SYNTAX-DRIVEN ARGUMENT IDENTIFICATION AND MULTI-ARGUMENT CLASSIFICATION FOR SEMANTIC ROLE LABELING

CHI-SAN ALTHON LIN

This thesis is submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science at the University of Waikato.

September 2007

Abstract

Semantic role labeling is an important stage in systems for Natural Language Understanding. The basic problem is one of identifying who did what to whom for each predicate in a sentence. Thus labeling is a two-step process: identify constituent phrases that are arguments to a predicate, then label those arguments with appropriate thematic roles.

Existing systems for semantic role labeling use machine learning methods to assign roles one-at-a-time to candidate arguments. There are several drawbacks to this general approach. First, more than one candidate can be assigned the same role, which is undesirable. Second, the search for each candidate argument is exponential with respect to the number of words in the sentence. Third, single-role assignment cannot take advantage of dependencies known to exist between semantic roles of predicate arguments, such as their relative juxtaposition. And fourth, execution times for existing algorithm are excessive, making them unsuitable for real-time use.

This thesis seeks to obviate these problems by approaching semantic role labeling as a multi-argument classification process. It observes that the only valid arguments to a predicate are unembedded constituent phrases that do not overlap that predicate. Given that semantic role labeling occurs after parsing, this thesis proposes an algorithm that systematically traverses the parse tree when looking for arguments, thereby eliminating the vast majority of impossible candidates.

Moreover, instead of assigning semantic roles one at a time, an algorithm is proposed to assign all labels simultaneously; leveraging dependencies between roles and eliminating the problem of duplicate assignment.

Experimental results are provided as evidence to show that a combination of the proposed argument identification and multi-argument classification algorithms outperforms all existing systems that use the same syntactic information.

Acknowledgements

I would like to express my gratitude to all those who supported me during writing this thesis. I want to thank the Department of Computer Science at Waikato University for funding support and for the departmental resources that made the research work possible.

Sincerest thanks and appreciation go to my supervisor, Dr. Tony Smith, whose support and guidance helped make this thesis (and many other things) possible, and to my co-supervisors, Prof. Ian Witten and Dr. Eibe Frank, who provided many useful ideas throughout the period of this research.

I would also like to thank my family, relatives, and friends for their concerns and caring during the preparation of my thesis. Most of all, I would like to thank my parents, parents-in-law and wife for their immeasurable encouragement and support in the completion of this work; especially for the company, inspiration, and growth they bring to my world. This thesis is dedicated to them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Natural Language Processing (NLP) involves the general task of finding the structure of text at morphologic, syntactic and semantic levels (Jurafsky & Martin, 2000). "Broadly speaking, analyzing a sentence consists of segmenting it into words, recognizing syntactic elements and their relationships within the structure, and inferring a syntactico-semantic representation of the many concepts the sentence may express" (Carreras, 2005, p.11). Several fundamental tasks in NLP are based on recognizing phrases or constituents of some type. Examples of these tasks include Noun Phrase Bracketing, Chunking, Clause Identification, Named Entity Identification, Syntactic Analysis, Natural Language Parsing and Semantic Role Labeling. The main differences between these tasks concern the nature of the phrases and the relations they exhibit in a sentence. Semantic Role Labeling (SRL) is, fundamentally, the problem of discovering who did what to whom (and possibly with what, where and when) in a sentence. It is an essential step toward the ultimate goal of Natural Language Understanding, and is the subject of study in this thesis.

Generally speaking, Semantic Role Labeling (SRL) is a two step process. Firstly, all arguments must be identified with exact word spans, which is the process of identifying all arguments for each predicate in a sentence (argument identification). Secondly, these arguments must be labelled with correct semantic roles, which is the process of assigning to them corresponding semantic role labels (argument classification). Existing systems label each argument of a predicate one-at-a-time, and thus cannot take advantage of real semantic dependencies that exist between different roles. This thesis claims that more accurate assignment will be achieved if such dependencies are exploited for argument classification.

To test this theory, a Multi-Argument Classification (MAC) technique for semantic role assignment is proposed. This technique labels a group of arguments simultaneously using a statistical pattern-matching (PM) algorithm and uses existing Singular-Argument Classification techniques to complete the assignment only when the database does not contain a fully-matching pattern. In addition, this thesis also proposes a new tree-based pre-processor algorithm, Predicate Argument Recognition Algorithm (PARA), for argument identification to forward unlabelled arguments to classifiers. Generally speaking, valid arguments are non-overlapping and not embedded within each other. State-of-the-art syntactic parsers such as Collins (1999) or Charniak (2000) already solve the overlapping problem and their output provides an ideal structure for finding arguments. The remaining problem is to select valid semantic arguments from these non-overlapping constituents of parse trees obtained by a parser. Cursory examination of hand-corrected parses reveals that upper-most nodes that do not include predicates are all valid arguments. From this observation, PARA is proposed based on a hypothesis that the upper-most ancestor nodes in the parse tree that do not include predicates are the only potentially valid arguments and need not be rediscovered during argument identification.

Experimental results are provided as evidence in support of these claims. They show that the combination of PARA and PM using very basic features not only outperforms systems that use the same syntactic information in the CoNLL 2005 shared task (Carreras and Marquez, 2005) but offers competitive performance compared to the best system that uses rich features and rich syntactic information. Therefore this thesis concludes that the proposed Multi-Argument Classification technique provides a more effective way to assign semantic role labels.

1.1 Motivation

Since 1999, the Conference on Natural Language Learning (CoNLL) has organized several shared tasks exploring many NLP problems. Each edition poses a natural language problem to be solved with Machine Learning (ML) techniques, with the aim of comparing different learning approaches in a common problem setting. For a given problem, training and test data derived from existing corpora are prepared and made public for system development. Systems can then be compared in terms of their performance based on standard evaluation measures computed on the test set. These shared tasks include Noun Phrase (NP) Bracketing in 1999, Chunking in 2000, Clause Identification in 2001, Name Entity Recognition in 2002 and 2003, Semantic Role Labeling in 2004, and 2005, and Multi-lingual Dependency Parsing in 2006. This thesis focuses on the problem studied in the CoNLL 2004 and 2005 shared tasks, namely Semantic Role Labeling (SRL).

1.1.1 Semantic Role Labeling

The task of Semantic Role Labeling is to find the arguments for all predicates (or verbs) in a sentence, and classify them according to semantic roles such as AGENT, THEME, and TIME. AGENT is the argument having control of the action expressed by the predicate, THEME is the participant affected by the action of the predicate and TIME is when the action happens. AGENT, and THEME are more usually assigned abstract labels such as A0 and A1 in the CoNLL shared tasks respectively.

As articulated in the CoNLL 2004 and 2005 shared tasks:

> A semantic role in language is the relationship that a syntactic constituent has with a predicate. Typical semantic arguments include Agent, Patient, Instrument, etc. and also adjunctive arguments indicating Locative, Temporal, Manner, Cause, etc. aspects. Recognizing and labeling semantic arguments is a key task for answering "Who", "When", "What", "Where", "Why", etc. questions in Information Extraction, Question Answering, Summarization, and, in general, in all NLP tasks in which some kind of semantic interpretation is needed.[1]

[1] http://www.lsi.upc.edu/~srlconll/

Consider the following example sentence, annotated with semantic roles from PropBank[2]:

[$_{\text{A0}}$ He] [$_{\text{AM-MOD}}$ would] [$_{\text{AM-NEG}}$ n't] [$_{\text{V}}$ accept] [$_{\text{A1}}$ anything of value] [$_{\text{A2}}$ from those he was writing about] .

The roles for the predicate, "accept" and its related arguments are defined in the PropBank Frames scheme as:

V: verb	A0: acceptor	A1: thing accepted
A2: accepted-from	A3: attribute	AM-MOD: modal
AM-NEG: negation		

An example of a syntactic tree for the sentence "The officer came to his office" and its semantic roles is shown in Figure 1.1. "The officer" is the first argument and fulfills the role of Agent (or A0 in the PropBank annotation), for the predicate "came", and "to his office" a local adjunct, AM-LOC, that includes a preposition, "to". A system for SRL must find all the arguments for the predicates in each sentence and assign semantic roles to each argument.

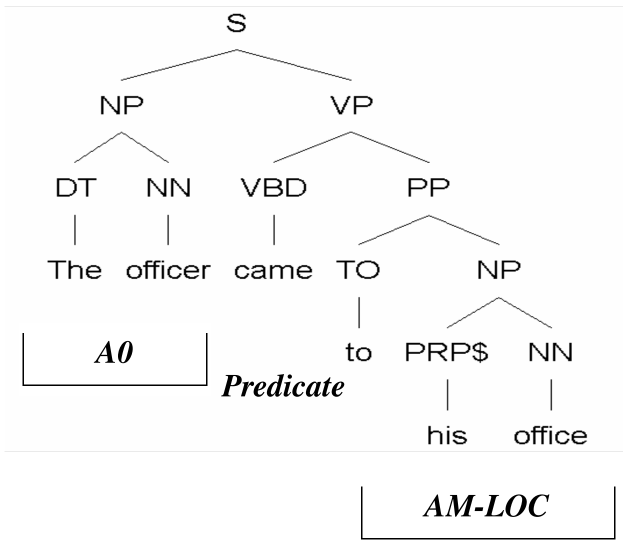


Figure 1.1. An example of a syntactic tree.

[2] http://www.cis.upenn.edu/~ace/

1.1.2 Research Aspects

The CoNLL shared tasks for SRL have drawn a multitude of different approaches and techniques, such as SRL with partial syntactic information in 2004 (Carreras and Marquez, 2004), and with full parses in 2005 (Carreras and Marquez, 2005). This thesis proposes two novel methods to solve this problem:

i) a syntax-driven pre-processor for identifying unlabelled semantic arguments, and
ii) a Multi-Argument Classification technique instead of the commonly used Singular-Argument Classification.

The first issue is argument identification. Traditional argument recognizers have to spend time on each phrase and word to find all possible semantic arguments. For example, in Figure 1.1, there are five phrases (i.e. internal nodes), and six words (i.e. leaves). Each phrase or word is considered an argument candidate. During argument identification, these argument candidates must be verified as valid arguments typically via a ML approach (such as the statistical estimation by Palmer et al., 2005), which considers a lot of unnecessary invalid nodes.

In order to reduce computational time, Xue and Palmer (2004) describe a pruning strategy as a pre-processing step to filter out constituents that are clearly not semantic arguments to the predicate. Briefly, their pruning strategy keeps as argument candidates the upper-most ancestor nodes and their children if the ancestor nodes are PP that do not include predicates. For example, in Figure 1.1, the upper-most ancestor nodes not including the predicate, "come", are the NP node, "The officer", the PP node, "to his office", and the PP child node "his office". Such a pruning strategy, which is widely used in Semantic Role Labelling systems helps reduce a number of incorrect arguments, more than 60% as articulated in Tsai et al. (2005).

Cursory examination of hand-corrected sentences reveals that all valid arguments are upper-most nodes in the parse tree. This makes sense when one realises that the process of parsing eliminates the possibility of overlapping arguments. Moreover, if argument identification proceeds top-down from the parse instead of bottom-up from the words, the problem of overlapping arguments goes away. Therefore, this thesis hypothesizes that such upper-most ancestor nodes that do not include predicates are valid candidates of semantic arguments and do not need validation by ML estimations. The two arguments, NP and PP, in Figure 1.1, are examples of such candidates. By this hypothesis, this thesis proposes a deterministic algorithm as a pre-processing step for argument identification. It can directly map syntactic parses to unlabeled semantic (or predicate) arguments without verifying each candidate derived from their pre-processing method for better speed performance.

The second issue is how to classify arguments. Current systems on SRL are based on Singular-Argument Classification, which labels semantic roles for arguments one by one. These Singular-Argument Classification techniques do not make use of the constraint of role dependency when classifying arguments. As an alternative methodology, this thesis introduces a Multi-Argument Classification technique to achieve better performance. Unlike other systems that use Singular-Argument Classification in the CoNLL 2005 shared task, the Multi-Argument Classification technique is a group and memory-based approach, which can assign semantic roles for several argument candidates of a predicate at the same time. The underlying approach of Multi-Argument Classification is to try to find all possible patterns for a test example from training data then select the best one according to some probability estimation, and uses Singular-Argument Classification to complete the assignment only when the database does not contain a fully-matching pattern.

This thesis focuses on exploring and developing a framework to demonstrate solutions to SRL via the pre-processor for argument identification and the Multi-Argument Classification technique for argument classification.

