) field: *inductive inference* and *transductive inference*. In *inductive inference*, first we train a global model based on labeled data, and then use this model to classify unlabeled data in the test phase. By contrast, a *transductive* learner predicts labels for test data directly without learning a global model. It considers both labeled data and unlabeled data (test data) in the learning process. The advantage of this approach is that it can focus on test data. In traditional approach, we need to learn a global model from the whole problem space and classify new unlabeled instances by it. However, such a global model is hard to obtain when training data are not enough. In addition, this global model may be unnecessary when we only care for specific data.

Transductive SVM [14, 47] and graph-based semi-supervised methods [47] are instances which belong to the transductive inference approach. Transductive SVM (TSVM) is an extension of traditional support vector machines. In TSVM, both labeled data and unlabeled data (test data) are used to find the maximum margin. Graph-based semi-supervised methods define a graph where nodes are labeled and unlabeled samples, and edges reflect the similarity between samples. The goal is to learn a function f on the graphs such that f classifies correctly labeled nodes, and f should be smooth on the whole graph.

Another approach is to learn some underlying predictive functional structures from unlabeled data, and then use these structures to support learning models. Alternating structure optimization (ASO) algorithm belongs to this second approach [1]. The key idea of ASO algorithm is to learn predictive functional structures by considering simultaneously multiple prediction problems. By doing this, we can find the common structures shared by some predictors, and then use the information about these structures to improve each individual problem.

Recently, a simple and general semi-supervised learning method has been proposed. The main idea of this method is to use *unsupervised* word representation as extra word features of a *supervised* model [42]. Some kinds of word representations have been investigated (including Brown clusters [3], Collobert and Weston embeddings [9], and HLBL embeddings [27]) and applied successfully in many NLP tasks such as chunking and named entity recognition [42], dependency parsing [19], semantic dependency parsing [46], etc. Among word representation methods, Brown clustering produces better word representations than embeddings, especially for rare words. It is the most popular method, and also gives the highest accuracy in some tasks [42].

In the rest of this chapter, we will describe briefly Brown clustering algorithm and show how to use Brown word clusters to improve the RRE task. Among word representation methods, we chose Brown clustering algorithm for our work because of its simplicity and efficiency.

5.2 Brown Clustering

A main goal of word clustering is to deal with the problem of data sparsity by providing a lower-dimensional representation of words [24]. A good clustering should produce clusters so that the words in the same cluster should be similar in some definitions. However, how we can measure the similarity between words if we only have a raw text? Usually, two similar words are defined as two words that appear in similar contexts or that they are exchangeable to some extent [24].

The Brown clustering algorithm is a word clustering algorithm based on the mutual information of bigrams [3]. Given an input text $w_1, \ldots, w_n$ (the raw text can be presented as a sequence of words), we want to find a clustering C that map each word w_i to a cluster $C(w_i)$ so that the quality of C is maximized. The quality of clustering C is defined as the logarithm of the probability that a *class-based bigram language model* (see Figure 5.1) assigns to the input text, normalized by the length of the text.

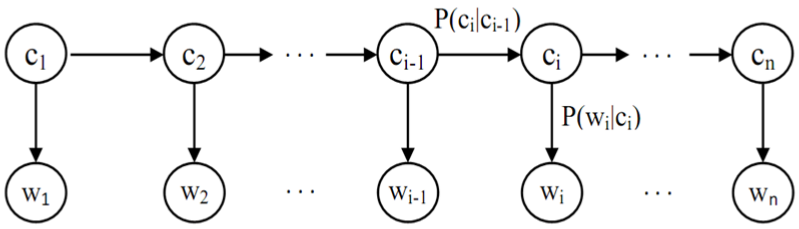


Figure 5.1: The class-based bigram language model ($c_i = C(w_i)$).

$$Quality(C) = \frac{1}{n} log\ P(w_1, \ldots, w_n). \tag{5.1}$$

Using the deterministic property of C, and from the definition of the model, the quality of C can be rewritten as follows:

$$\begin{aligned} Quality(C) &= \frac{1}{n} log\ P(w_1, \ldots, w_n, C(w_1), \ldots, C(w_n)) \\ &= \frac{1}{n} log\ \prod_{i=1}^{n} P(C(w_i)|C(w_{i-1}))P(w_i|C(w_i)) \\ &= \frac{1}{n} \sum_{i=1}^{n} log\ P(C(w_i)|C(w_{i-1}))P(w_i|C(w_i)). \end{aligned} \tag{5.2}$$

Let $count(w)$ and $count(w, w')$ be the number of times word w and bigram (w, w') appear in the input text, respectively. The number of times a word in cluster c appears in the input text, denoted by $count(c)$, will be defined as follows:

$$count(c) = \sum_{w \in c} count(w) \tag{5.3}$$

and the number of times a bigram with the first word in cluster c and the second word in cluster c' appears in the text, denoted by $count(c, c')$, will be defined:

$$count(c, c') = \sum_{w \in c, w' \in c'} count(w, w'). \tag{5.4}$$

The quality of C now becomes:

$$\begin{aligned}
Quality(C) &= \sum_{w,w'} \frac{count(w,w')}{n} log\ P(C(w')|C(w))P(w'|C(w')) \\
&= \sum_{w,w'} \frac{count(w,w')}{n} log\ \frac{count(C(w), C(w'))}{count(C(w))} \frac{count(w')}{count(C(w'))} \\
&= \sum_{w,w'} \frac{count(w,w')}{n} log\ \frac{n * count(C(w), C(w'))}{count(C(w))count(C(w'))} \frac{count(w')}{n} \\
&= \sum_{w,w'} \frac{count(w,w')}{n} log\ \frac{n * count(C(w), C(w'))}{count(C(w))count(C(w'))} + \sum_{w,w'} \frac{count(w,w')}{n} log\ \frac{count(w')}{n} \\
&= \sum_{c,c'} \frac{count(c,c')}{n} log\ \frac{n * count(c, c')}{count(c)count(c')} + \sum_{w'} \frac{count(w')}{n} log\ \frac{count(w')}{n} \\
&= \sum_{c,c'} P(c,c') log\ \frac{P(c,c')}{P(c)P(c')} + \sum_{w} P(w) log\ P(w) \\
&= I(C) - H.
\end{aligned} \tag{5.5}$$

where $P(w) = \frac{count(w)}{n}$, $P(c) = \frac{count(c)}{n}$, and $P(c, c') = \frac{count(c,c')}{n}$ are empirical distributions over words, clusters, and pairs of clusters, respectively; and $I(C) = \sum_{c,c'} P(c, c')log\ \frac{P(c,c')}{P(c)P(c')}$ and $H = -\sum_w P(w)log\ P(w)$ are the mutual information between adjacent clusters and the entropy of the word distribution, respectively.

Because entropy H is independent on C, we want to find clustering C that maximizes mutual information $I(C)$. Brown (in [3]) presents a greedy algorithm that can find the approximation solution in $O(k^3)$ time, where k is the number of different word types[1]. In the initial step, each word belongs to its own individual cluster. The algorithm then gradually groups clusters to build a hierarchical clustering of words. In each step, two clusters are merged so that the loss in mutual information $I(C)$ is least[2].

[1]There is no practical method for finding the clustering that maximizes mutual information $I(C)$.

[2]Brown clustering algorithm produces a hard clustering. Each word belongs to exactly one cluster.

Figure 5.2 shows an example of Brown word-cluster hierarchy in a binary tree style. In this tree, each leaf node corresponds to a word, which is uniquely identified by the path from the root node to it. This path can be represented by a bit string, as shown in Figure 5.2. From the root node, we add bit 0 to the left branch and bit 1 to the right branch.

A word-cluster hierarchy is reduced to depth n if all words with the same n-bit prefix are grouped in one cluster. For example, if the word-cluster hierarchy in Figure 5.2 is reduced to depth 2, we will obtain a new hierarchy in Figure 5.3. In the new tree, we only have four leaf nodes according to four clusters: 00, 01, 10, and 11.

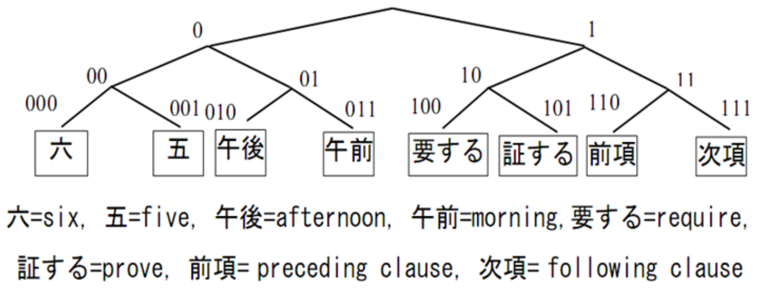

0
1
00
01
10
11
000
001
010
011
100
101
110
111
六
五
午後
午前
要する
証する
前項
次項
六=six, 五=five, 午後=afternoon, 午前=morning, 要する=require,
証する=prove, 前項= preceding clause, 次項= following clause

Figure 5.2: An example of Brown word-cluster hierarchy.