1.2 Thesis Statement

Semantic Role Labeling is a conversion between syntactic structure and semantic structure, and includes two parts: converting syntactic structure to unlabeled semantic structure, and finishing the construction of semantic structure by filling this unlabeled structure with semantic roles. This thesis demonstrates improved accuracy of argument classification by constructing a model of Multi-Argument Classification and Singular-Argument Classification to associate unlabelled semantic structure with semantic roles. Such unlabelled semantic structure can be identified via either a ML argument recogniser or a pre-processor to map syntactic structure to unlabelled semantic structure that is based on finding upper-most ancestor nodes in tree-based parses.

1.2.1 Syntactic Structure and Semantic Structure

Moschitti and Bejan (2004) state

> “there is no linguistic theory that supports the selection of syntactic features to recognize semantic structures. As a consequence, researchers are still trying to extend the basic features with other ones to improve the flat feature space” (Moschitti and Bejan, 2004, p. 18).

Despite the lack of an explicit theory to prove that syntactic structure can be directly converted into semantic structure, the problem must still be solved. Computational linguists like Gildea and Jurafsky (2002) propose feature-based systems to solve this conversion task. These features (including basic ones such as phrase type, head word of a phrase, and so on,) found in syntactic structure are forwarded to a learning system and Semantic Role Labeling is inferred by finding similarity between test data and training data through various techniques.

The aim of computational research in Semantic Role Labeling is to design a system that exploits the similarity between syntactic structure in a test case and a training example according to probability estimation of syntactic features. Two structures can be an exact match, partially the same, or not related. Computational linguistics focuses on Machine Learning approaches to measure similarity and to infer a solution for the semantic roles even though there is no substantial linguistic theory yet to validate this process.

Current research has demonstrated the feasibility of converting from syntactic to semantic structure. This thesis

i) demonstrates that it is possible to recognize unlabelled semantic arguments deterministically in syntactic structure, and

ii) provides a technique to label all unlabelled semantic arguments simultaneously with semantic roles by utilizing role dependencies.

1.2.2 Techniques for the conversion

This thesis develops techniques for argument identification and argument classification using a combination of heuristic algorithms and machine learning approaches to achieve faster and more accurate Semantic Role Labeling than existing systems provide.

Heuristic algorithms are handcrafted rules that help to offer general solutions to a system. These heuristic algorithms are introduced mainly to mitigate any problems of data sparseness.

This thesis proposes heuristic algorithms of two types:

1) a general algorithm for argument identification (PARA); and
2) some pre-processing and post-processing rules for SRL.

Machine Learning techniques, adopted when the heuristic algorithms do not suit each individual case or do not perform well, serve to predict which phrases of a sentence form arguments for predicates, and which labels should be assigned to them. The Machine Learning techniques studied in this thesis include:

1) an improved statistical approach based on Singular-Argument Classification (SAC) as a baseline system; and

2) a new technique, Pattern-Matching (PM) for Multi-Argument Classification (MAC), which utilizes role relationships in the semantic structures (such as is found in a semantic list [A0 V A1] as shown in Figures 1.2 and 1.3).

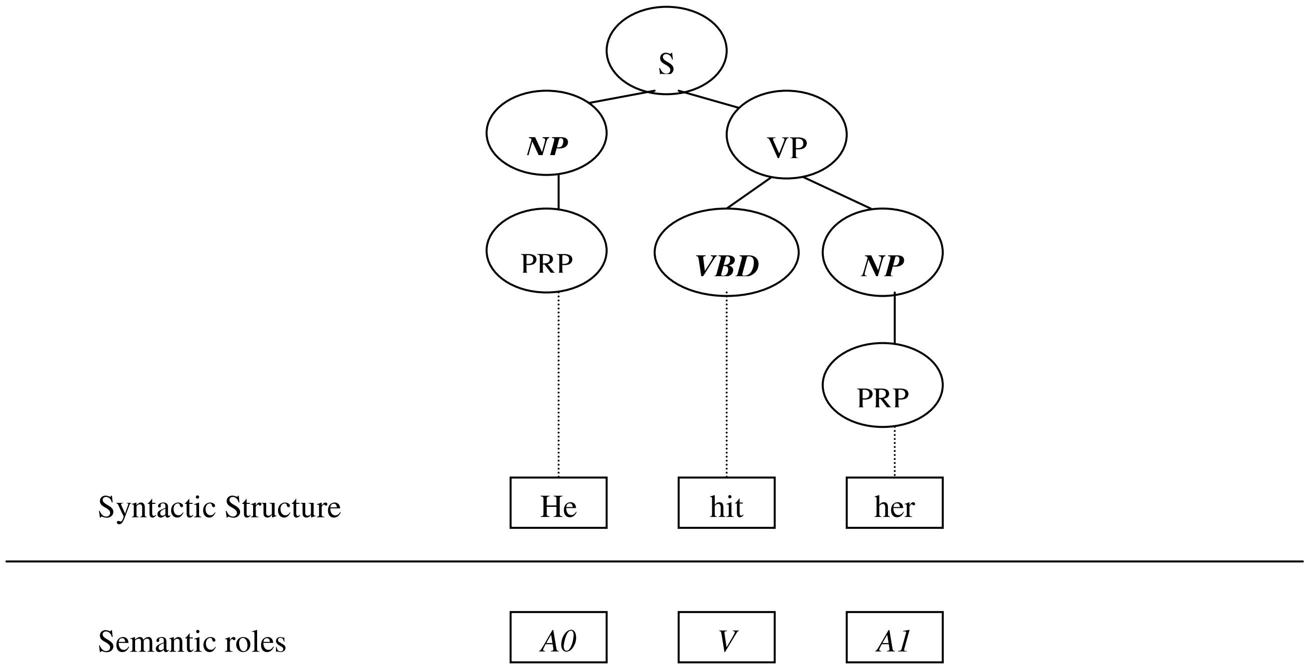


Figure 1.2. An example of a trained sentence with semantic roles.

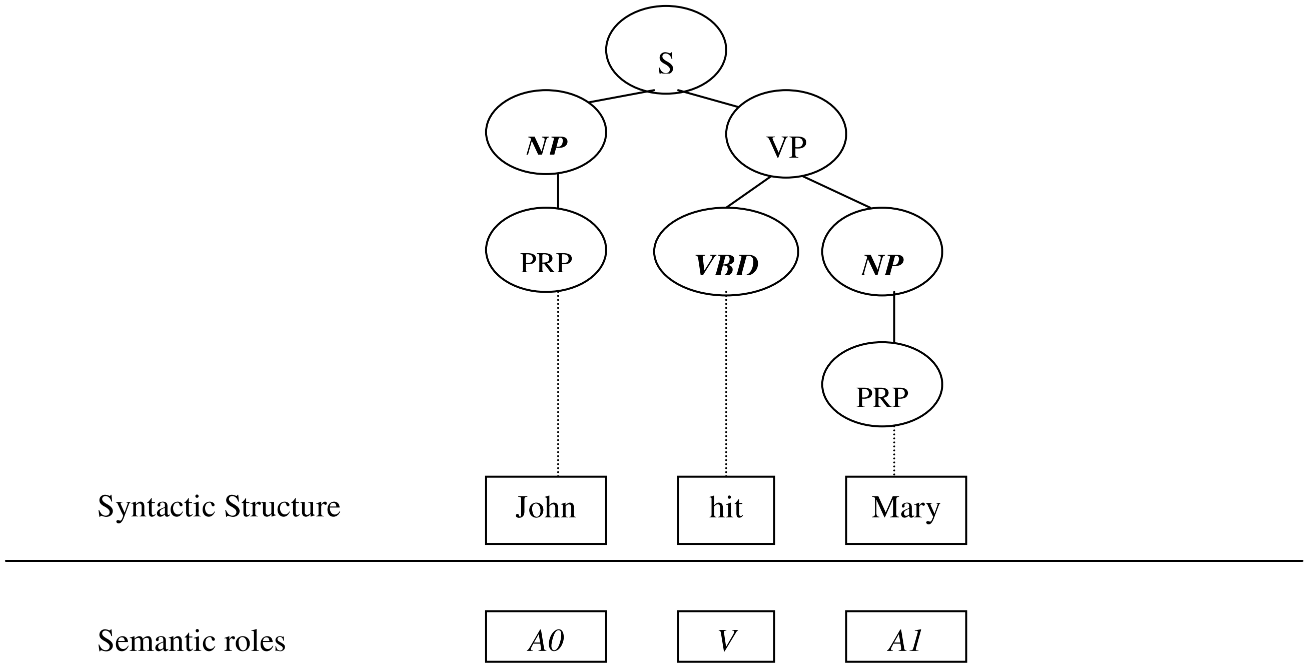


Figure 1.3. An example of a test sentence.

The new technique is predicated on the ability to exploit relationships between roles in a predicate-argument list or pattern. There exists a dependency relationship in a predicate-argument pattern such as [A0 V A1], [A1 V] and so on. Such a relationship in a predicate-argument structure is called ***semantic role dependency***. For example, in Figure 1.2, the predicate-argument structure [A0 V A1] is a pattern with two roles "A0" and "A1" positioned before and after the predicate "V" respectively, where A0 must precede A1 on either side of the predicate, and each role must be unique.

Existing argument classifiers do not make use of this semantic role dependency in their classification stage due to their limitation of singular argument classification. Therefore, they need to include some post-processor step to avoid duplicate typical roles or core roles (like A0 or A1). The proposed PM, utilizing the constraint of semantic role dependency, helps to boost accuracy performance in the classification stage compared to singular-argument classifiers (like Palmers et al, 2005).

This thesis outlines discriminative strategies for learning the predictors of argument identification and argument classification, focuses on differences between heuristic algorithms and machine learning techniques, and incorporates advantages of different models and systems.

Techniques of machine learning techniques and heuristic algorithms in this thesis are designed with two main concerns: ***accuracy*** and ***speed.***

The first issue, ***accuracy***, concerns the performance of the system when finding semantic relations that different phrases exhibit in a structure. In a sentence, phrases group words that together represent a linguistic element of some nature. For example, the sentence "the dog bites a tennis ball" can be segmented into three basic syntactic phrases: a noun phrase (the dog), a verb (bites), and another noun phrase (a tennis ball). Semantic Role Labeling must find constituents or groups of words related to predicates and classify these constituents with correct labels (for example, the semantic roles defined in PropBank). In the syntactic domain, it is common that a noun phrase is followed by a verb phrase. The former is usually the agent of the predicate expressed in the verb phrase, as “the dog” in “the dog bites a tennis ball”. When there is an Agent in a semantic structure, there will not be another Agent in the same semantic structure. One of the advantages of the methods proposed in this thesis is that they can utilize this role dependency as a useful constraint and help to increase labelling accuracy. Chapter 5 details this technique (Multi-Argument Classification) and shows the experimental results.

The second issue is ***speed***. Structurally, a sentence is a sequence of words. The number of possible phrases in a sentence grows quadratically with the length of the sentence, and the space of possible phrase structures is of exponential size (Carreras, 2005). For example, in Figure 1.2, the number of words in the training sentence is three, but there are five nodes of possible phrases in the syntactic tree. These five nodes are S, NP (before the predicate), VP, VBD, and NP (after the predicate). The space of possible phrase structures can be all combinations of any words in the sentence, which form a set of {“He”, “hit”, “her”, “He hit”, “hit her”, “He hit her”, and none of them}. If SRL is based on words, which is a bottom-up search, a lot of unnecessary calculation is entailed.

As an alternative, SRL can be solved by finding constituents as argument candidates through a top-down search to improve the processing time. These phrases or constituents are already generated by the syntactic parsers such as Collins (1999) or Charniak (2000) and need not be rediscovered during argument identification. The CoNLL 2005 shared task shows that these state-of-the-art ML parsers can provide good performance for converting a syntactic structure from plain text to a syntactic tree, therefore role labelling can proceed considering only pre-parsed constituents.

These parse trees have helped argument classifiers to reduce the different combination of words like the claim of the necessity of parsing for predicate argument recognition (Gildea and Palmer, 2002), but they also include a lot of embedded nodes. An invalid argument of a constituent “his office” in Figure.1.1 is embedded in the valid argument node “to his office”. This thesis proposes a refined algorithm (called Predicate-Argument Recognition Algorithm, PARA) that helps to convert a syntactic parse tree to semantic arguments more efficiently and correctly. Chapter 4 details this direct mapping methodology and shows the experimental results.

1.3 Contributions

This thesis makes contributions in the form of new methodologies for SRL including:

1) a direct mapping algorithm, Predicate-Argument Recognition Algorithm (PARA) for argument identification;

2) a new technique, Multi-Argument Classification implemented by Pattern-Matching (PM) for argument classification;

Specific contributions of these methodologies are itemised as follows:

Contribution to the problem of argument identification:

- Demonstration that it is possible to design a heuristic algorithm for directly mapping from parse trees to semantic arguments without any training.
- Demonstration that this approach outperforms existing systems for the CoNLL 2005 shared task using the same syntactic information.
- Demonstration that the proposed heuristic approach reduces execution time when compared to ML approaches.
- Evidence that this algorithm can be used as a pre-processor for argument identification to improve existing ML role classifiers, such as Palmer et al. (2005).
- Evidence that this algorithm is robust in a variety of corpus domains.

Contribution to the problem of argument classification:

- Demonstration that it is possible to do Multi-Argument Classification using the same input as was provided for the CoNLL 2005 shared task.

- Evidence that Multi-Argument Classification in SRL boosts the performance of argument classification in comparison to a statistical role classifier, (specifically the labeller of Palmer et al. 2005).
- Evidence that the combination of the Predicate-Argument Recognition Algorithm (PARA) as a pre-processor and the proposed multi-argument technique with basic syntactic features achieves better performance than existing systems that use rich syntactic features based on the same syntactic information.

1.4 Thesis Structure

The research presented in this thesis is organized as follows.

A background of Semantic Role Labeling is presented in Chapter 2. The chapter explains the main concepts involved in Semantic Role Labeling research based on supervised machine learning. It also reviews the relevant literature including domain-independent and specific semantic roles, basic features used in Semantic Role Labeling, the CoNLL shared tasks, and latest approaches.

An existing statistical approach in Semantic Role Labeling is described and reimplemented in Chapter 3. The chapter describes some improvements based on a statistical estimation proposed by Palmer et al. (2005). This thesis uses this modified system as a baseline because of its simplicity. Some heuristic improvements are proposed for this system, which are also included as part of the new technique in Chapter 5.

A tree-based Predicate-Argument Recognition Algorithm (PARA) presented in Chapter 4 is one of the main contributions of this thesis for the sub-problem, argument identification. This chapter demonstrates it is possible to achieve a direct mapping from syntactic parses to semantic arguments in Semantic Role Labeling. Despite being based solely on heuristic rules, this algorithm not only produces comparable results to the best system for argument recognition in the CoNLL 2005 shared task but also outperforms existing approaches using the same syntactic information.

Existing systems for role classification are all singular argument classifiers. Chapter 5 describes the second (and more significant) contribution of this thesis: argument labeling using Multi-Argument Classification. In order to boost performance, the technique for Multi-Argument Classification builds upon the technique of Singular-Argument Classification introduced in Chapter 3. This chapter first describes the architecture of the system and then proposes a new approach combined with traditional role classifiers to deal with novel role patterns. The system is analyzed by extensive empirical experimentation, showing that the system provides good performance.

The last chapter, Chapter 6, discusses conclusions to the work presented in this thesis and outlines future research directions.

Chapter 2

Background

This chapter provides the linguistic background for Semantic Role Labeling (SRL), which includes two types of semantic roles: domain-independent and domain-dependent. The SRL problem is defined in Section 2.2 and domain-independent semantic role assignment is discussed in Section 2.3, domain-dependent semantic role assignment in Section 2.4, and the CoNLL 2005 shared tasks in Section 2.5.

2.1 Introduction

The goal of Semantic Role Labeling (SRL) is to find all arguments for all predicates in a sentence (argument identification), and classify them according to semantic roles such as AGENT, THEME, TIME (argument classification). AGENT is the argument having control of the action expressed by the predicate, THEME is the participant affected by the action of the predicate and TIME is when the action happens. General thematic roles are classified as AGENT, THEME, and TIME. These classifications were proposed in Fillmore (1968) and Jackendoff (1972) to explain the syntactic realization of semantic arguments. Gildea and Jurafsky (2002, p. 279) note that "This level of roles, often called **thematic roles**, was seen as useful for expressing generalizations such as 'If a sentence has an AGENT, the AGENT will occur in the subject position', and such correlations might enable a statistical system to generalize from one semantic domain to another." Another set of role annotations in the PropBank[1], introduced by Palmer et al., (2005) has verb-specific numbered roles, such as A0 and A1, and modifiers, such as temporal markers and locatives. Palmer et al., (2005) used numeric

[1] http://www.cis.upenn.edu/~ace/

roles beginning with zero (A0) instead of general thematic roles (such as AGENT) for each verb or predicate. They use numerated roles to avoid disputing the meanings of labels. Generally, A0 is the argument exhibiting features of a prototypical AGENT (Dowty 1991), while A1 is a prototypical THEME. But no consistent generalizations can be made across verbs for the higher-numbered arguments such as A2, A3, and so on. More details are introduced in Section 2.2 and Section 2.3.

In order to ground what has been discussed so far, an example of SRL is given for the following sentence.

(Ex. 2.1) John, who kicked Mary this morning, was scolded by his mother at home.

The parse tree of this sentence is shown in Figure 2.1. There are two predicates "kick" and "scold". The arguments associated with the predicate "kick" are "John", "who", "Mary" and "this morning", which are subject, reference of subject, object, and time modifier respectively (Gildea and Jurafsky, 2002). Their shallow semantic roles are A0, R-A0, A1, and AM-TMP, respectively, using the definition from PropBank. A0 refers to the subject of the predicate; R-A0 to the reference of this subject; A1 to the object of the predicate; and AM-TMP to the time modifier. The arguments associated with the predicate "scold" are "John, who kicked Mary this morning", "by his mother", and "at home", which are the subject, object, and location modifier respectively. The semantic roles using the definition of PropBank, are denoted as A1, A0, and AM-LOC respectively.

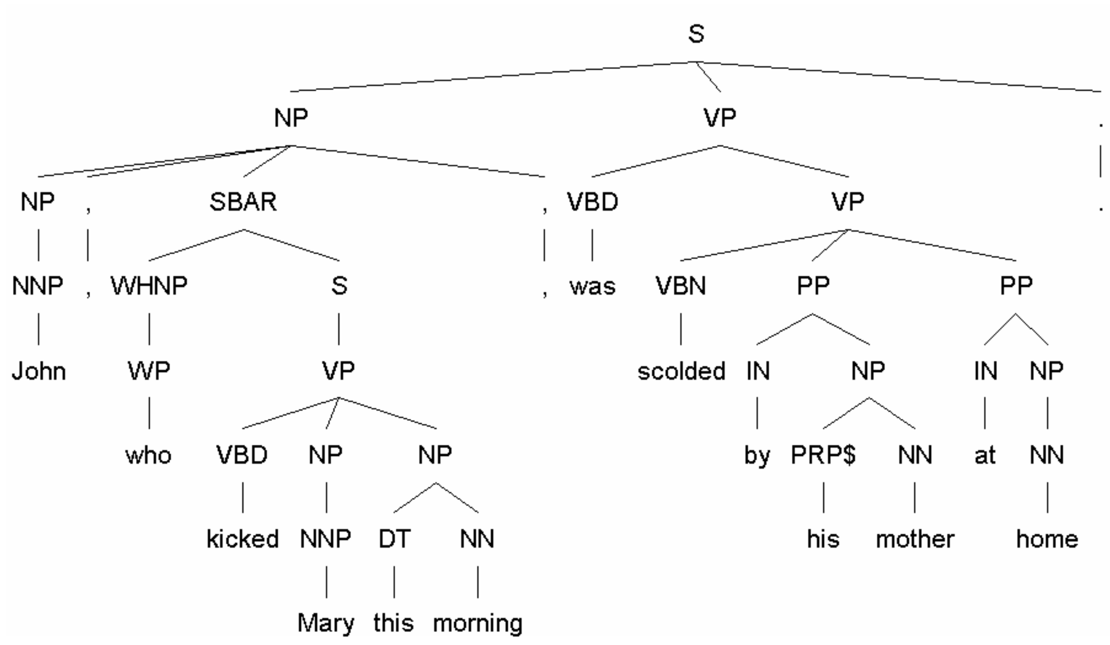


Figure 2.1. Parse tree for (Ex 2.1).

2.2 Problem definition

In a sentence, phrases group words that together represent a linguistic element of some nature. For example, the sentence “the dog bites a tennis ball” can be segmented into three basic syntactic phrases: a noun phrase (the dog), a verb phrase (bites), and another noun phrase (a tennis ball). SRL seeks to find constituents or groups of words related to predicates and classify these constituents with correct labels. The following is a definition of the general problem discussed in this thesis, in the context of a supervised learning problem, and a description of the standard evaluation method of a SRL system.

Let X be the space of sentences of a language. Let Y be the space of labeled argument structures for sentences. In particular, an element y belonging to Y is a set of labeled arguments constituting a well-formed bracketing in a sentence (that is, arguments do not overlap, though they admit embedding in a sentence).

Given a training set S = { $(x_1; y_1), \ldots, (x_m; y_m)$ }, where x_i are sentences in X, and y_i are argument structures in Y, the goal is to construct a function, $R : X \rightarrow Y$ that correctly recognizes arguments with correct labels on unseen sentences with given predicates.

For example, in Figure 2.1, the sequence of words, "Mary this morning" after the predicate "kicked" can be a set of one argument with three words, {"Mary this morning"}, a set of two arguments, {"Mary", "this morning"}, {"Mary this", "morning"} or {"Mary morning", "this"}, a set of three arguments {"Mary", "this", "morning"}, or null, which means there are no arguments in this sequence. The valid set of arguments is {"Mary", "this morning"}. The two arguments are evaluated as correct semantic roles if and only if they have the correct words (or word spans in the sentence) and are also assigned the correct role labels, which are A1 and AM-TMP according to PropBank respectively. The result would be incorrect if the semantic roles were the same but with different word spans, such as the set of result {"Mary this", "morning"} with roles {A1, AM-TMP}.

Therefore, the goal of SRL is to find all arguments with correct word spans in a sentence (argument identification) and classify these arguments with correct semantic roles (argument classification). An argument related to a predicate is correct if and only if both word spans and labels in that argument are correct.

There are two types of semantic roles: domain-dependent and domain-independent. Generally, when we refer to semantic roles like AGENT or THEME, they are domain-independent, which means these roles reveal a general semantic concept for predicates. For example, subjects of active predicates are always "AGENTs". The subjects of an active predicate in a domain-dependent setting vary according to the domain concerned. The following sections, 2.2 and 2.3, describe domain-independent and domain-dependent semantic roles respectively.

Roles	**Liu**	**Rosa**	**Description**
AGENT	+	+	the argument having control of the action expressed by the predicate
EXPERIENCER		+	the participant who does not have control of an action, expressing a psychological state
CAUSE	+	+	the argument that initiates the action expressed by the predicate without controlling it
PATIENT		+	the participant affected directly by the action of the predicate, usually changing state
THEME	+	+	the participant affected indirectly by the action of the predicate without changing state
SOURCE	+	+	the entity where the action originates
GOAL	+	+	the entity towards which the action takes place
BENEFICIARY	+	+	the referent that receives some advantage or disadvantage from the action
VALUE/ QUANTITY	+	+	the quantity
INSTRUMENT	+	+	the tool used in the action
MANNER	+		the way in which the action is performed
TIME	+		when the action happens
LOCATION	+		where the action happens
PROPOSITION	+		the argument that is a plan, an offer, or a subject suggested by the predicate
RESULT	+		the argument that is initiated by the action expressed by the predicate

Table 2.1. Lists of semantic roles introduced by Liu and Soo (1993) and Rosa and Francozo (1999)

2.3 Domain-independent semantic roles

Linguistic theory refers to the roles words have in relation to a verb as *semantic role*s (Haegeman, 1991). The verb is called the predicate. The semantic relationships between a group of words or phrases in a sentence are accounted for by the assignment of *semantic role*s.

2.3.1 Semantic Roles

Various sets of semantic roles have been proposed. For example, Liu and Soo (1993) described thirteen semantic roles, and Rosa and Francozo (1999) used another set of ten terms. Both semantic roles or thematic role sets are shown in Table 2.1. The following examples show semantic roles in brackets (using Liu's definition):

(Ex. 2.2) Bill [AGENT] knocked John [THEME] twice [QUANTITY] with a stone [INSTRUMENT] in school [LOCATION] yesterday [TIME].

(Ex. 2.3) Mary [BENEFECIARY] won $300 [QUANTITY] from Lotto [SOURCE] last week [TIME].

(Ex. 2.4) The fire [CAUSE] killed two persons [PATIENT].