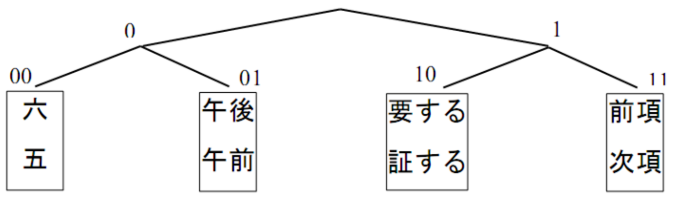


Figure 5.3: A Brown word-cluster hierarchy after reduction to depth 2.

Features extracted at n-bit depth are binary strings with length n. By reducing the word-cluster tree to different values of depth n, we can group words at various levels, from coarse clusters (small value of n) to fine clusters (large value of n).

5.3 RRE with Extra Word Features

5.3.1 Framework

The main idea of our semi-supervised learning method is to use *unsupervised* word representations as extra word features of a *supervised* model. We use Brown word clusters as the word representation method. In this framework, *unlabeled data* are used to produce word clusters. From these word clusters, we extract extra word features, and add these features to a *supervised* model (*labeled data* are used to train this model). Figure 5.4 shows our semi-supervised learning framework. This framework consists of two phases: *unsupervised* phase with the Brown clustering algorithm, and *supervised* phase with CRFs.

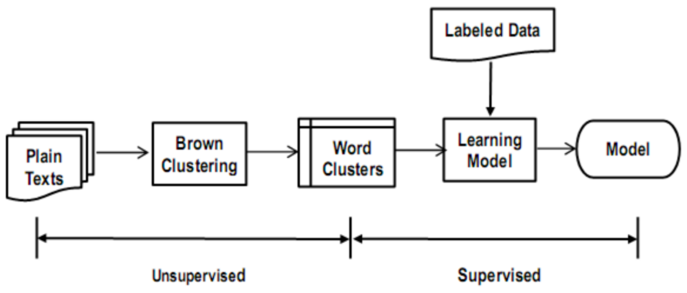


Figure 5.4: Semi-supervised learning framework.

5.3.2 Word Cluster Extraction and Feature Design

To produce word representations, we first collected plain text from the address *http://www.japaneselawtranslation.go.jp*[3]. Our plain text corpus consists of 30 acts with more than 13 thousand Japanese law sentences. Table 5.3 lists 30 acts in our plain text corpus. After word segmenting (using Cabocha tool [20]), we conducted the Brown clustering algorithm to cluster words. In our work, we used the implementation of Percy Liang [24], and the number of clusters was set to 200.

We designed features similar to features presented in [19]. Details of feature design are shown in Table 5.1, where n is an integer. For example, a feature hw[0]-4=0101 will receive value 1 if 4-bit prefix of the head word of the current Bunsetsu is 0101, otherwise it will receive value 0.

Table 5.1: Feature Design.

Feature	Meaning
hw[-1]-n	n-bit prefix of the head word of the previous Bunsetsu
hw[0]-n	n-bit prefix of the head word of the current Bunsetsu
hw[1]-n	n-bit prefix of the head word of the next Bunsetsu
fw[-1]-n	n-bit prefix of the functional word of the previous Bunsetsu
fw[0]-n	n-bit prefix of the functional word of the current Bunsetsu
fw[1]-n	n-bit prefix of the functional word of the next Bunsetsu

[3]This website provides many Japanese law articles in both Japanese and English. We downloaded articles available on the website in March, 2010.

5.3.3 Experiments

For the RRE task, we extracted features at 4-bit depth and 6-bit depth. We integrated these features into two models: the Bunsetsu-based model and the reranking model. The experimental results of the semi-supervised method with extra word features are shown in Figure 5.5. In both models, the semi-supervised method outperforms the supervised method. For the Bunsetsu-based model, the $F_{\beta=1}$ score was 88.63%, compared with 88.18% for the supervised method. The reranking model got 88.84% in the $F_{\beta=1}$ score, compared with 88.58% for the supervised method. The Bunsetsu-based model using the semi-supervised method even got better results than the reranking model using the supervised method (88.63%, compared with 88.58%).

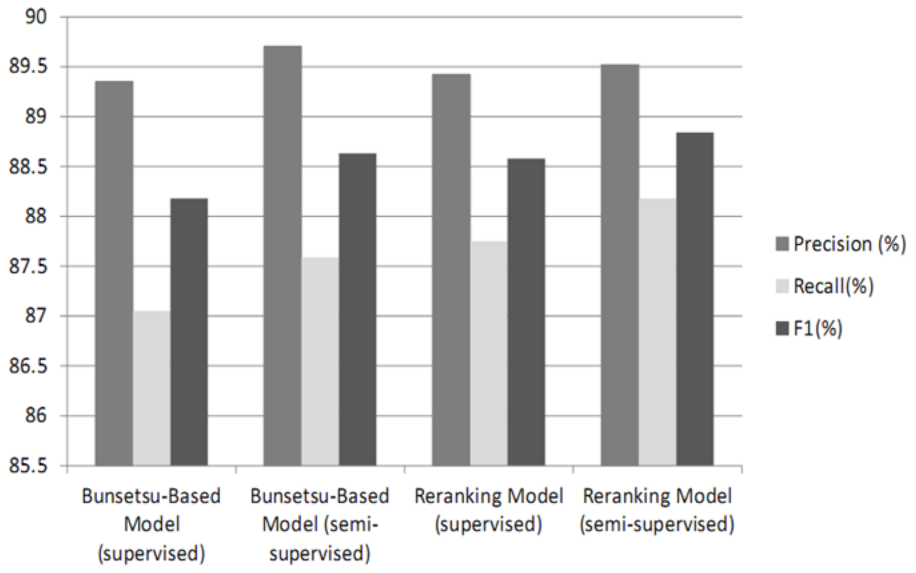


Figure 5.5: Comparison between the supervised method and the semi-supervised method.

We conducted additional experiments to evaluate the effect of the Brown cluster features as the amount of training data is varied. In these experiments, the Bunsetsu-based model was exploited. We started with 5% of training data, and then gradually doubled the amount of training data to 10%, 20%, 40%, and 80%. Experimental results are shown in Table 5.2, in which the last column indicates the improvement of the model using features extracted from Brown clusters compared with the model not using these features. In all experiments, the semi-supervised method outperformed the supervised method. Figure 5.6 illustrates experimental results in graph. We note that, when the amount of training data is small, the improvement is bigger (about 1.5% when using less than 20% of training data, and about 0.4% when using more training data).

Table 5.2: Experiments with different training sizes.

Training size	Supervised	Semi-supervised	Δ
5%	64.03	65.51	1.48
10%	72.83	74.06	1.23
20%	81.71	83.54	1.83
40%	83.27	83.80	0.53
80%	87.90	88.11	0.21
All	88.18	88.63	0.45

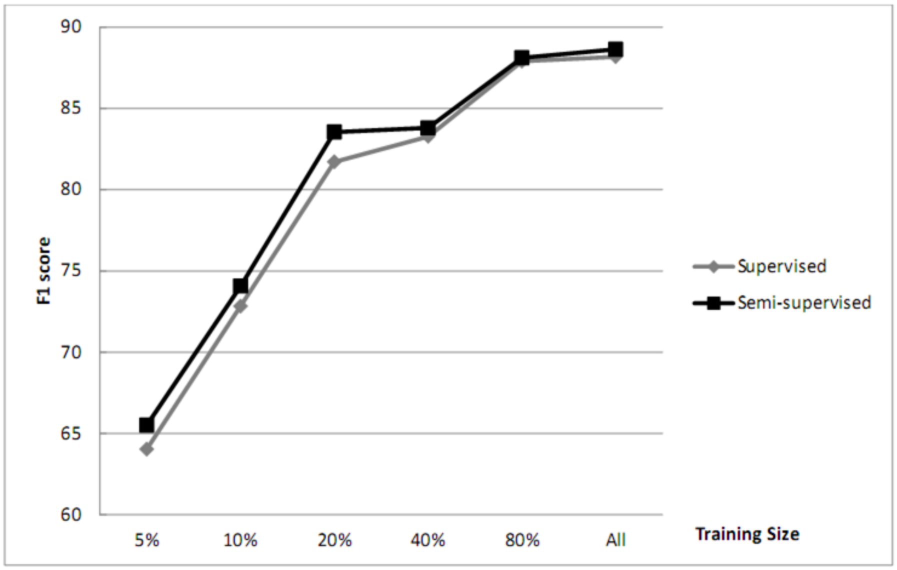


Figure 5.6: Experimental results illustrated in graph.

5.4 Conclusions

In this chapter, we studied how to exploit unlabeled data to improve the RRE task. We presented a simple semi-supervised learning method using Brown clusters as extra word features for a supervised learning model. We showed that, for the RRE task:

1. Using unlabeled data combining with labeled data in a semi-supervised learning model can improve the RRE task compared with using only labeled data.