In (Ex. 2.2), "Bill" has control of the "knock" action and has performed it twice to affect the participant "John" with an instrument "stone" in the location "school". The temporal factor of this action is "yesterday". In (Ex. 2.3), a beneficiary "Mary" has an action "win", which was happened "yesterday". The QUANTITY of reward is $300 obtained from the SOURCE, "Lotto". In (Ex. 2.4), "two persons" is the PATIENT and "the fire" is the CAUSE of the action "kill".

In general, a noun phrase constituent of a sentence may have different semantic roles for different verbs in different uses. For example, "Bill" has different semantic roles in the following examples (using Liu's definition).

(Ex. 2.5) Bill [AGENT] turned on the light.

(Ex. 2.6) Bill [BENEFICIARY] inherited a million dollars.

(Ex. 2.7) The magic wand turned Bill [THEME] into a frog.

Even with the same predicate, the structure of semantic roles can change. In the sentence

(Ex. 2.8) The man [AGENT] broke the window [PATIENT] with the stone [INSTRUMENT],

one can intuitively find an AGENT *(the man*), a THEME *(the windo*w), and an INSTRUMENT *(the ston*e) so that one can say that *break* has a semantic structure with the roles [AGENT, PATIENT, INSTRUMENT] for 'the man', 'the windows' and 'the stone' respectively. But this structure can change, depending on the sentence. For the sentence

(Ex. 2.9) The stone broke the vase

there is a different semantic structure, since *the stone* is a CAUSE (it causes the action) and *the vase* is a THEME. The difference between (Ex. 2.8) and (Ex. 2.9) is that although the same verb is employed *(break*), no AGENT or INSTRUMENT is expressed in (Ex. 2.8); thus, the semantic structure for (Ex. 2.9) – [CAUSE, THEME] – is different from the semantic structure for (Ex. 2.8).

There is no standard definition for general semantic roles. Table 2.2 shows another set of abstract semantic roles defined in Gildea and Jurafsky (2002), with representative examples from the FrameNet corpus[2] (Baker, Fillmore, and Lowe, 1998) (Fillmore and Baker 2000). All predicates are underlined and italicized and the corresponding roles are emboldened and highlighted.

[2] http://www.icsi.berkeley.edu/~framenet/

Role	Example
AGENT	**Henry** *pushed* the door open and went in.
CAUSE	Jeez, **that** *amazes* me as well as riles me.
DEGREE	I **rather** *deplore* the recent manifestation of Pop; it doesn't seem to me to have the intellectual force of the art of the Sixties.
EXPERIENCER	It may even have been that **John** *anticipating* his imminent doom ratified some such arrangement perhaps in the ceremony at the Jordan.
FORCE	If this is the case, can it be *substantiated* **by evidence from the history of developed societies**?
GOAL	Distant across the river, the towers of the castle rose against the sky straddling the only land *approach* **into Shrewsbury**.
INSTRUMENT	In the children with colonic contractions, **fasting motility** did not *differentiate* children with and without constipation.
LOCATION	These fleshy appendages are used to detect and *taste* food **amongst the weed and debris on the bottom of a river**.
MANNER	His brow *arched* **delicately**.
NULL	Yet while she had no intention of surrendering her home, **it** would be *foolish* to let the atmosphere between them become too acrimonious.
PATIENT	As soon as a character lays a hand on this item, the skeletal Cleric *grips* **it** more tightly.
PATH	The dung-collector *ambled* slowly **over**, one eye on Sir John.
PERCEPT	What is *apparent* is **that this manual is aimed at the non-specialist technician, possibly an embalmer who has good knowledge of some medical procedures**.
PROPOSITION	It says that rotation of partners does not *demonstrate* **independence**.
RESULT	All the arrangements for stay-behind agents in north-west Europe collapsed, but Dansey was able to *charm* most of the governments in exile in London **into recruiting spies**.
SOURCE	He heard the sound of liquid slurping in a metal container as Farrell *approached* him **from behind**.
STATE	Rex *spied* out Sam Maggott **hollering at all and sundry and making good use of his over-sized red gingham handkerchief**.
TOPIC	He said, "We would urge people to be aware and be *alert* **with fireworks** because your fun might be someone else's tragedy."

Table 2.2. Abstract semantic roles reproduced from Gildea and Jurafsky (2002).

Note that not all predicates or targets defined in FrameNet are verbs, for example "alert" in the "TOPIC" example, is an adjective.

2.3.2 Proposition Bank (PropBank)

The Proposition Bank (PropBank) (Pamler et al., 2005) is a version of the Penn TreeBank[3] annotated with verb argument structure discussed here in detail because it will be used throughout this thesis It introduces another domain-independent semantic role set with so-called shallow semantic roles such as A0, A1 (as core roles), representing roles similar to AGENT, THEME, and AM-TMP, AM-LOC (as adjunctive roles), representing temporal, locative roles.

The semantic roles covered by PropBank are as follows:

- Numbered arguments (**core roles**): Arguments defining verb-specific roles including A0, A1, A2, A3, A4, A5, AA;

- Adjuncts (**adjunctive roles**): General arguments that any verb may take optionally including AM-ADV, AM-CAU, AM-DIR, AM-EXT, AM-LOC, AM-MNR, AM-MOD, AM-NEG, AM-PNC, AM-PRD, AM-REC, AM-STR, AM-TMP, AM.

The following section illustrates the definition and examples of semantic roles in PropBank including core roles and adjunctive roles, based on the description by Olga Babko-Malaya[4].

Core roles

Olga Babko-Malaya states that there is no attempt made to ensure consistency of mapping between argument labels and the "roles" played by the arguments themselves in PropBank. In other words, the argument labels have no intrinsic meaning. For any

[3] http://www.cis.upenn.edu/~treebank/
[4] http://verbs.colorado.edu/~mpalmer/projects/ace/FramingGuidelines.pdf

individual verb or predicate, the mapping from argument label to semantic role is arbitrary, but it is consistent. For instance, in the examples below A0 maps to “agent” or “one offering”, A1 to “theme” or “thing offered”, and A2 to “beneficiary” or “person being offered to”.

(Ex. 2.10) ...the company [A0] to offer [V] a 15% stake [A1] to the public[A2].

(Ex. 2.11) ...Sotheby[A0] offered [V] the Dorrance heirs [A2] a money-back guarantee[A1].

(Ex. 2.12) ...an amendment [A1] offered [V] by Rep. Peter DeFazio...[A0]

(Ex. 2.13) ...Subcontractors [A2] will be offered [V] a settlement...[A1]

A0 .. A2 are arguments associated with a verb predicate, defined in the PropBank Frames scheme. If an argument satisfies two roles, the highest ranked argument label should be selected, where A0 >> A1 >> A2 >>… .

The role, AA, in PropBank denotes causative agents. An example of AA from the CoNLL 2005 shared task is:

(Ex. 2.14) “Moreover, your hypothetical investor has forsaken the gains to be had in reducing risk by diversifying his portfolio.”

The argument “your hypothetical investor” to the predicate “diversifying” is assigned the label AA denoting the causative agent for the argument, and the argument “his portfolio” the label A0.

The semantics of core roles depend on the verb and the verb usage in a sentence. Generally, the most frequent roles are A0 and A1 and, commonly, A0 stands for the agent and A1 corresponds to the patient or theme of the proposition.

Adjunctive roles

Directionals (AM-DIR), Locatives (AM-LOC), Manner Markers (AM-MNR), Temporal Markers (AM-TMP), Extent Markers (AM-EXT), Reciprocals (AM-REC), Markers of Secondary Predication (AM-PRD), Purpose clauses (AM-PNC), Cause clauses (AM-CAU) Discourse Markers (AM-DIS), Adverbials (AM-ADV), Modals (AM-MOD), Negation (AM-NEG), Stranded (AM-STR) and Bare ArgM (AM) are adjunctive roles or functional tags. Those adjuncts are defined in PropBank as optional elements within the argument structure of a verb. Following are more details for each adjunct (summarized from Olga Babko-Malaya).

1. Directionals (DIR)

Directionals are markers that show motion along some path. Both source and goal are grouped under the directional tag, and if there is no clear path being followed, a location marker should be used instead. For example, "walk *along the road*" is a directional, but "walk *around the countryside*" is a location.

2. Locatives (LOC)

Locatives indicate where an action takes place. The label is not restricted to physical locations, but abstract locations are included as well. For example, the italic words in the sentence of "*in his speech* he was talking about …" are labeled as a location.

3. Manner Markers (MNR)

A classical adverb shows how an action is performed. For example, works *well with others* is a marker of manner. Manner tags should be used when an adverb is an answer to a question starting with *how*.

4. Temporal markers (TMP)

Temporal markers show when an action took place, such as *in 2007*, *next Wednesday*, *sooner or later* or *now*. Adverbs of frequency (e.g. *often*, *always*, or *sometimes*, but *never* is an exception, see NEG below), adverbs of duration (*for a year/in an year*), order (e.g. *first*), and repetition (e.g. *again*) are also included in this category.

5. Extent Markers (EXT)

Extent Markers indicate the amount of change occurring from an action. It is mostly used for numerical adjuncts like "raised prices *by 10%*", quantifiers such as *a lot*, and comparatives such as "he raised prices *more than she did*".

6. Reciprocals (REC)

Reciprocals include reflexives and reciprocals such as *himself, itself, themselves, together, each other, jointly, both*, which refer back to one of the other arguments. These two arguments are co-indexed. Thus, "John and Mary killed *each other*" is a reciprocal. This marker is also used for reflexives, as in "John killed *himself*."

7. Markers of secondary predication (PRD)

Markers of secondary predication are used to show that an argument (or rarely an adjunct) of a predicate is in itself capable of carrying some predicate structure. Typical examples include resultatives (e.g. "The boys pinched them *dead*", and depictives (e.g. "upplied *as security in the transaction*".

8. Purpose clauses (PRP)

This is a special type of secondary predication, which is used to show the reason or motivation for some action. Therefore, clauses starting with "*because of*" or "*in order to*" belong to the purpose clauses.

9. Cause clauses (CAU)

Similar to "Purpose clauses", Cause clauses indicate the reason for an action. Clauses starting with "*because*" or "*as a result of*" are cause clauses. Questions with '*why*' also belong to this marker.

10. Discourse Markers (DIS)

Discourse Markers are tags which connect a sentence to other sentences in the context. They include small items such as *also* and *however* but also clauses such *as we have seen before*. Examples of discourse markers are *also, however, too, as well, but, and, as we have seen before, instead, on the other hand, for instanc*e, etc. Conjunctions such as *but* or *and* are only marked as DIS, if they occur at the beginning of the sentence.

11. Adverbials (ADV)

Adverbials are used for syntactic elements which clearly modify the event structure of the verb in question, but which do not fall under any of the headings above.

12. Modals (MOD)

Any of the modal verbs will be assigned as modal markers. Modals are: *will, may, can, might, should, could, would, probably.* Phrasal modals such as *going to* and *used to* are included in this category.

13. Negation (NEG)

The negation tag is used for elements such as *not, n't, never, no longer* and other markers of negative sentences.

14. Stranded (STR)

The stranded tag is not a modifier tag, meaning that it does not mark a certain type of modifier and does not appear in the final, or released, version of Propbank annotations. The use of the tag is motivated by technical reasons. This research disregards this tag because no examples of this tag appeared in the data of the CoNLL 2005 shared task.

15. Bare ArgM

A bare ArgM is often used as an adjunctive clause, which modifies one of the arguments of the verb, but is syntactically attached as an adjunct of the VP. An example of a Bare ArgM marker from the CoNLL 2005 shared task is:

(Ex. 2.15) "At the new rate, that would give them about $30 to travel on."

The argument, "about $30" to the predicate "travel" is assigned as Bar ArgM, which is used as an extra posed relative clause to modify the argument, "them" assigned as A0, but attached as an adjunctive role.

References, Continuation and Predicate

There are another two general roles assigned to all core roles and adjuncts, which are reference and continuation roles denoted as R-*, and C-*. Example 2.1 from Section 1.1 shows the referential role on A0, denoted as R-A0 for the predicate, "kick".

Predicates denoted as V include the continuation role as well, denoted as C-V. For example, the sentence "Kao [A0] took [V] U.S. market share [A1] away [C-V] from the mighty P&G [AM-MNR]" shows there is a continuation predicate (or verb particle) in the argument "away".