2. Using unlabeled data is more effective when the amount of labeled data is small.

Table 5.3: Acts in the plain text corpus

#	Act Name	Act Number
1	Administrative Procedure Act	Act No. 88 of November 12, 1993
2	Bank of Japan Act	Act No. 89 of June 18, 1997
3	Child Welfare Act	Act No. 164 of December 12, 1947
4	City Planning Act	Act No. 100 of June 15, 1968
5	Civil Code Act	Act No. 89 of April 27, 1896
6	Civil Execution Act	Act No. 4 of March 30, 1979
7	Commercial Registration Act	Act No. 125 of July 9, 1963
8	Commodity Exchange Act	Act No. 239 of August 5, 1950
9	Companies Act	Act No. 86 of July 26, 2005
10	The Constitution of Japan	Constitution November 3, 1946
11	Consumer Product Safety Act	Act No. 31 of June 6, 1973
12	Corporation Tax Act	Act No. 34 of March 31, 1965
13	Basic Act on Crime Victims	Act No. 161 of December 8, 2004
14	Electrical Appliance and Material Safety Act	Act No. 234 of November 16, 1961
15	Electricity Business Act	Act No. 170 of July 11, 1964
16	Act on Electronic Signatures and Certification Business	Act No. 102 of May 31, 2000
17	Gas Business Act	Act No. 51 of March 31, 1954
18	Income Tax Act	Act No. 33 of March 31, 1965
19	Labor Insurance Act	Act No. 84 of December 9, 1969
20	Basic Act for Land	Act No. 84 of December 22, 1989
21	Long-Term Care Insurance Act	Act No. 123 of December 17, 1997
22	Measurement Act	Act No. 51 of May 20, 1992
23	National Government Organization Act	Act No. 120 of July 10, 1948
24	National Public Service Ethics Act	Act No. 129 of August 13, 1999
25	Act on Nippon Telegraph and Telephone Corporation	Act No. 85 of December 25, 1984
26	Plant Protection Act	Act No. 151 of May 4, 1950
27	Professional Engineer Act	Act No. 25 of April 27, 1983
28	Quarantine Act	Act No. 201 of June 6, 1951
29	Act on the Rational Use of Energy	Act No. 49 of June 22, 1979
30	Services and Supports for Persons with Disabilities Act	Act No. 123 of November 7, 2005

Chapter 6

Recognition of Requisite Parts and Effectuation Parts in Paragraphs Consisting of Multiple Sentences

In the previous chapters (Chapters 2,3,4, and 5), we have investigated the RRE task, where the input is a single law sentence. In this chapter, we will present a study on the RREP task, in which the input of the task is a paragraph consisting of multiple sentences. In this task, we also examine cases where a logical part contains other logical parts. In addition to recognizing logical parts, we also recognize logical structures between logical parts. A logical structure is a set of some related logical parts.

The remainder of this chapter is organized as follows. First, we formulate the RREP task as two sub-tasks: *Recognition of Logical Parts* and *Recognition of Logical Structures* (Section 6.1). Then, we present our proposed solutions for two sub-tasks: *multi-layer sequence learning model* for Sub-task 1 and *graph-based model* for Sub-task 2 (Section 6.2). Next, we describe experiments on two sub-tasks on the Japanese National Pension Law (JNPL) corpus (Section 6.3). Finally, we provide some conclusions (Section 6.4).

6.1 Formulation

6.1.1 Sub-Task 1: Recognition of Logical Parts

Recall that the goal of this sub-task is to recognize logical parts given a paragraph consisting of multiple sentences.

Let s be a law sentence[1] in the law sentence space S, then s can be represented by a sequence of words $s = [w_1 w_2 \dots w_n]$. A legal paragraph x in the legal paragraph space

[1]s may be a complete or non-complete sentence.

X is a sequence of law sentences $x = [s_1 s_2 \dots s_l]$, where $s_i \in S, \forall i = 1, 2, \dots, l$. For each paragraph x, we denote a logical part p by a quad-tuple $p = (b, e, k, c)$ where b, e, and k are three integers which indicate *beginning position*, *end position*, and *sentence position* of p, and c is a logical part category in the set of predefined categories C. Formally, the set P of all possible logical parts defined in a paragraph x can be described as follows:

$$P = \{(b, e, k, c) | 1 \leq k \leq l, 1 \leq b \leq e \leq len(k), c \in C\}. \tag{6.1}$$

In the above definition, l is the number of sentences in the paragraph x, and $len(k)$ is the length of the k^{th} sentence.

In this sub-task, we want to recognize some *non-overlapping* (but possibly *embedded*) logical parts in an input paragraph. A solution for this task is a subset $y \subseteq P$ which does not violate the *overlapping* relationship. We say that two logical parts p_1 and p_2 are *overlapping* if and only if they are in the same sentence ($k_1 = k_2$) and $b_1 < b_2 \leq e_1 < e_2$ or $b_2 < b_1 \leq e_2 < e_1$. We denote the *overlapping* relationship by $\sim$. We also say that p_1 is *embedded* in p_2 if and only if they are in the same sentence ($k_1 = k_2$) and $b_2 \leq b_1 \leq e_1 \leq e_2$, and denote the *embedded* relationship by $\prec$. Formally, the solution space can be described as follows:

$$Y = \{y \subseteq P | \forall u, v \in y, u \not\sim v\}. \tag{6.2}$$

The learning problem in this sub-task is to learn a function $R : X \rightarrow Y$ from a set of m training samples $\{(x^i, y^i) | x^i \in X, y^i \in Y, \forall i = 1, 2, \dots, m\}$.

In this task, we consider the following types of logical parts:

1. An antecedent part is denoted by A
2. A consequent part is denoted by C
3. A topic part which depends on the antecedent part is denoted by T_1
4. A topic part which depends on the consequent part is denoted by T_2
5. A topic part which depends on both the antecedent part and the consequent part is denoted by T_3
6. The left part of an equivalent statement is denoted by EL
7. The right part of an equivalent statement is denoted by ER
8. An object part, whose meaning is defined differently in different cases, is denoted by Ob
9. An original replacement part, which will be replaced by other replacement parts (denoted by $RepR$) in specific cases, is denoted by $RepO$.

Compared with the RRE task, we introduce three new kinds of logical parts: *Ob*, *RepO*, and *RepR*.

6.1.2 Sub-Task 2: Recognition of Logical Structures

Recall that the goal of this sub-task is to recognize logical structures given a set of logical parts in a paragraph consisting of multiple sentences.

Let $y \subseteq P$ be a set of logical parts in a paragraph x, and $2^{|y|}$ be the set of all the subsets of y. The logical structure of a formula is a subset of y which contains at least two elements. A solution for this sub-task is a subset $z \subset 2^{|y|}$ that satisfies the following constraints:

1. $\forall u \in z, |u| \geq 2$,
2. $\cup_{u \in z} u = y$,
3. $\forall u, v \in z$, if $u \subseteq v$ then $u = v$,
4. $\forall u \in z, \cup_{v \in z, v \neq u} v \neq y$.

Constraint 1) says that each logical structure must contain at least two logical parts. Constraint 2) says that each logical part must belong to at least one logical structure. Constraint 3) says that we cannot have two different logical structures such that the set of logical parts in one logical structure contains the set of logical parts in the other logical structure. Constraint 4) says that if we remove any logical structure from the solution, Constraint 2) will be violated. Although Constraint 3) is guaranteed by Constraint 4), we introduce it because of its importance.

Let Z be the solution space, $Z \subset 2^{2^{|y|}}$. Usually, Z is a huge set even with a small set y. The learning problem in this sub-task is to learn a function $I : Y \rightarrow Z$ from a set of m training samples $\{(y^i, z^i) | y^i \in Y, z^i \in Z, \forall i = 1, 2, \ldots, m\}$.

6.2 Proposed Solutions

6.2.1 Multi-layer Sequence Learning Model for Logical Part Recognition

This sub-section presents our model for recognizing logical parts. We consider the recognition problem as a multi-layer sequence learning problem. First, we give some related notions.

Let s be a law sentence, and P be the set of logical parts of s, $P = \{p_1, p_2, \ldots, p_m\}$. $Layer^1(s)$ (outer most layer) is defined as a set of logical parts in P, which are not

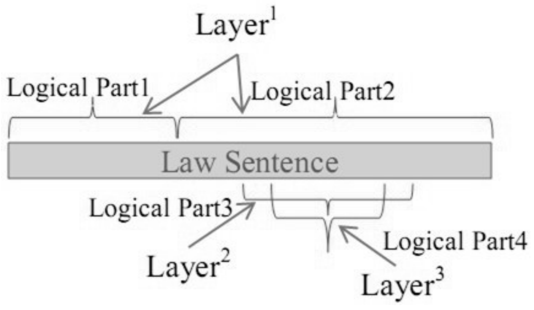


Figure 6.1: A law sentence with logical parts in three layers.

embedded in any other part. $Layer^i(s)$ is defined as a set of logical parts in $P \backslash \cup_{k=1}^{i-1} Layer^k(s)$, which are not embedded in any other part in $P \backslash \cup_{k=1}^{i-1} Layer^k(s)$. Formally, we have:

$$Layer^1(s) = \{p | p \in P, p \not\prec q, \forall q \in P, q \neq p\}, \tag{6.3}$$

$$Layer^i(s) = \{p | p \in Q^i, p \not\prec q, \forall q \in Q^i, q \neq p\}, \tag{6.4}$$

where

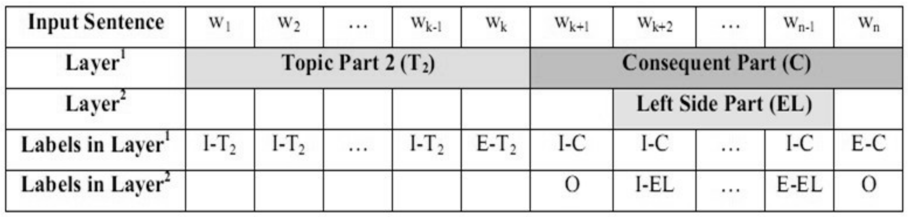
$$Q^i = P \backslash \cup_{k=1}^{i-1} Layer^k(s). \tag{6.5}$$

Figure 6.1 illustrates a law sentence with four logical parts in three layers: Part 1 and Part 2 in $Layer^1$, Part 3 in $Layer^2$, and Part 4 in $Layer^3$.