2.4 Domain-dependent semantic roles

Domain-dependent semantic role assignment includes different semantic frames, and frame elements (like semantic roles), and related target words (also called predicates or lexical units). Domain-dependent semantic roles represent the participants in an action or relationship captured by a semantic frame, which is a script-like conceptual structure that describes a particular type of situation, object, or event along with its participants and props. For example, the *Cooking_creation* frame describes a common situation involving a *Cook*, some *Produced_food*, raw Ingredients, and *Heating_Instrument* according to the definition in the FrameNet database as follows:

> "*Cooking_creation* : This frame describes food and meal preparation. A Cook creates a Produced_food from (raw) Ingredients. The Heating_Instrument and/or the Container may also be specified.
> (ex) **Caitlin** **BAKED** **some cookies** **from the pre-packaged dough**."

"Current information extraction and dialogue understanding systems are based on domain-dependent frame-and-slot templates to extract facts about such things as financial news, or interesting political events" (Gildea and Jurafsky, 2002). For example, systems for booking airplane information are based on domain-dependent frames with slots like FROM_AIRPORT, TO_AIRPORT, or DEPART_TIME (Stallard 2000). Systems for studying mergers and acquisitions are based on slots like JOINT_VENTURE_COMPANY, PRODUCTS, RELATIONSHIP, and AMOUNT (Hobbs et al. 1997).

Frame:	***Giving***
Frame Elements:	*Donor* *Recipient* *Theme*
Lexical Unit:	*bequeath.v, donate.v, donation.n, donor.n, endow.v, fob_off.v, foist.v, gift.n, gift.v, give.v, give_out.v, hand_in.v, hand.v, hand_out.v, hand_over.v, pass.v, pass_out.v, treat.v*
Frame:	***Adding_up***
Frame Elements:	*Cognizer* *Numbers* *Result*
Lexical Unit:	*add_up.v, number.v, tally.v, total.v*

Figure 2.2. Sample semantic frames from the FrameNet Lexicon.

A hand-labeled dataset called the FrameNet database (Baker, Fillmore, and Lowe, 1998; Fillmore and Baker 2000) (http://www.icsi.berkeley.edu/~framenet/) proposes roles that are neither as general as the ten abstract thematic roles, nor as specific as the thousands of potential verb-specific roles. The latest FrameNet lexical database (release 1.3) contains more than 10,000 targets words, more than 6,000 of which are fully annotated, in nearly 800 semantic frames, exemplified in more than 135,000 annotated sentences. Figure 2.2 illustrates two semantic frames, *Giving and Adding_up*, and their corresponding frame elements.

The *Giving* frame is invoked by semantically related verbs – *bequeath, donate, endow, fob_off, foist, gift give, give_out, hand_in, hand, hand_out, hand_over, pass, pass_out, and treate* as well as the nouns *donation, donor, and gift*. The roles defined for this frame, and shared by all its lexical entries, include three frame elements, *Donor*, *Recipient* and *Theme*. *Theme* is the object that changes ownership. *Recipient* is the entity that ends up in possession of the *Theme*. And *Donor* is the person that is in possession of the *Theme* and causes it to be in the possession of the *Recipient*. Similarly, the *Adding_up* frame has the roles *Cognizer*, *Numbers*, and *Result*, and is invoked by verbs like *add_up, number, tally* and *total*. *Cognizer* is the person doing the calculation. *Numbers* are the numbers that are used in the calculation, and *Result* is the outcome of the Cognizer's calculation.

A number of annotated examples from the JUDGMENT frame by Gildea and Jurafsky (2002) are included in Figure 2.3. The related frame elements or semantic roles are bracketed. More details and resources are listed in the FrameNet project.[5]

There are many resources related to SRL such as a free Lexical Conceptual Structure (LCS)[6] knowledge database for thematic and semantic roles, a lexical database for English language called WordNet[7], a verb-structure database called VerbNet[8], and so on.

[5] http://framenet.icsi.berkeley.edu/
[6] http://www.umiacs.umd.edu/~bonnie/LCS_Database_Documentation.html
[7] http://wordnet.princeton.edu/
[8] http://www.cis.upenn.edu/~mpalmer/project_pages/VerbNet.htm

The SRL systems presented in this thesis do not include these resources. The reason is that it would not be fair to compare a system that has those extra resources to systems that do not.

[$_{\text{Judge}}$ She] **blames** [$_{\text{Evaluee}}$ the Government] [$_{\text{Reason}}$ for failing to do enough to help.]

Holman would characterize this as **blaming** [$_{\text{Evaluee}}$ the poor.]

The letter quotes Black as saying that [$_{\text{Judge}}$ white and Navajo ranchers] misrepresent their livestock losses and **blame** [$_{\text{Reason}}$ everything] [$_{\text{Evaluee}}$ on coyotes.]

The only dish she made that we could tolerate was [$_{\text{Evaluee}}$ syrup tart which] [$_{\text{Judge}}$ we] **praised** extravagantly with the result that it became our unhealthy staple diet.

I'm bound to say that I meet a lot of [$_{\text{Judge}}$ people who] **praise** [$_{\text{Evaluee}}$ me] [$_{\text{Reason}}$ for speaking up] but don't speak up themselves.

Specimens of her verse translations of Tasso, Jerusalem Delivered and Verri Roman Nights circulated to [$_{\text{Manner}}$ warm][$_{\text{Judge}}$ critical] **praise** but unforeseen circumstances prevented their publication.

And if Sam Snort hails Doyler as monumental, is he perhaps erring on the side of being excessive in [$_{\text{Judge}}$ his] praise?

Figure 2.3. Examples of domain-dependent semantic roles, or frame elements, for target word JUDGMENT in the FrameNet database (reproduced from Gildea and Jurafsky, 2002).

2.5 CoNLL shared tasks in 2004 and 2005

The Conference on Natural Language Learning (CoNLL) has organized different shared tasks since 1999, including syntactic chunking, clause identification, name entity recognition, Semantic Role Labeling (SRL), and multi-lingual dependency parsing. In the problem of SRL as used in CoNLL 2004 and 2005 (Carreras and Marquez, 2004, 2005), the goal is to recognize all the arguments and their corresponding labels of given predicates in a sentence. Arguments related to a predicate are mostly phrases in the sentence that form a relationship with the predicate. This relationship is called a semantic role. The PropBank corpus discussed in Section 2.3.2 is used to annotate the predicate-argument relations of the verbs in the Wall Street Journal (WSJ) corpus with their semantic roles. In a sentence, each verb has a set of labeled arguments. In the following example sentence, the arguments of the verb "issue" are indicated:

(Ex. 2.16) (The San Francisco Examiner) [A0] (issued) [V] (a special edition) [A1] (around noon) [AM-TMP] (yesterday) [AM-TMP] (that was filled entirely with earthquake news and information) [C_A1].

According to PropBank, A0 is the issuer of the predicate "issue", and A1 is the thing issued (in the example, this argument is broken down into two pieces, the second annotated as C-A1). V stands for the verb, and AM-TMP is a general modifier expressing a temporal relation. For the verb "fill" the arguments are:

> The San Francisco Examiner issued (a special edition) [A1] around noon yesterday (that) [R_A1] was (filled) [V] (entirely) [AM-MNR] with (earthquake news and information) [A2].

A1 is the destination, R-A1 is a referent to A1, and A2 is the theme. AM-MNR stands for a general modifier expressing a manner relation.

```
(S
    (NP  (DT The)
         (ADJP
              (QP ($ $) (CD 1.4) (CD billion) )
         )
         (NN robot) (NN spacecraft)
    )
    (VP  (VBZ faces)
         (NP (DT a) (JJ six-year) (NN journey)
              (S
                   (VP  (TO to)
                        (VP  (VB explore)
                             (NP
                                  (NP (NNP Jupiter) )
                                  (CC and)
                                  (NP (PRP$ its) (CD 16) (JJ known) (NNS moons))
                             )
                        )
                   )
              )
         )
    )
    (. .)
)
```

Figure 2.4. An example in Penn Treebank.

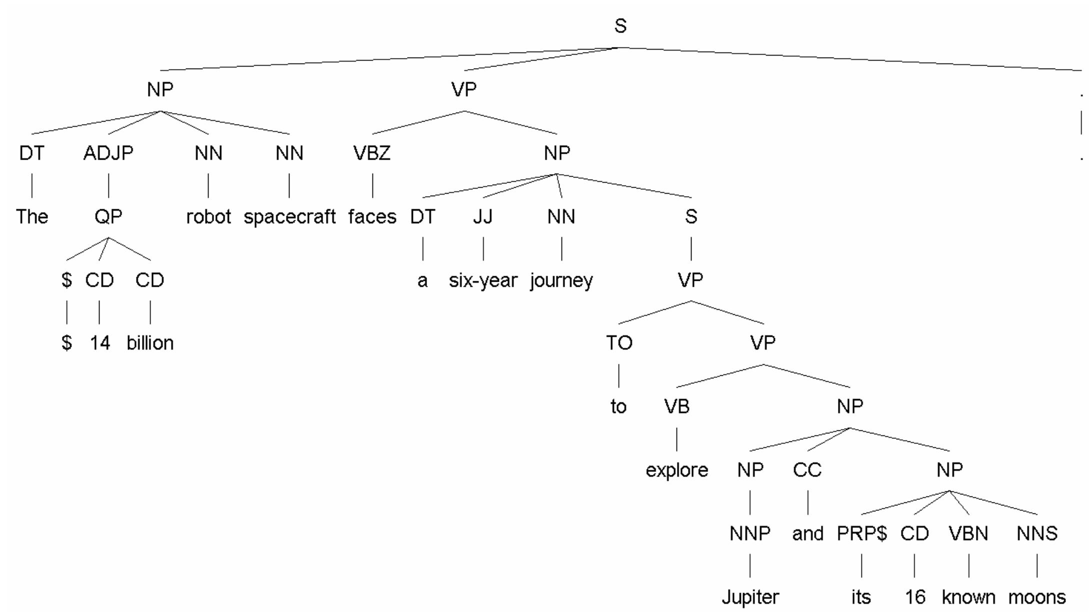


Figure 2.5. Syntax tree for a sentence illustrating the PropBank tags.

2.5.1 Data Format in CoNLL 2004, 2005

The Propbank corpus is based on the Penn TreeBank project[9]. In the following, we discuss an example structure extracted from the PropBank corpus used in the CoNLL shared tasks. The textual tree format for (Ex. 2.17) from the Penn TreeBank is in Figure 2.4, and the graphical representation is shown in Figure 2.5.

(Ex. 2.17) "The $1.4 billion robot spacecraft faces a six-year journey to explore Jupiter and its 16 known moons."

The CoNLL 2005 input data based on the syntax tree is shown in Figure 2.6. There is one line for each word (token), and a blank line after the last token. The columns, separated by spaces, represent different annotations of the sentence with a tagging alongside the words. Column 1 shows words in a sentence, Column 2, the name entity tags, Column 3, part-of-speech tags, and Columns 4 and 5 the partial syntactic information including phrase chunks and clauses. Column 6 covers the full syntactic information from the parse tree, and Column 7 has the target predicates.

9 http://www.cis.upenn.edu/~treebank/

WORDS	NE	POS	PARTIAL_SYNT		FULL_SYNT	TARGETS	PROPS	
The	*	DT	(NP*	(S*	(S(NP*	-	(A0*	(A0*
$	*	$	*	*	(ADJP(QP*	-	*	*
1.4	*	CD	*	*	*	-	*	*
billion	*	CD	*	*	*))	-	*	*
robot	*	NN	*	*	*	-	*	*
spacecraft	*	NN	*)	*	*)	-	*)	*)
faces	*	VBZ	(VP*)	*	(VP*	face	(V*)	*
a	*	DT	(NP*	*	(NP*	-	(A1*	*
six-year	*	JJ	*	*	*	-	*	*
journey	*	NN	*)	*	*	-	*	*
to	*	TO	(VP*	(S*	(S(VP*	-	*	*
explore	*	VB	*)	*	(VP*	explore	*	(V*)
Jupiter	(ORG*)	NNP	(NP*)	*	(NP(NP*)	-	*	(A1*
and	*	CC	*	*	*	-	*	*
its	*	PRP$	(NP*	*	(NP*	-	*	*
16	*	CD	*	*	*	-	*	*
known	*	JJ	*	*	*	-	*	*
moons	*	NNS	*)	*)	*)))))))	-	*)	*)
.	*	.	*	*)	*)	-	*	*

Figure 2.6. A tabular format for Ex. (2.4) in the CoNLL 2004, and 2005 shared tasks.

If a word is a predicate, the corresponding row contains the lemma of the predicate, otherwise a null node, "-". The seventh and twelfth rows correspond to the two predicates, "face" and "explore", and Columns 8 and 9 specify their respective arguments and roles.