Let K be the number of layers in a law sentence s, our model will recognize logical parts in K steps. In the k^{th} step we recognize logical parts in $Layer^k$. In each layer, we model the recognition problem as a sequence labeling task, in which each word is an element. Logical parts in $Layer^{i-1}$ will be used as input sequence in the i^{th} step (in the first step, we use original sentence as input).

Figure 6.2 gives an example of labeling for an input sentence. The sentence consists of three logical parts in two layers. In our model, we use IOE tag setting: the last element of a part is tagged with E, the other elements of a part are tagged with I, and an element not included in any part is tagged with O.

Let K^* be the maximum number of layers in all law sentences in training data. We learn K^* models, in which the k^{th} model is learned from logical parts in the $Layer^k$ of training data, using Conditional random fields [23, 21]. In the testing phase, we first apply the first model to the input law sentence, and then apply the i^{th} model to the predicted logical parts in $Layer^{i-1}$.



Input Sentence	w_1	w_2	…	w_{k-1}	w_k	w_{k+1}	w_{k+2}	…	w_{n-1}	w_n
Layer^1	Topic Part 2 (T_2)					Consequent Part (C)				
Layer^2							Left Side Part (EL)			
Labels in Layer^1	I-T_2	I-T_2	…	I-T_2	E-T_2	I-C	I-C	…	I-C	E-C
Labels in Layer^2						O	I-EL	…	E-EL	O

Figure 6.2: An example of labeling in the multi-layer model.

6.2.2 A Graph-based Model for Recognition of Logical Structures

Let $G =< V, E >$ be a complete graph with the vertex set V and the edge set E. A real value function f is defined on E as follows:

$f : E \to R$, $e \in E \mapsto f(e) \in R$.

In this sub-task, each vertex of the graph corresponds to a logical part, and a complete sub-graph corresponds to a logical structure. The value on an edge connecting two vertices expresses the degree to which the two vertices belong to one logical structure. The positive (negative) value means that two vertices are likely (not likely) to belong to one logical structure.

Let G_s be a complete sub-graph of G, then $v(G_s)$ and $e(G_s)$ are the set of vertices and the set of edges of G_s, respectively. We define the total value of a sub-graph as follows:

$$f(G_s) = f(e(G_s)) = \sum_{e \in e(G_s)} f(e). \qquad (6.6)$$

Let Ω be the set of all complete sub-graphs of G. The problem becomes determining a subset $X \subseteq \Omega$ such that:

1. $\forall x \in X, |x| \geq 2$,
2. $\cup_{x \in X} v(x) = V$,
3. $\forall x_1, x_2 \in X, x_1 \subseteq x_2 \Rightarrow x_1 = x_2$,
4. $\forall x \in X, \cup_{y \in X, y \neq x} v(y) \neq V$, and
5. $\sum_{x \in X} f(x) \to$ maximize.

Suppose that $|V| = n$, then $|\Omega| = 2^n$, and the number of subsets of Ω is 2^{2^n}. Here we only consider the cases in which $n > 1$ (in the case $n = 0$ we do not have any logical structure; if $n = 1$, we have a special logical structure with one node).

We now describe a heuristic algorithm to solve this sub-task on graphs. This is an approximate algorithm which satisfies four constraints from 1) to 4). The main idea of our algorithm is selecting as many positive edges as possible, and as few negative edges as possible. We consider two cases:

- Case 1: There is no positive value edge on the input graph.
- Case 2: There are some positive value edges on the input graph.

Our algorithm in the first case is presented as Algorithm 5. Because all the edges have negative values, we build logical structures with as few logical parts as possible. In this case, each logical structure contains exactly two logical parts. So we gradually choose two nodes in the graph with the maximum value on the edge connecting them.

Algorithm 5 Case 1 (no positive edge)

```
1: Initialize: $L \leftarrow \emptyset, V_1 \leftarrow V$
2: while $V_1 \neq \emptyset$ do
3:    if $|V_1| \geq 2$ then
4:       $(u, v) \leftarrow argmax_{u \neq v \in V_1} f(u, v)$
5:    else
6:       Let $v$ be the element in $V_1$
7:       $u \leftarrow argmax_{u \in V} f(u, v)$
8:    end if
9:    $L \leftarrow L \cup \{\{u, v\}\}$
10:   Update $V_1$: $V_1 \leftarrow V_1 \backslash \{u, v\}$
11: end while
12: Return $L$.
```

An example of the first case is illustrated in Figure 6.3. The maximum value on an edge is -0.1, so the first logical structure will contain Node 1 and Node 3. The second logical structure contains Node 2 and Node 4[2].

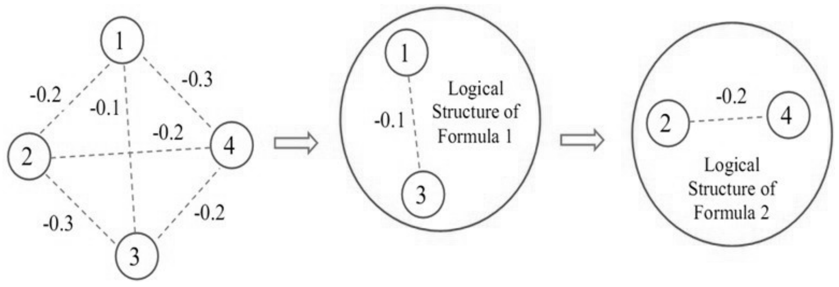
The second case of the algorithm is described as Algorithm 6. First, we consider a graph G^+, which only contains non-negative value edges (E^+). We divide the vertices of G^+ into two subsets, set V_1 of zero-degree vertices and V_2 of other vertices. In the sub-graph with vertex set V_2 and non-negative value edges, we repeatedly build logical structures with as many logical parts as possible. After building successfully a logical structure, we remove all the nodes and the edges according to it on the graph. When have no positive edge, we will build logical structures with exactly two logical parts.

Logical structures with exactly two parts are built in two steps. In the first step[3], we consider one node in V_2, which has not appeared in any logical structure[4]. Then we find

[2]If the number of nodes is odd, the final logical structure will consist of the final node and another node, so that the edge connecting them has the maximal value.

[3]In Algorithm 6, code for this step is described in lines 7 to 13

[4]This node is chosen in $V_2 \backslash R$ in Algorithm 6



-0.2
-0.1
-0.3
-0.2
-0.3
-0.2
Logical Structure of Formula 1
-0.1
-0.2
Logical Structure of Formula 2

Figure 6.3: An example of the first case.

another node, which has appeared in a logical structure[5] so that the edge connecting them has maximum value. A new logical structure is built from these two nodes.

In the second step[6], we consider two nodes u, v in V_1 such that the edge connecting them has maximum value. Then we find two nodes u_1, v_1 (which have appeared in a logical structure) so that the edges (u, u_1), (v, v_1) have maximum values. If $f(u, v) > f(u, u_1) + f(v, v_1)$, a new logical structure is built from two nodes $\{u, v\}$, otherwise two new logical structures are built from two sets of nodes, $\{u, u_1\}$ and $\{v, v_1\}$. To make the algorithm easier to understand, we will provide some examples below.

An example of the second case is illustrated in Figure 6.4. First, we consider the graph with positive edges. This graph consists of five nodes $\{1, 2, 3, 4, 5\}$ and four edges $\{(1, 2), (1, 3), (2, 3), (2, 4)\}$. We have $V_1 = \{5\}$ and $V_2 = \{1, 2, 3, 4\}$. The maximal sub-graph of this graph is the graph with three nodes $\{1, 2, 3\}$, so we have the first logical structure with these three nodes. We remove these nodes and the positive edges connecting to these nodes. We have two nodes $\{4, 5\}$ with no positive edges.

Now we build logical structures with exactly two nodes. In the first step, we consider Node 4 (the remainder node in V_2). Among edges connecting to Node 4, edge $(2, 4)$ has maximal value. So we have the second logical structure with two nodes $\{2, 4\}$. In the second step, we consider Node 5, and we have the third logical structure with two nodes $\{1, 5\}$.