The **Start-End format** in the CoNLL 2005 shared tasks represents phrases (chunks, arguments, and syntactic constituents) that constitute a well-formed bracketing in a sentence (that is, phrases do not overlap, though they admit embedding). The first six elements of the ninth column in Figure 2.6 are an example of how a syntactic constituent is specified. "(A0*" in the first line represents the start of the first argument for the predicate, “face”, and “*)” in the sixth line shows the end of the first argument. Each tag is of the form *STARTS*ENDS*, and represents phrases that start and end at the corresponding word.

2.5.2 Syntactic information

There are three kinds of syntactic information in the data of the CoNLL 2005 shared task: partial syntactic (UPC) notes, Collins’ parses, and Charniak’s parses. The partial syntactic notes (UPC) are located in the fourth column of Figure 2.6. A full syntactic parse tree (Collins’ or Charniak’s parses) appears in the fifth column of Figure 2.6. Note that some systems in the CoNLL 2005 shared task used lists of different syntactic parses including n-best parsings generated by other available tools. For example, n-best parsings for Charniak’s parser is denoted as “n-cha” (Pradhan et al., 2005), and “n-bikel” for Collins’ parser based on Bikel’s implementation (Sutton and McCallum, 2005). Most of the best performing systems in the CoNLL 2005 shared task used such additional syntactic input structures; such as the best system, Punyakanok et al., (2005), using five kinds of Charniak’s parses and one Collins’ parse.

2.5.3 Features

There are many features used in the CoNLL shared tasks (2004 and 2005). The basic features for Semantic Role Labeling (SRL) used by Palmer et al. (2005) are predicate, path, phrase type, position, voice, and head word, as shown in Table 2.3. All these features (except the predicate) are extracted from the syntactic parse tree of a given sentence. Some of the features, such as predicate, voice, and verb sub-categorization, are shared by all the nodes in the parse tree. All the others change with the constituent under consideration. For evaluation purposes, the CoNLL 2005 shared task only considers predicates and arguments which were annotated in PropBank. In an actual application, all verbs, such as the verb *is* and so on, would obviously also be considered as predicates.

Basic Features
· **Predicate** – The given predicate lemma (a verb without inflection by any tenses) is used as a feature.
· **Path** – The syntactic path through the parse tree. From the parse constituent to the predicate being classified. For example, in Figure 2.7, the path from A0 – "The officer" to the predicate "came", is represented with the string "NP ↑ S ↓ VP ↓ VBD" representing upward and downward movements in the tree respectively. There are 11 nodes in this sentence. Only three are valid arguments, which are the "NP" before the predicate, "VBD", and "PP".
· **Phrase Type –** This is the syntactic category (NP, PP, S, etc.) of the phrase corresponding to the semantic argument.
· **Position –** This is a binary feature identifying whether the constituent is before or after the predicate.
· **Voice –** Whether the predicate is realized as an active or passive construction. As Pradhan et al. (2004) mentioned, approximately 11% of the sentences in PropBank exhibit with a passive instantiation.
· **Head Word –** The syntactic head of the phrase. This is simply calculated by finding the last noun in a noun phrase. For example, in Figure 2.7, a valid NP argument, "The officer", for the predicate "come" has its last noun, "officer" as the head word of this valid argument.

Table 2.3. Basic features used in Palmer et al. (2005).

In total, the systems used in the CoNLL shared tasks (2004 and 2005) introduce more than 40 features (including feature combinations) to detect the arguments and roles of given predicates, such as first word, last word, length of the target constituent and so on. Table 2.4 lists some of these features. The partial path is a path from the constituent to the lowest common ancestor of the predicate and the constituent. For example, in Figure 2.7, the lowest common ancestor of the predicate "come" and the constituent "the officer" is the S node, and the partial path for it is represented with the string "NP ↑ S".

Some Rich Features
POS of Head word: the POS of the head word
Partial path: Path from the constituent to the lowest common ancestor of the predicate and the constituent.
Named Entities in constituents: Person, Organization, Location and Miscellaneous.
First and Last words and their POS tags in constituents
Punctuation: punctuation before and after the constituent

Table 2.4. Some rich features in SRL.

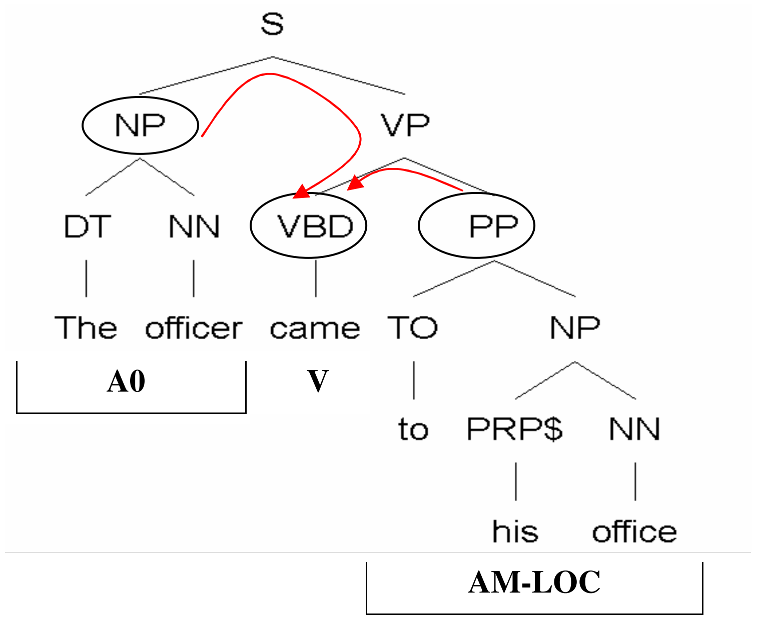


Figure 2.7. Illustration of path "NP ↑ S ↓ VP ↓ VBD" from a constituent "The officer" to the predicate "came".

In the same way, the partial path for the constituent “to his office” is “PP ↑ VP”. Some words in the CoNLL 2005 shared task are chunked and labeled with name entities as input for classification. These name entities are classified to Person, Organization, Location and Miscellaneous. The punctuation feature is used for if a constituent contains punctuation at the first and last words. More details of these rich features can be found in the description of the CoNLL shared tasks (2004 and 2005).

2.5.4 Probability Estimations

Many kinds of machine learning (ML) approaches for SRL were evaluated in the CoNLL 2005 shared task, including Maximum Entropy (ME) (Che et al. 2005, Haghighi et al 2005, Park and Rim 2005, Sang Tjong Kim 2005, Sutton and McCallum 2005, Tsai et al. 2005, Venkatapathy et al. 2005, Yi and Palmer 2005), Support Vector Machine (SVM) (Mistsumori et al. 2005, Moschitti et l. 2005, Ozgencil and McCracken 2005, Pradhan et al. 2005, Sang Tjong Kim 2005, Tsai et al. 2005), a Winnow-based networking of linear separators (SNoW) (Punyakanok et al. 2005), Decision Tree learning (Ponzetto and Strube 2005), AdaBoost algorithm (Marquez et al. 2005, Surdeanu and Turmo 2005), Memory-based Learning (MBL) (Sang Tjong Kim et al. 2005), Relevant Vector Machine (RVM) (Johansson and Nugues 2005), Tree Conditional Random Fields (T-CRF) (Cohn and Blunsom 2005), and a preliminary Multi-Argument Classification (Lin and Smith 2005). Palmer et al. (2005) proposed a statistical system based on Gildea and Jurafsky (2002). Some of most-used ML methods are briefly described as follows.

Support vector machines map input vectors to a higher dimensional space where a maximal separating hyperplane is constructed. Two parallel hyperplanes are constructed on each side of the hyperplane that separates the data. The separating hyperplane is the hyperplane that maximises the distance between the two parallel hyperplanes. An assumption is made that the larger the margin or distance between these parallel hyperplanes the better the generalisation performance (i.e. error rates on test sets) of the classifier will be.

The principle of maximum entropy first expounded by Jaynes (1957) is a method for analyzing the available information in order to determine a unique subjective probability distribution. It states that the most unbiased representation of knowledge from given information is that which maximizes the information entropy.

A decision tree is a predictive model, which is a mapping from observations about an item to conclusions about the item's target value. More descriptive names for such tree models are classification tree for discrete outcome or regression tree for continuous outcome. The machine learning technique for inducing a decision tree from data is called decision tree learning, or decision trees. In a decision tree, its internal or non-leaf nodes are tests on input data, and its leaf nodes are categories. A decision tree assigns a classification to a new input pattern by filtering the pattern down through the tests in the tree.

AdaBoost (or Adaptive Boosting) is an algorithm for constructing a "strong" classifier as a linear combination of "simple" "weak" classifiers. It can be used in conjunction with many other weak or basis algorithms (for example each feature can be considered as a weak classifier for SRL) to improve their performance.

A memory-based learning system is an extended memory management system that decomposes the input space either statically or dynamically into sub-regions for the purpose of storing and retrieving functional information. The main generalization techniques employed by memory-based learning systems are the nearest-neighbor search, space decomposition techniques, and clustering.

Generally, the main principal of ML approaches for SRL is to find the candidate arguments with the highest probabilities or scores. The probability or score function can be summarized as the probability of the optimal role assignment by Gildea and Jurafsky (2002):

$$argmax_{r_{1 \ldots n}} \sum P(r_{1 \ldots n} \mid f_{1,\ldots,n}, predicate)$$

where P($r_{1\ldots n}$ | $f_{1,\ldots,n}$, *predicate*) represents the probability or score of an overall assignment of roles $r_{1..n}$ to the n constituents or possible arguments of a sentence, given the predicate *predicate* and the various features $f_{1,\ldots,n}$ of each of the constituents. Details are shown in Section 3.3.

2.5.5 Results

The data used in this task consists of sections of the Wall Street Journal (WSJ) from the Penn TreeBank (Marcus et al., 1993), and three sections of the Brown corpus (namely, ck01-03). The evaluation is calculated with respect to precision, recall and $F_{\beta=1}$ (or F_1) measure of the predicated arguments. Precision (P) is the proportion of arguments classified by a system that are actually correct. Recall (R) is the proportion of correct arguments that are actually predicated by a system. The F_1 measure computes the harmonic mean of precision and recall. Chapter 3 will discuss these data and the evaluation measurements in detail.

Carreras and Marquez, (2005) state that the optimum achievable accuracy on these given parses from Charniak's parser is 88.25 in F_1 measurement on the WSJ 23 data, and 80.84 on the Brown corpus sections. Based on these parse trees, the best system (Punyakanok et al., 2005) in Table 2.5 can achieve F_1:79.44 on the WSJ domain and F_1:67.75 on the Brown corpus domain. This constitutes a drop of F_1:8.81 on the WSJ domain between the optimum achievable result and the result obtained by Punyakanok et al. (2005). This drop reveals processing propagating errors in Natural Language Processing applications when moving to different applications. It is mainly caused by the chained modules, which means errors are carried forward to next processing steps. These propagating errors affect results even more in other domains; there is a 13.09 drop in F_1 measurement (from 80.84 to 67.75) by Punyakanok et al., 2005) on the Brown corpus base on the given Charniak's parses. Performances by Charniak's parses show an about F_1:7.41 of degradation before classification from moving WSJ (88.75) to Brown (80.84) domains, but performances by Punyakanok et al. (2005) increase to 11.69 after classification. The additional 4.28 drop is caused by the SRL classifier, Punyakanok et al. (2005). The CoNLL 2005 shared task organisers state that "Error propagation and amplification through the chained modules (such as PoS Tags, Chunking, Parses, and so on) make the final output generalize very badly when changing the domain of application" (CoNLL 2005, p. 163).

It is also clear that systems with multi-syntactic parses outperform other systems that just use one syntactic parse. The best performing system, Punyakanok et al. (2005) uses seven different types of syntactic information ("Synt" in Table 2.5). The best system, developed by Surdeanu and Turmo (2005), in the CoNLL 2005 shared task that used only one type of syntactic information obtains 76.36 in F_1 measurement, which is about 3 points below the best performing system.