Another example of the second case is shown in Figure 6.5. First, we consider the graph with positive edges. This graph consists of four nodes $\{1, 2, 3, 4\}$ and one edge $\{(1, 2)\}$. We have the set of zero-degree vertices $V_1 = \{3, 4\}$, and the set of other vertices $V_2 = \{1, 2\}$. The maximal sub-graph of this graph is the graph with two nodes $\{1, 2\}$, so we have the first logical structure with these two nodes. We remove these nodes and the positive edges connecting to these nodes. We have two nodes $\{3, 4\}$ with no positive edges, and $f(3, 4) = -0.15$. With Node 3, the maximum edge connecting to it is the edge $(2, 3)$ with $f(2, 3) = -0.1$. Similar to Node 4, the maximum edge connecting to it is the edge $(1, 4)$ with $f(1, 4) = -0.1$. We have[7]:

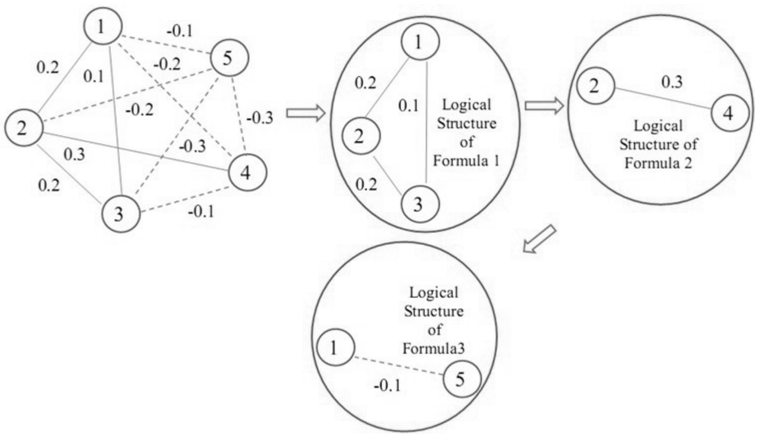
[5]This node is chosen in R in Algorithm 6.

[6]In Algorithm 6, code for this step is described in lines 14 to 35

[7]In Algorithm 6, code for this comparison is described in line 27.

Algorithm 6 Case 2 (have some positive edges)

```
1: Initialize: $L \leftarrow \emptyset$, $G^{'} \leftarrow < V_2, E^+ >$
2: while $e(G^{'}) \neq \emptyset$ do
3:     $g$ is the complete sub-graph of $G^{'}$ that maximizes $f(g)$
4:     $L \leftarrow L \cup \{g\}$
5:     Remove $g$ and edges connecting to a vertex in $g$ from $G^{'}$
6: end while
7: $R \leftarrow \cup_{l \in L} v(l)$, $R^{'} \leftarrow \emptyset$
8: for $v \in V_2 \backslash R$ do
9:     $S \leftarrow \{s \in R | \forall l \in L, v(l) \not\subseteq R^{'} \cup \{s\}\}$
10:    $u \leftarrow argmax_{u \in S} f(u, v)$
11:    $L \leftarrow L \cup \{\{u, v\}\}$
12:    Update $R^{'}$: $R^{'} \leftarrow R^{'} \cup \{u\}$
13: end for
14: while $V_1 \neq \emptyset$ do
15:     if $|V_1| = 1$ then
16:         Let $v$ be the element in $V_1$
17:         $S \leftarrow \{s \in R | \forall l \in L, v(l) \not\subseteq R^{'} \cup \{s\}\}$
18:         $u \leftarrow argmax_{u \in S} f(u, v)$
19:         $L \leftarrow L \cup \{\{u, v\}\}$
20:         Remove $v$ from $V_1$: $V_1 \leftarrow V_1 \backslash \{v\}$
21:     else
22:         $(u, v) \leftarrow argmax_{u \neq v \in V_1} f(u, v)$
23:         $S \leftarrow \{s \in R | \forall l \in L, v(l) \not\subseteq R^{'} \cup \{s\}\}$
24:         $u_1 \leftarrow argmax_{u_1 \in S} f(u_1, u)$
25:         $S \leftarrow \{s \in R | \forall l \in L, v(l) \not\subseteq R^{'} \cup \{u_1, s\}\}$
26:         $v_1 \leftarrow argmax_{v_1 \in S} f(v_1, v)$
27:         if $f(u, v) > f(v, v_1) + f(u, u_1)$ then
28:             $L \leftarrow L \cup \{\{u, v\}\}$
29:         else
30:             $L \leftarrow L \cup \{\{u, u_1\}, \{v, v_1\}\}$
31:             Update $R^{'}$: $R^{'} \leftarrow R^{'} \cup \{u_1, v_1\}$
32:         end if
33:         Remove $u$,$v$ from $V_1$: $V_1 \leftarrow V_1 \backslash \{u, v\}$
34:     end if
35: end while
36: Return $L$.
```

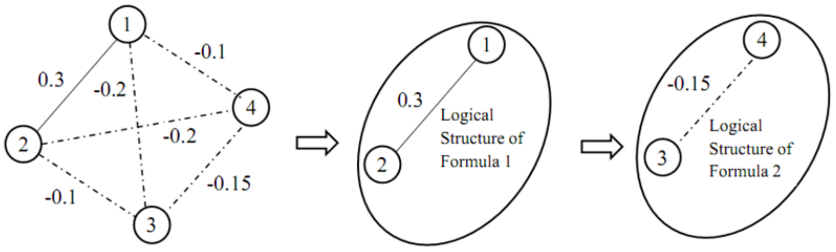


1
2
3
4
5
0.2
0.1
-0.1
-0.2
-0.2
-0.3
-0.3
0.3
0.2
-0.1
Logical Structure of Formula 1
Logical Structure of Formula 2
Logical Structure of Formula3

Figure 6.4: An example of the second case.

$$f(3,4) > f(2,3) + f(1,4). \tag{6.7}$$

Hence, in the next step[8], we have the second logical structure with two node $\{3,4\}$. Note that if $f(3,4) = -0.3$, then $f(3,4) < f(2,3) + f(1,4)$. In the next step[9], we will have two logical structures $\{2,3\}$ and $\{1,4\}$.



1
2
3
4
0.3
-0.2
-0.1
-0.2
-0.1
-0.15
Logical Structure of Formula 1
Logical Structure of Formula 2

Figure 6.5: Another example of the second case.

In the first case, we can easily see that all four constraints are satisfied. In the second case, constraints 1) and 2) are easily satisfied, too. Now we explain how the constraints 3) and 4) are also satisfied. When building logical structures with two nodes, at each step, we choose one node in the set of nodes which have appeared in a logical structure (Set 1) (we call the built logical structure set L), and one node in the set of other nodes (Set 2) so that the edge connecting them has maximum value. For example, in Figure 6.4, two sets of nodes are, $Set1 = \{1,2,3\}$ and $Set2 = \{4,5\}$. However, we only choose one node in a subset $S \subseteq Set1$. At first, we initialize $S = Set1$. After each step, we check each logical structure l in L. If there are k nodes in l, and $k-1$ nodes in l have been chosen in previous steps, the final node in l will be removed from S. By doing this, each logical structure will always contain one node that appears in exactly one logical structure.

[8]In Algorithm 6, code of this step is described in line 28.
[9]In Algorithm 6, code for this case is described in line 30.

An example that illustrates this method is shown in Figure 6.6. First, we consider the graph with positive edges. This graph consists of four nodes $\{1, 2, 3, 4\}$ and two edges $\{(1, 2), (2, 3)\}$. We have the set of zero-degree vertices $V_1 = \{4\}$, and the set of other vertices $V_2 = \{1, 2, 3\}$. The maximal sub-graph of this graph is the graph with two nodes $\{1, 2\}$, so we have the first logical structure with these two nodes. We remove these nodes and the positive edges connecting to these nodes. We initialize[10] $R = \{1, 2\}$, and $R' = \emptyset$. Now we consider Node 3, the remainder node in V_2 which has not been chosen. The set of nodes that can be connect to Node 3 in the next logical structures, $S = R = \{1, 2\}$. Because $f(2, 3) > f(1, 3)$, in the next step we have the second logical structure with two nodes $\{2, 3\}$. After choosing Node 2 to connect to Node 3, we update[11] $R' = R' \cup \{2\} = \{2\}$.

Now we consider the final node, Node 4. The set of nodes that can be connect to Node 4 in the next logical structures, $S = \{2\}$. Note that we remove Node 1 from S because $R' \cup \{1\} = \{1, 2\}$ will contain the first logical structure. Because only Node 2 can connect to Node 4, in the final step we have the third logical structure with two nodes $\{2, 4\}$. Note that if we do not remove Node 1 from S then in the final step we will have the logical structure with two nodes $\{1, 4\}$. Three logical structures $\{1, 2\}$,$\{2, 3\}$, and $\{1, 4\}$ will violate the Constraint 4.

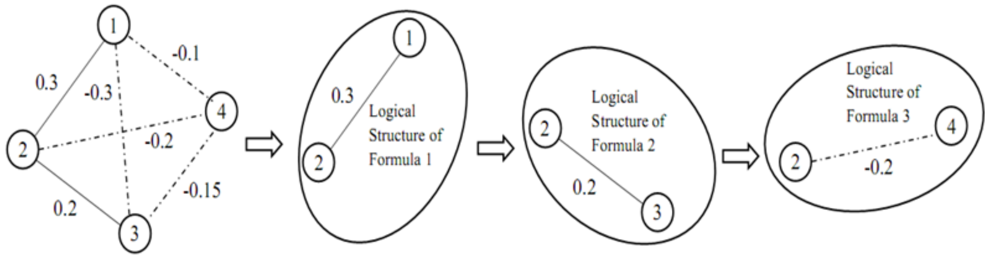

1
2
3
4
0.3
-0.3
-0.1
-0.2
0.2
-0.15
Logical Structure of Formula 1
Logical Structure of Formula 2
Logical Structure of Formula 3
0.3
0.2
-0.2

Figure 6.6: The third example of the second case.