	WSJ			**Brown**			
System	**P**	**R**	**F1**	**P**	**R**	**F1**	**Synt**
Punyakanok	82.28	76.78	**79.44**	73.38	62.93	**67.75**	**7**
Haghighi	79.54	77.39	78.45	70.24	65.37	67.71	3
Marquez	79.55	76.45	77.97	70.79	64.35	67.42	5
Pradhan	81.97	73.27	77.37	73.73	61.51	67.07	2
Surdeanu	80.32	72.95	76.46	72.41	59.67	65.42	1
Tsai	82.77	70.90	76.38	73.21	59.49	65.64	1
Che	80.48	72.79	76.44	71.13	59.99	65.09	1
Moschitti	76.55	75.24	75.89	65.92	61.83	63.81	1
Tjongkimsang	79.03	72.03	75.37	70.45	60.13	64.88	1
Yi	77.51	72.97	75.17	67.88	59.03	63.14	1

Table 2.5. Top ten systems in the CoNLL 2005 shared task

Not all systems submitted to the CoNLL 2005 shared task reported their training and execution time. McCracken, (2005) highlighted training size and training time for each system. The training time varies from 9.7 hours (Haghighi et al., 2005) to about two and half days (Moschitti et al., 2005). Ponzetto and Strube (2005) used a "lazy learning" configuration for SRL. Although they do not provide details on their approach, they reported that the training time for all 21 datasets is approximately 2.5 hours on a 2 GHZ Opteron dual processor server with 2GB memory based on using only a single CPU at runtime, since the implementation is not parallelised. Execution time varies from 0.12 to 5.7 sec/sentence. Table 2.6 lists the training and execution time and hardware for some systems used in the CoNLL 2005 shared task.

System	Training time	Testing (Sec/Sen)	Hardware
Punyakanok	16.5 hours	5.7	
Haghighi	9 hours 40 mins		3.6GHz, 4GB RAM
Marquez	About 2 days	1.34	6-Linux cluster, 2 GB RAM
Che	20 hours	0.12	2.4 GHz, 1GB RAM
Moschitti	About 2.5 days		2.4 GHz, 4GB RAM
Ozgencil	30 hours	3.3	
Cohn	15 hours	0.22	20-node PowerPC cluster, 62GB RAM
Ponzetto	2.5 hours	0.76	2GHz Opteron, 2GB RAM

Table 2.6. List of training and execution time for some systems in the CoNLL 2005 shared task.

2.6 Summary

Semantic Role Labeling (SRL) must find all arguments for all predicates in a sentence and classify these arguments according to semantic roles. There are two types of semantic roles: domain-independent and domain-dependent. PropBank belongs to the former category, and FrameNet to the latter.

The CoNLL 2005 shared task evaluated SRL systems based on the domain-independent category. More than 40 features (including feature combinations) were used in the CoNLL 2005 shared task. Most of the systems in the CoNLL shared tasks (2004, and 2005) split SRL into two sub-problems: argument identification and argument classification. Most systems use Charniak parses, some use combinations of parsers including Charniak parses, Collins parsers, chunk-based parses, and n-best parsers such as n-cha for Charniak's parses, and n-bik for Collins' parses.

The four top-performing systems combine many individual SRL systems, each working on different syntactic structures to improve robustness and overcome the limitations in coverage and precision that occur when working with a single and incorrect syntactic structure. The best system obtained 79.44 in F1 on the WSJ 23 dataset, and was implemented by Punyakanok et al. (2005) and used seven parse trees. Surdeanu and Turmo (2005) presented the best system based on only one type of syntactic informant. When moving to another domain such as the Brown Corpus, performance of each system is about 10 points in F_1 lower than in the WSJ domain.

Timing evaluation is not mentioned for each system applied to the CoNLL shared tasks (2004, and 2005). McCracken (McCracken, 2005) shows the training time varies from 2.5 hours to about 2.5 days, depending on system architectures and hardware.

There are many methods that have been used to address the problem of SRL. Gildea and Jurafsky (2002) have addressed the SRL problem with basic features and tree parses for domain-dependent semantic roles and thematic roles. Palmer et al.'s, (2005) system based on Gildea and Jurafsky's work implemented this statistical approach with basic features for shallow SRL in PropBank annotations, which is similar to the CoNLL 2005 shared task. The system by Palmer et al. (2005) is a good starting point for a researcher who is new to the SRL problem because of its simplicity and easy implementation. Their statistical approach has been embedded in the system described in Chapter 5 as a baseline to compare against the multi-argument architecture advanced in this thesis. This baseline system is the subject of the next chapter including improvements.

Chapter 3

Extension of a statistical system

This chapter discusses existing statistical systems (Gildea and Jurafsky, 2002, Palmer et al., 2005), and improvements for these systems. Palmer et al. (2005) include a statistical argument recognizer (SAR) for argument identification, and a statistical role classifier for role classification. In this chapter, we re-implement SAR and modify the statistical role classifier with additional features and heuristics to increase the overall performance by more than 8 points in F_1 on WSJ 23.

Section 3.1 identifies different tasks and features used in SRL. Section 3.2 shows the features used by the systems. The systems themselves are described in Section 3.3. Improvements to Palmer et al. (2005)'s system are described in Sections 3.4. The data, evaluation method, and results are discussed in Section 3.5, and 3.6. Section 3.7 provides some conclusions, remarks and a summary for this chapter.

3.1 Introduction

The problem of Semantic Role Labeling is described in Chapter 2. Figure 3.1 shows an example of a parse tree with syntactic information for the sentence "He has helped his father for a while". Each node in the parse tree can be classified as a semantic argument candidate (i.e. a Potential Argument node) or one that does not represent any semantic argument (i.e. a NULL node). The Potential Argument nodes can then be further classified into the set of arguments labels. For example, in Figure 3.1, the node PP that encompasses "for a while" is a Potential-Argument node, since it does correspond to a semantic argument – "AM-TMP". But the NP that encompasses "a while" is a NULL or invalid node because the NP is part of the valid semantic argument, “for a while” and does not correspond to a valid semantic argument to the predicate, “help”.

Once all potential arguments are identified, these arguments can be labeled with semantic roles. The results of Semantic Role Labelling for the example in Figure 3.1 are A0 for “he”, V (or Predicate) for “help”, A1 for “his father”, and AM-TMP for “for a while”.

SRL can be solved as one problem, like in (Gildea and Jurafsky, 2002), or separated into two sub problems, argument identification, and argument classification. Pradhan et al., (2004) summarizes the possible division of labour as follows:

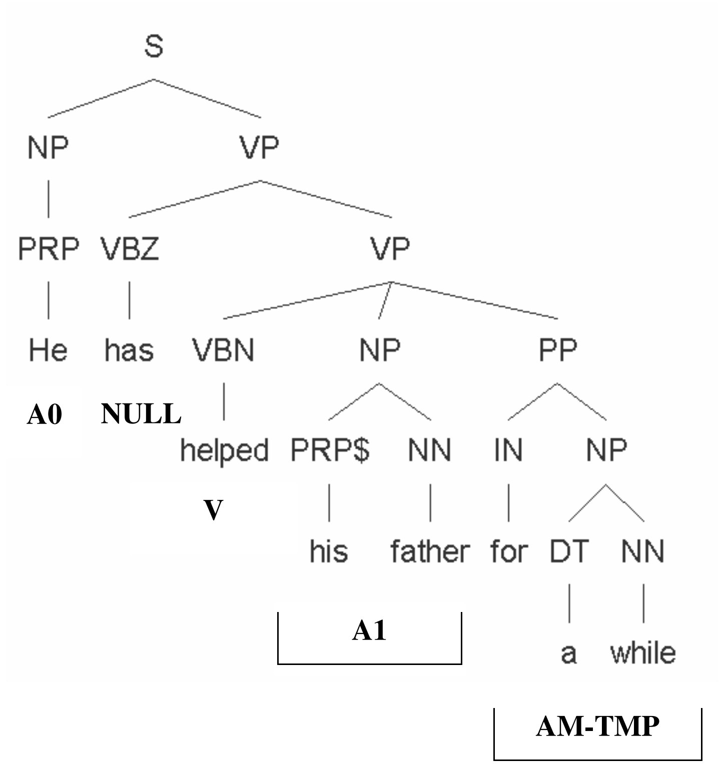


Figure 3.1. A sample sentence with syntactic information and semantic labels.

- **Argument identification –** This is the process of identifying phrases, constituents, or groups of words in a given sentence that represent valid semantic arguments for a related predicate or verb.
- **Argument classification** – This is the process of classifying given constituents or arguments. The given constituents, known to represent valid arguments of a predicate, are assigned appropriate semantic labels.
- **Argument identification and classification –** This is the process of combining the above two tasks, thus handling both the task of identifying probable constituents or phrases that represent semantic arguments of a predicate, and assigning to these constituents or semantic arguments the most likely semantic labels at the same time.

3.2 Features for Palmer et al. (2005)

Six features are used in the system of Palmer et al. (2005). These are Predicate, Path, Phrase type, Position, Voice and Head word. These basic features are shown in Table 2.3 and are widely used in most systems for SRL. These features are found or derived from a parse tree of a sentence. For example, features for the valid semantic argument “for a while” in Figure 3.1 are the **predicate** feature “help”, the **path** feature “PP ↑ NP ↑ VP ↓ VBN”, the **phrase type** feature “PP”, the **position** feature “after” and the **head word** feature “while” since this argument includes a child NP with “while” as the last noun. The “ ↑ ” in the path feature represents moving from the present node upward to the parent node and the “ ↓ ”, downward to the child node.

3.3 Probability Estimation Classifiers

This subsection describes the statistical systems used by Gildea and Jurafsky (2002) and Palmer et al., (2005) including their implementation. Both systems use statistical approaches for argument identification and argument classification.

3.3.1 Argument classification

To label the semantic role of a constituent or semantic argument automatically, Palmer et al. (2005) use a probability distribution indicating how likely the semantic argument is in filling each possible role, given the features described in Table 2.3, using the following estimation formula:

$$\mathrm{P}\,(r \mid hw, pt, path, pos, voice, predicate)$$

where r indicates the semantic role, *hw* the head word, *pt* the phrase type, and *pos* the position.

Gildea and Jurafsky (2002) claim that it is possible to calculate this distribution directly from the training data by counting the number of times each role appears with a combination of features and dividing by the total number of times the combination of features appears, such that:

$$\mathrm{P}\,(r \mid hw, pt, path, pos, voice, predicate) = \frac{\#\,(r, hw, pt, path, pos, voice, predicate)}{\#\,(hw, pt, path, pos, voice, predicate)}$$

Instead of using the whole feature set for the probability calculation, Gildea and Jurafsky (2002) built a classifier by combining probabilities from distributions conditioned on a variety of subsets of the features, to compensate for sparseness of data to avoid zero frequency. Gildea and Jurafsky (2002) point out that these probabilities can be simply calculated from the empirical distributions of the training data. In this way, the occurrences of each role and of each set of conditioning events are counted in a table, and probabilities are calculated by dividing the counts for each role by the total number of observations for each conditioning event. For example, the distribution $\mathrm{P}(r \mid pt, predicate)$ is calculated as follows:

$$P\,(r \mid pt, predicate) = \frac{\#\,(r, pt, predicate)}{\#\,(pt, predicate)}$$

There is a trade-off between more highly specified distributions, which result in higher accuracy but lower coverage, and less specified distributions, which have lower accuracy but higher coverage. The lexical head word statistics in particular are valuable for the system performance, as shown in Section 3.6.3, but are particularly sparse because of the large number of possible head words.

To combine the strengths of the various distributions, Gildea and Jurafsky (2002) merge them in various ways in order to obtain an estimate of the full distribution P (r | *hw, pt, path, pos, voice, predicate*). They investigate several combinations. The first combination method is linear interpolation, which simply averages the probabilities given by each of the distributions for each Potential-Argument:

$$\begin{aligned} P\,(r \mid hw, pt, path, pos, voice, predicate) = {} & \lambda_1 * P(r \mid predicate) + \lambda_2 * P(r \mid pt, predicate) + \lambda_3 * P(r \mid pt, path, predicate) + \\ & \lambda_4 * P(r \mid pt, pos, voice) + \lambda_5 * P(r \mid pt, pos, voice, predicate) + \lambda_6 * P(r \mid hw) + \\ & \lambda_7 * P(r \mid hw, predicate) + \lambda_8 * P(r \mid hw, pt, predicate) \end{aligned}$$

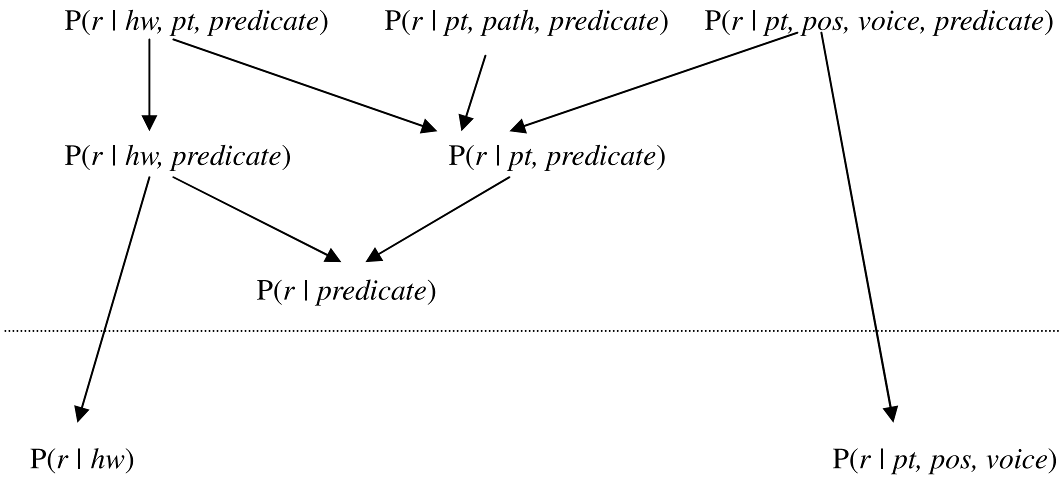
where $\sum_i \lambda_i = 1$ and $\lambda_i = \lambda_{i+1}$, i.e. $\lambda_i = 1/8$.