The remaining problem is how to define the value function f. Our solution is that, first we learn a binary classifier C using maximum entropy model. This classifier takes a pair of logical parts as the input, and outputs $+1$ if two logical parts belong to one logical structure, otherwise it will output -1. Then, we define the value function f for two logical parts as follows:

$$f(p_1, p_2) = Prob(C(p_1, p_2) = +1) - 0.5. \tag{6.8}$$

Function f will receive a value from -0.5 to $+0.5$. Function f equals zero in the case the classifier assigns the same probability to $+1$ and -1.

[10]In Algorithm 6, code for this step is described in line 7.

[11]In Algorithm 6, code for this step is described in line 12.

6.3 Experiments

This section describes our annotated corpus, evaluation methods, and experimental results of our solutions.

6.3.1 Corpus and Evaluation Methods

Corpus

Our Japanese National Pension Law (JNPL) corpus consists of 83 legal articles[12], which contain 119 paragraphs with 426 sentences. On average, each paragraph consists of 3.6 sentences. The total number of logical parts is 807, and the number of logical structures is 351. On average, each paragraph consists of 6.8 logical parts and 3 logical structures. Table 6.1 shows some statistics on the number of logical parts of each type.

Table 6.1: Statistics on logical parts of the JNPL corpus

Logical Part	C	A	T_1	T_2	T_3	EL	ER	Ob	RepO	RepR
Number	248	286	0	114	12	55	57	9	12	14

Main types of parts are A(35.4%), C(30.7%), T_2(14.1%), ER(7.1%), and EL(6.8%). Five main types of parts make up more than 94% of all types.

Evaluation Methods

We divided the JNLP corpus into 10 sets, and conducted 10-fold cross-validation tests. For the first sub-task, we evaluated the performance of our system by precision, recall, and F_1 scores as follows[13]:

$$precision = \frac{\#correct\ parts}{\#predicted\ parts}, recall = \frac{\#correct\ parts}{\#actual\ parts}, \tag{6.9}$$

$$F_1 = \frac{2 * precision * recall}{precision + recall}. \tag{6.10}$$

For the second sub-task, we used MUC precision, recall, and F_1 scores as described in [44]. We summarize them here for clarity.

Let $P_1, P_2, \ldots, P_n$ be n predicted logical structures, and $G_1, G_2, \ldots, G_m$ be the correct answers or gold logical structures. To calculate recall, for each gold logical structure

[12]The corpus does not include all the articles of JNPL.

[13]The evaluation methods for Sub-task 1 are similar to evaluation methods for the RRE task.

$G_i (i = 1, 2, \ldots, m)$, let $k(G_i)$ be the smallest number such that there exist $k(G_i)$ predicted structures $P_1^i, P_2^i, \ldots, P_{k(G_i)}^i$ which satisfy $G_i \subseteq \cup_{j=1}^{k(G_i)} P_j^i$:

$$recall = \frac{\sum_{i=1}^{m} (|G_i| - k(G_i))}{\sum_{i=1}^{m} (|G_i| - 1)}. \tag{6.11}$$

To calculate precision, we switch the roles of predicted structures and gold structures. Finally, F_1 score is computed in a similar manner as in the first sub-task.

Table 6.2 shows two examples of the evaluation method for Sub-task 2. In two examples, we have five input logical parts numbered 1,2,3,4, and 5, and the system predicts three logical structures {1,2,3}, {1,4}, and {2,5}. In the first case, the correct answer (gold) consists of two logical structures {1,2,3,4} and {1,5}, while in the second case, the correct answer consists of three logical structures {1,2,3}, {1,5}, and {2,4}.

Table 6.2: Examples of evaluation method for Sub-task 2

Input	Predicted	Gold	Recall	Precision	$F_{\beta=1}$
1,2,3,4,5	{1,2,3} {1,4} {2,5}	{1,2,3,4} {1,5}	$\frac{(4-2)+(2-2)}{(4-1)+(2-1)} = 0.50$	$\frac{(3-1)+(2-1)+(2-2)}{(3-1)+(2-1)+(2-1)} = 0.75$	0.60
1,2,3,4,5	{1,2,3} {1,4} {2,5}	{1,2,3} {1,5} {2,4}	$\frac{(3-1)+(2-2)+(2-2)}{(3-1)+(2-1)+(2-1)} = 0.50$	$\frac{(3-1)+(2-2)+(2-2)}{(3-1)+(2-1)+(2-1)} = 0.50$	0.50

6.3.2 Experiments on Sub-Task 1

Baseline

We chose the Filter-Ranking (FR) Perceptron algorithm proposed by [5, 6] as our baseline model because of its effectiveness on phrase recognition problems, especially on problems that accept the *embedded* relationship[14]. We use FR-perceptron algorithm to recognize logical parts in law sentences one by one in an input paragraph.

The idea of the FR-perceptron algorithm is to build a recognition model with two components. The first component, operating at word level, is a *filtering* function F, which identifies a set of candidate logical parts for an input law sentence s, $F(s) \subseteq P$. The second one, operating at part level, is a *score* function which produces a real value score for a logical part. The recognizer will use the score function to search an optimal coherent subset from the candidate set $F(s)$.

[14]We re-implement the FR-perceptron algorithm by ourself.

$$R(s) = argmax_{y \subseteq F(s) | y \in Y} \sum_{p \in y} score(p, s, y), \tag{6.12}$$

$$score(p, s, y) = W \cdot \phi(p, s, y), \tag{6.13}$$

where $\phi(p, s, y)$ is the feature vector defined on logical part p of sentence s in solution y.

The *filtering* component F is only used to reduce the search space. Instead of searching in the space P, the R function only searches in a subset $F(s)$ of P. The setting for function F is a *begin-end* classification for each logical part category: a word is considered as *c-begin* if it is likely to begin a *category-c* logical part, and as *c-end* if it is likely to end a *category-c* logical part. Each pair of *c-begin* word w_b and *c-end* word w_e forms a logical part candidate (b, e, k, c)[15]. Suppose that h_b^c and h_e^c are begin and end classification functions for each category c, the filtering function F can be described as follows:

$$F(s) = \{(b, e, k, c) | h_b^c(w_b, s, k) = +1 \wedge h_e^c(w_e, s, k) = +1\}. \tag{6.14}$$

For *begin/end* predictors, we get features of words, POS tags, and Bunsetsu tags. We obtained following features in a window size 2: $f[-2]$, $f[-1]$, $f[0]$, $f[+1]$, $f[+2]$ (for words, POS tags, and Bunsetsu tags), $f[-2]f[-1]$, $f[-1]f[0]$, $f[0]f[+1]$, $f[+1]f[+2]$, $f[-2]f[-1]f[0]$, $f[-1]f[0]f[+1]$, $f[0]f[+1]f[+2]$ (for words and POS tags). For example, if f is word feature then $f[0]$ is the current word, $f[-1]$ is the preceding word, and $f[-1]f[0]$ is the co-occurrence of them. Moreover, with *begin* predictor, we use a feature for checking whether this position is the beginning of the sentence or not. Similarly, with *end* predictor, we use a feature for checking whether this position is the end of the sentence or not.

With each candidate of logical part, we extract following kinds of features:

1. Length of the part
2. Internal structure: this feature is the concatenation of the top logical parts, punctuation marks, parenthesis, and quotes inside the candidate. An example about internal structure may be $(A+, +C + .)$ (plus is used to concatenate items). This means that the candidate consists of an antecedent part, a comma, a consequent part, and a period at the end.
3. Uni-gram of words and part-of-speech tags,
4. Bi-gram of words and part-of-speech tags,
5. Tri-gram of words and part-of-speech tags.

[15] k is the sentence position in the input paragraph.

Experimental Results

In our experiments, we focus on paragraphs in Type A, B, and C defined in [39]. In these types, the first sentence is the main sentence, which usually contains more logical parts than other sentences. The other sentences often have a few logical parts, and in most cases these logical parts only appear in one layer. The first sentences usually contain logical parts in two layers.

We divided sentences into two groups. The first group consists of the first sentences in paragraphs, and the second group consists of other sentences. We set the number of layers k to 2 for sentences in the first group, and to 1 for sentences in the second group. To learn sequence labeling models, we used CRFs [23, 21].

Experimental results on the JNPL corpus are described in Table 6.3. We conducted experiments with four feature sets: words; words and POS tags; words and Bunsetsu tags; and words, POS tags, and Bunsetsu tags. To extract features from source sentences, we used the Cabocha tool [20], a Japanese morphological and syntactic analyzer. The best model (word and Bunsetsu tag features) achieved 74.37% in F_1 score. It improves 11.04% in F_1 score (30.11% in error rate) compared with the baseline model.

Table 6.3: Experimental results for Sub-task 1 on the JNLP corpus(W:Word; P: POS tag; B: Bunsetsu tag)

Model	Prec(%)	Recall(%)	F_1(%)
Baseline	**79.70**	52.54	63.33
W	79.18	69.27	73.89
W+P	77.62	68.77	72.93
W+B	79.63	**69.76**	**74.37**
W+P+B	77.89	69.39	73.39

Table 6.4 shows experimental results of our best model in more detail. Our model got good results on most main parts: C(78.98%), A(80.42%), and T_2(82.14%).

6.3.3 Experiments on Sub-Task 2

In our experiment, to learn a maximum entropy binary classification we used the implementation of Tsuruoka [41]. With a pair of logical parts, we extracted the following features (and combinations of them):

- Categories of two parts.
- Levels of two parts.
- The positions of the sentences that contain two parts (the first sentence or not).

Table 6.4: Experimental results in more details

Logical Part	Prec(%)	Recall(%)	F_1(%)
C	**83.41**	**75.00**	**78.98**
EL	76.74	60.00	67.35
ER	41.94	22.81	29.55
Ob	0.00	0.00	0.00
A	**80.42**	**80.42**	**80.42**
RepO	100	16.67	28.57
RepR	100	28.57	44.44
T_2	**83.64**	**80.70**	**82.14**
T_3	60.00	25.00	35.29
Overall	**79.63**	**69.76**	**74.37**

- Categories of other parts in the input paragraph.

We conducted experiments on this sub-task in two settings. In the first setting, we used annotated logical parts (gold inputs) as the inputs to the system. The purpose of this experiment is to evaluate the performance of the graph-based model on Sub-task 2. In the second setting (end-to-end), predicted logical parts outputted by Sub-task 1 were used as the inputs to the system. The purpose of this experiment is to evaluate the performance of our framework on the whole task.

In the second setting, end-to-end setting, because input logical parts may differ from the correct logical parts, we need to modify the MUC scores. Let $P_1, P_2, \ldots, P_n$ be n predicted logical structures, and $G_1, G_2, \ldots, G_m$ be the gold logical structures. For each gold logical structure $G_i (i = 1, 2, \ldots, m)$, let D_i be the set of logical parts in G_i which are not included in the set of input logical parts. $D_i = \{p \in G_i | p \notin \cup_{j=1}^{n} P_j\}$. Let $k(G_i)$ be the smallest number such that there exist $k(G_i)$ predicted structures $P_1^i, P_2^i, \ldots, P_{k(G_i)}^i$ which satisfy $G_i \subseteq (\cup_{j=1}^{k(G_i)} P_j^i) \cup D_i$.

$$recall = \frac{\sum_{i=1}^{m} (|G_i| - |D_i| - k(G_i))}{\sum_{i=1}^{m} (|G_i| - 1)}. \tag{6.15}$$

To calculate the precision, we switch the roles of predicted structures and gold structures.

Two examples of evaluation method for Sub-task 2 in the end-to-end setting are shown in Table 6.5. In the first case, the input (output predicted by our system in Sub-task 1) consists of three logical parts 1,2, and 3, while the actual input (gold input) consists of four logical parts 1,2,3, and 4. Logical part 4 is not included in the set of input logical parts. Therefore, when calculating recall, we need to subtract 1 from each factor in the numerator. In the second case, the input is unchanged, while the gold input consists of four logical parts 1,2, and 4. The input does not contain logical part 4, and the gold input does not contain logical part 3. Hence, we need to subtract 1 from each factor in the numerator when calculating both recall and precision.

Table 6.5: Examples of evaluation method for Sub-task 2 in the end-to-end setting (Pre = Predicted, GIn = Gold Input, GRes = Gold Result)

Input	Pre	GIn	GRes	Recall	Precision	$F_{\beta=1}$
1,2,3	{1,2} {1,3}	1,2,3,4	{1,2,4} {3,4}	$\frac{(3-1-1)+(2-1-1)}{(3-1)+(2-1)}$=0.33	$\frac{(2-0-1)+(2-0-2)}{(2-1)+(2-1)} = 0.50$	0.40
1,2,3	{1,3} {2,3}	1,2,4	{1,4} {2,4}	$\frac{(2-1-1)+(2-1-1)}{(2-1)+(2-1)}$=0.00	$\frac{(2-1-1)+(2-1-1)}{(2-1)+(2-1)} = 0.00$	0.00

Table 6.6 shows experimental results on the second sub-task. When using gold inputs, our model achieved 75.89% in MUC F_1 score. However, when using outputs of the first sub-task as the input, we only got 51.12%. This is reasonable, because errors accumulate in two sub-tasks.

Table 6.6: Experimental results on Sub-task 2

Setting	Prec(%)	Recall(%)	F1(%)
Gold Inputs	81.24	71.19	75.89
End-to-End	54.88	47.84	51.12

6.4 Conclusions

We presented the RREP task, in which we consider legal paragraphs consisting of multiple sentences. We proposed a two-phase framework to solve the task. In the first phase, we provided a multi-layer sequence learning model to recognize logical parts. We divided logical parts in a law sentence into some layers, and modeled the task of recognizing logical parts in each layer as a sequence learning problem. In the second phase, we introduced a graph-based model to recognize logical structures. Our experimental results provides a baseline for futher researches on this interesting task.

Chapter 7

Conclusions

7.1 Summary of the Thesis

In this thesis, we introduced two tasks in Legal Engineering: (1) *Recognition of Requisite Part and Effectuation Part in Law Sentences*, or RRE task; and (2) *Recognition of Requisite Parts and Effectuation Parts in Paragraphs Consisting of Multiple Sentences*, or RREP task. For the RRE task, the goal is to recognize logical parts given a law sentence. For the RREP task, given a paragraph in a legal article, the goal is to recognize logical parts in law sentences and group related logical parts into some logical structures. We also introduced a corpus of real legal data for these tasks, Japanese National Pension Law corpus.

We presented a study on the RRE task in some aspects:

1. *Linguistic features.* We found that word features are the most important features to the RRE task. Features other than word features and part-of-speech features are not effective to the task. We also described an exploring on contributions of words to the RRE task, in which we found that statistical machine learning models use the same words as human do when recognizing logical parts in law sentences.

2. *Problem modeling.* We showed that modeling based on Bunsetsus is better than modeling based on words. On the model based on words, using only head words and functional words gives better results than using all words.

3. *Tag setting.* We found that among four kinds of tag settings, IOB, IOE, FIL, and FILC, IOE (the last element of a part is tagged with E, the other elements are tagged with I, elements outside every part are tagged with O) is the most suitable tag setting for the RRE task.

4. *Semi-supervised learning.* We showed that by exploiting unlabeled data in a simple manner (use extra word features derived from Brown clusters), we can improve the

results of the RRE task. The unlabeled data are more helpful when the amount of labeled data is small.

For the RRE task, Our best model achieved 88.84% in $F_{\beta=1}$ score on the Japanese National Pension Law corpus.

We also proposed a two-phase framework to solve the RREP task:

1. Sub-task 1: *Recognition of Logical Parts.* We divided logical parts in a law sentence into some layers. The first layer consists of logical parts that are not embedded in any other parts. Suppose that we remove logical parts in the first layer, the second layer will consist of logical parts that are not embedded in any other parts. The higher layers are similarly defined. We proposed a multi-layer sequence learning model, in which we trained a CRFs model to recognize logical parts in each layer.
2. Sub-task 2: *Recognition of Logical Structures.* We considered this task as a problem of searching a set of sub-graphs in a complete weighted graph with some constraints. In this graph, each vertex corresponds to a logical part and a complete sub-graph (or a set of vertices) corresponds to a logical structures. The weight on an edge indicates the degree in which two vertices belongs to one logical structure. We proposed a heuristic algorithm to solve the problem. The main idea of the algorithm is to take as many positive edges as possible, and as few negative edges as possible.

Our models achieved 74.37% in recognizing logical parts, 75.89% in recognizing logical structures, and 51.12% in the whole task on the Japanese National Pension Law corpus. Our results provide a baseline for further researches on this interesting task.

7.2 Future Work

In our future work, we will continue to investigate these two tasks:

1. **RRE task**

   In this thesis, we have investigated the RRE task using Markov models (CRFs). In the future, we will compare Markov and semi-Markov models (semi-CRFs [36]) on the RRE task. Some studies show that sometimes semi-Markov models can improve performance over Markov models [24, 36].

2. **RREP task**

   In this thesis, we considered this task as two separate sub-tasks. This means that the process of recognizing logical parts and the process of recognizing logical structures are independent. The information of one process, however, may support the other process, and vice versa.

In the future, we will try to integrate two sub-tasks into a unified process, where we recognize both logical parts and logical structures at the same time.

From the results of these two tasks, we also investigate the task of *Translating Legal Articles into Logical and Formal Representations*, where the input is a set of documents and the outputs are their formal representations.

Publications

The works in this thesis have been published (accepted to publish) in the following papers:

- **Journal Papers**

  1. **Ngo Xuan Bach**, Nguyen Le Minh, Akira Shimazu. RRE Task: The Task of Recognition of Requisite Part and Effectuation Part in Law Sentences. Accepted to publish in *International Journal of Computer Processing Of Languages (IJCPOL)*, 2011.

- **Papers in International Conferences and Workshops**

  1. **Ngo Xuan Bach**, Nguyen Le Minh, Akira Shimazu. Recognition of Requisite Part and Effectuation Part in Law Sentences. In *Proceedings of the* 23rd *International Conference on the Computer Processing of Oriental Languages (ICCPOL)*, pp. 29-34, 2010.
  2. Le Minh Nguyen, **Xuan Bach Ngo**, Viet Cuong Nguyen, Quang Nhat Minh Pham and Akira Shimazu. A Semi-Supervised Learning Method for Vietnamese Part of Speech Tagging[1]. In *Proceedings of the* 2nd *International Conference on Knowledge and Systems Engineering (KSE)*, pp. 141-146, 2010.
  3. **Ngo Xuan Bach**, Nguyen Le Minh, Akira Shimazu. Exploring Contributions of Words to Recognition of Requisite Part and Effectuation Part in Law Sentences. In *Proceedings of the* 4th *International Workshop on Juris-Informatics (JURISIN)*, pp. 121-132, 2010.

- **Papers in Domestic Conferences**

  1. **Ngo Xuan Bach**, Nguyen Le Minh, Akira Shimazu. Recognition of Requisite Part and Effectuation Part in Law Sentences. In *Proceedings of* 16th *Annual Meeting of Association for Natural Language Processing*, pp. 35-38, 2010.

[1]This work does not relate to the thesis. It only uses the same semi-supervised technique as the technique we do in Chapter 5.

Bibliography

[1] R.K. Ando, T. Zhang. A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. In *Journal of Machine Learning Research*, Volume 6, pp.1817-1853, 2005.

[2] A. Blum, T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Workshop on Computational Learning Theory*, 1998.

[3] P.F. Brown, P.V. deSouza, R.L. Mercer, V.J.D. Pietra, J.C. Lai. Class-Based n-gram Models of Natural Language. In *Computational Linguistics*, Volume 18, Issue 4, pp.467-479, 1992.

[4] R.H. Byrd, J. Nocedal, R.B. Schnabel. Representations of Quasi-Newton Matrices and their use in Limited Memory Methods. In *Mathematical Programming*, Volume 63, Issue 4, pp.129-156, 1994.

[5] X. Carreras, L. Màrquez, J. Castro. Filtering-Ranking Perceptron Learning for Partial Parsing. In *Machine Learning*, Volume 60, pp.41-71, 2005.

[6] X. Carreras, L. Màrquez, V. Punyakanok, D. Roth. Learning and Inference for Clause Identification. In *Proceedings of ECML*, pp. 35-47, 2002.

[7] M. Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of EMNLP*, pp.1-8, 2002.

[8] M. Collins, T. Koo. Discriminative Reranking for Natural Language Parsing. In *Computational Linguistics*, Volume 31, Issue 1, pp.25-70, 2005.

[9] R. Collobert, J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of ICML*, pp.160-167, 2008.

[10] E. Ejerhed. Finding clauses in unrestricted text by finitary and stochastic methods. In *Proceedings of the 2nd Conference on Applied Natural Language Processing*, pp.219-227, 1988.

[11] G.D.Jr. Forney. The viterbi algorithm. In *Proceedings of the IEEE*, Volume 61, Issue 3, pp.268-278, 1973.

[12] G. Haffari, A. Sarkar. Analysis of semi-supervised learning with the Yarowsky algorithm. In *Proceedings of UAI*, pp.159-166, 2007.

[13] A. Haghighi, D. Klein. Simple Coreference Resolution with Rich Syntactic and Semantic Features. In *Proceedings of EMNLP*, pp.1152-1161, 2009.

[14] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of ICML*, pp.200-209, 1999.

[15] T. Katayama. The current status of the art of the 21st COE programs in the information sciences field. Verifiable and evolvable e-society - realization of trustworthy e-society by computer science - (in Japanese). In *IPSJ (Information Processing Society of Japan) Journal*, 46(5), pp.515-521, 2005.

[16] T. Katayama. Legal engineering - an engineering approach to laws in e-society age. In *Proceedings of the 1st International Workshop on JURISIN*, 2007.

[17] T. Katayama, A. Shimazu, S. Tojo, K. Futatsugi, K. Ochimizu. e-Society and Legal Engineering (in Japanese). In *Journal of the Japanese Society for Artificial Intelligence*, 23(4), pp.529-536, 2008.

[18] Y. Kimura, M. Nakamura, A. Shimazu. Treatment of Legal Sentences Including Itemized and Referential Expressions - Towards Translation into Logical Forms. In *New Frontiers in Artifical Intelligence*, volume 5447 of LNAI, pp.242-253.

[19] T. Koo, X. Carreras, M. Collins. Simple semi-supervised dependency parsing. In *Proceedings of ACL*, pp.595-603, 2008.

[20] T. Kudo. Yet Another Japanese Dependency Structure Analyzer. http://chasen.org/ taku/software/cabocha/

[21] T. Kudo. CRF++: Yet Another CRF toolkit. http://crfpp.sourceforge.net/

[22] T. Kudo, K. Yamamoto, Y. Matsumoto. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of EMNLP*, pp.230-237, 2004.

[23] J. Lafferty, A. McCallum, F. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of ICML*, pp.282-289, 2001.

[24] P. Liang. Semi-Supervised Learning for Natural Language. *Master's thesis, Massachusetts Institute of Technology*, 2005.

[25] A. McCallum, D. Freitag, F. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of ICML*, pp.591-598, 2000.

[26] D. Lüdtke, S. Sato. Fast Base NP Chunking with Decision Trees-Experiments on Different POS Tag Settings. In *Proceedings of CICLing*, pp.139-150, 2003.

[27] A. Mnih, Y. Bengio. A scalable hierarchical distributed language model. In *Proceedings of NIPS*, pp.1081-1088, 2009.

[28] M. Murata, K. Uchimoto, Q. Ma, H. Isahara. Bunsetsu identification using category-exclusive rules. In *Proceedings of COLING*, pp.565-571, 2000.

[29] D. Nadeau, S. Sekine. A survey of named entity recognition and classification. In *Lingvisticae Investigationes*, Volume 30, Issue 1, pp.3-26, 2007.

[30] M. Nakamura, S. Nobuoka, A. Shimazu. Towards Translation of Legal Sentences into Logical Forms. In *Proceedings of the 1st International Workshop on JURISIN*, 2007.

[31] V. Ng. Supervised Noun Phrase Coreference Research: The First Fifteen Years. In *Proceedings of ACL*, pp. 1396-1411, 2010.

[32] F. Peng, A. McCallum. Information extraction from research papers using conditional random fields. In *Information Proceesing and Management*,Volume 42, Issue 4, pp.963-979, 2006.

[33] E.T.K. Sang, S. Buchholz. Introduction to the CoNLL-2000 Shared Task: Chunking. In *Proceedings of CoNLL*, pp.127-132, 2000.

[34] E.T.K. Sang, H. Dějean. Introduction to the CoNLL-2001 Shared Task: Clause Identification. In *Proceedings of CoNLL*, pp.53-57, 2001.

[35] E.T.K. Sang. Introduction to the CoNLL-2002 Shared Task: language-independent named entity recognition. In *Proceedings of CoNLL*, pp.1-4, 2002.

[36] S. Sarawagi, W. Cohen. Semi-Markov Conditional Random Fields for Information Extraction. In *Proceedings of NIPS*, pp.1185–1192, 2004.

[37] F. Sha, F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of NAACL*, pp.213-220, 2003.

[38] C. Sutton, A. McCallum. An Introduction to Conditional Random Fields for Relational Learning. In *Introduction to Statistical Relational Learning*, Chapter 4, MIT Press, 2006.

[39] K. Takano, M. Nakamura, Y. Oyama, A. Shimazu. Semantic Analysis of Paragraphs Consisting of Multiple Sentences - Towards Development of a Logical Formulation System. In *Proceedings of JURIX,* pp. 117-126, 2010.

[40] K. Tanaka, I. Kawazoe, H. Narita. Standard structure of legal provisions - for the legal knowledge processing by natural language. In *IPSJ Research Report on Natural Language Processing*, pp.79-86, 1993.

[41] Y. Tsuruoka. A simple C++ library for maximum entropy classification. *http://www-tsujii.is.s.u-tokyo.ac.jp/ tsuruoka/maxent/.*

[42] J. Turian, L. Ratinov, Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*, pp.384-394, 2010.

[43] V.N. Vapnik. Statistical learning theory. New York: Wiley, pp.339-371, 1998.

[44] M. Vilain, et al. A Model-Theoretic Coreference Scoring Scheme. In *Proceedings of MUC-6*, pp.45-52, 1995.

[45] H.M. Wallach. Conditional Random Fields: An Introduction. University of Pennsylvania CIS Technical Report MS-CIS-04-21.

[46] H. Zhao, W. Chen, C. Kit, G. Zhou. Multilingual dependency learning: a huge feature engineering method to semantic dependency parsing. In *Proceedings of CoNLL*, pp.55-60, 2009.

[47] X. Zhu. Semi-Supervised Learning Literature Survey. Computer Sciences TR 1530 University of Wisconsin-Madison, 2008.