An alternative is to use the geometric mean, expressed in the logarithmic domain in a similar way as:

$$
\begin{aligned}
&P(r \mid hw, pt, path, pos, voice, predicate) = \\
&\quad 1/z * \exp\{\ \lambda_1 * \log P(r \mid predicate) + \lambda_2 * \log P(r \mid pt, predicate) + \\
&\qquad \lambda_3 * \log P(r \mid pt, path, predicate) + \lambda_4 * \log P(r \mid pt, pos, voice) + \\
&\qquad \lambda_5 * \log P(r \mid pt, pos, voice, predicate) + \lambda_6 * \log P(r \mid hw) + \\
&\qquad \lambda_7 * \log P(r \mid hw, predicate) + \lambda_8 * \log P(r \mid hw, pt, predicate) \\
&\qquad \}
\end{aligned}
$$

where z is a normalizing constant ensuring that $\sum_r P(r \mid \textit{Potential-Argument}) = 1$

Gildea and Jurafsky (2002) also propose another combination method—the back-off lattice. The back-off lattice constructs the distribution from more highly specified conditioning events to less specified ones. Palmer et al. (2005) modified this back-off lattice and applied it to PropBank corpus, as shown in Figure 3.2.



$$\begin{tikzcd}
P(r \mid hw, pt, predicate) \arrow[d] \arrow[drr] & & P(r \mid pt, path, predicate) \arrow[d] & & P(r \mid pt, pos, voice, predicate) \arrow[dll] \arrow[ddd] \\
P(r \mid hw, predicate) \arrow[dr] \arrow[dd] & & P(r \mid pt, predicate) \arrow[dl] & & \\
& P(r \mid predicate) & & & \\
P(r \mid hw) & & & & P(r \mid pt, pos, voice)
\end{tikzcd}$$

Figure 3.2. The back-off lattice for the Palmer et al., (2005) system.

The lattice represents just one way of choosing subsets of features for Palmer et al's, (2005) system. As Gildea and Jurafsky (2002) mentioned, with a set of N conditioning variables, there are 2^N possible subsets, and 2^{2^N} possible sets of subsets, giving a doubly exponential number of possible combinations. Instead of applying all possible subsets of features to probability estimation, based on the research of Gildea and Jurafsky (2002), Palmer et al., (2005) use the back-off lattice shown in Figure 3.2, which contains selected subsets of features, such as P(*r* | *hw, pt, predicate*) and so on.

The back-off lattice is used to select a subset of the available distributions to combine. The less specified distributions are used only when no data are present for any more highly specified distribution. For example, P(*r* | *pt, predicate*) is used when any more highly specified probabilities are zero, which are P(*r* | *hw, pt, predicate*), P(*r* | *pt, path, predicate*), and P(*r* | *pt, pos, voice, predicate*).

Combining Method	**Correct**
Equal linear interpolation	79.5 %
Geometric mean	79.6 %
Back-off, linear interpolation	80.4 %
Back-off, geometric mean	79.6 %

Table 3.1. Results on development set with 8167 observations, from Gildea and Jurafsky (2002).

Thus, the distributions selected are arranged in priority and the most specified distributions for which data is available are used. The selected probabilities can then be combined with either linear interpolation or the geometric mean, with results shown in Table 3.1. The combination of the back-off method and linear interpolation achieves the highest percentage of correct predication.

The system assigns arguments semantic roles in new data by looking for the highest-probability assignment of roles r_i to all constituents or semantic arguments i in the sentence, given the set of features $f_i = \{pt_i, path_i, pos_i, voice_i, hw_i\}$ for each constituent in the parse tree. The probability of the optimal role assignment r^* is formulated in (F 3.1) as follows.

(F 3.1) $r^* = \mathrm{argmax}_{r_{1..n}} \mathrm{P}(r_{1...n} \mid f_{1,..n}, predicate)$

where $\mathrm{P}(r_{1...n} \mid f_{1,..n}, predicate)$ represents the probability of an overall assignment of roles r_i (or $r_{1...n}$) to each of the n constituents or semantic arguments of a sentence, given the predicate, *predicate* and the various features f_i (or $f_{1,..n}$) of each of the constituents.

Gildea and Jurafsky (2002) apply Bayes' rule to this probability as follows:

$$r^* = \mathrm{argmax}_{r_{1..n}} \mathrm{P}(r_{1...n} \mid predicate) \frac{\mathrm{P}(f_{1...n} \mid r_{1...n}, predicate)}{\mathrm{P}(f_{1...n} \mid predicate)}$$

Then Gildea and Jurafsky (2002) make the assumption that the features of the various constituents of a sentence are independent given the predicate and each constituent's role and discard the term $\mathrm{P}(f_{1...n} \mid predicate)$, which is constant with respect to r:

$$r^* = \mathrm{argmax}_{r_{1..n}} \mathrm{P}(r_{1...n} \mid predicate) \prod_i \mathrm{P}(f_i \mid r_i, predicate)$$

Gildea and Jurafsky (2002) estimate the prior over constituent assignments as the probability of the constituent groups, represented with the set operator { }:

$$r^* = \mathrm{argmax}_{r_{1..n}} \mathrm{P}(\{r_{1...n}\} \mid predicate) \prod_i \mathrm{P}(f_i \mid r_i, predicate)$$

Gildea and Jurafsky (2002) apply Bayes' rule again,

$$r^* = \text{argmax}_{\,r_{1\,..\,n}} \text{P}(\{r_{1...n}\} \mid predicate) \prod_i \frac{\text{P}(r_i \mid f_i, predicate)\ \text{P}(f_i \mid predicate)}{\text{P}(r_i \mid predicate)}$$

Finally, Gildea and Jurafsky (2002) discard the feature prior P(f_i | *predicate*) as being constant over the argmax expression:

$$r^* = \text{argmax}_{\,r_{1\,..\,n}} \text{P}(\{r_{1...n}\} \mid predicate) \prod_i \frac{\text{P}(r_i \mid f_i, predicate)}{\text{P}(r_i \mid predicate)}$$

P(r_i | f_i, *predicate*) is the probability of a constituent's role given the above features for the constituent and the predicate. Palmer et al. (2005, p.96) mention "because of the sparseness of the data, it is not possible to estimate this probability from the counts in the training data. Instead, probabilities are estimated from various subsets of the features and interpolated as a linear combination of the resulting distributions. The interpolation is performed over the most specified distributions for which data are available, which can be thought of as choosing the topmost distributions available from a "back-off lattice" as shown in Figure 3.2. More details are described in Gildea and Jurafsky (2002) and Palmer et al. (2005).

The probabilities P(r_i | f_i, *predicate*) are combined with the probability P($\{r_{1...n}\}$ | *predicate*) for a set of roles appearing in a sentence given a predicate, using the following formula by Palmer et al. (2005):

$$\text{P}(r_{1...n} \mid f_{1...n}, predicate) \approx \text{P}(\{r_{1...n}\} \mid predicate) \prod_i \frac{\text{P}(r_i \mid f_i, predicate)}{\text{P}(r_i \mid predicate)}$$

Palmer et al. (2005, p.96) state "this approach allows interaction among the role assignments for individual constituents while making certain independence assumptions necessary for efficient probability estimation."

Figure 3.3 shows an example of a parse tree with propositions expressed as a chart. There is only one predicate, “take” with five arguments labelled with “A1”, “AM-MOD”, “A2”, “AM-TMP”, and “AM-ADV” respectively. These five arguments are “the economy’s temperature” for “A1”, “will” for “AM-MOD”, “from several vantage points” for “A2”, “this week” for “AM-TMP” and “with readings on trade, output, housing and inflation” for “AM-ADV”. The other words, like “be” in the sentence are not assigned any semantic roles.

The	(S1 (S (NP (NP*	-	(A1*
economy	*	-	*
's	*)	-	*
temperature	*)	-	*)
will	(VP*	-	(AM-MOD*)
be	(VP*	-	*
taken	(VP*	take	(V*)
from	(PP*	-	(A2*
several	(NP*	-	*
vantage	*	-	*
points	*))	-	*)
this	(NP*	-	(AM-TMP*
week	*)	-	*)
,	*	-	*
with	(PP*	-	(AM-ADV*
readings	(NP(NP*)	-	*
on	(PP*	-	*
trade	(NP*	-	*
,	*	-	*
output	*	-	*
,	*	-	*
housing	*	-	*
and	*	-	*
inflation	*)))))))	-	*)
.	*))	-	*

Figure 3.3. An example of a parse tree with propositions.

For argument classification with known arguments, the system is given the exact arguments without any label on each argument to classify the semantic role for each semantic argument. The system calculates the probability distribution from training data according to the formula (F 3.1), and sets up a selection table as shown in Table 3.2. The final results for the arguments are "A1", "AM-MOD" "A2", "AM-TMP" and "AM-MNR". There are five arguments, of which four are correctly labelled and one wrongly.

	ARG1	ARG2	ARG3	ARG4	ARG5
Highest Prob	A1	AM-MOD	A2	AM-TMP	AM-MNR

Table 3.2. An example of a selection table for argument classification with the highest probability (Highest Prob).

3.3.2 Argument identification

Section 3.3.1 describes how to label a semantic argument with a semantic role. The problem of argument identification is how to find these semantic arguments from a parse tree.

As with role labelling, to identify an argument from a parse tree, features are extracted from the sentence and its parse. These features are used to calculate probability tables for each node in the parse tree of a sentence. Each node in a parse tree is a Potential Argument, denoted as *pa*. In this case, each *pa* can be assigned NULL, or non-NULL by a binary indicator of whether a given constituent or Potential-Argument in the parse tree is or is not an argument to the specific predicate. Following Gildea and Jurafsky (2002), Palmer et al., (2005) calculated the three probability distributions learned from the training data : P(*pa* | *path*), P(*pa* | *path, predicate*), and P(*pa* | *hw, predicate*), where *pa* indicates an event where the parse constituent in question is an argument to the specific predicate, *path* is the path through the parse tree between the target predicate and the parse constituent, and *hw* is the head word of the parse constituent. An example is shown in the previous Section 3.3.1.

This research uses the same probability distribution from Gildea and Jurafsky (2002) with the back-off and linear interpolation method (in Table 3.1) for argument identification specifically:

(F 3.2) $P(pa \mid path, hw, predicate) =$

$$\lambda_1 * P(pa \mid path) + \lambda_2 * P(pa \mid path, predicate) + \lambda_3 * P(pa \mid hw, predicate)$$

where $\sum_i \lambda_i = 1$ and $\lambda_i = \lambda_{i+1}$, $\lambda_i = 1/3$.

This method can identify only those arguments that have a corresponding constituent in the automatically generated parse tree.

According to Gildea and Jurafsky (2002), more highly specified probability distributions such as P(*pa* | *path, predicate*) (denoted by P(*fe* | *path, t*) in Figure 3.4) perform relatively poorly compared to less ones, such as P(*pa* | *path*) (denoted by P(*fe* | *path*) in Figure 3.4), because of sparseness of the training data. There are only about 30 sentences available for each predicate (Gildea and Jurafsky, 2002). There is a threshold to this probability estimation for selecting potential arguments as valid ones. This threshold (set as 0.5 in their system) is used for a balance between precision and coverage. Figure 3.4 shows the Precision and Recall plot from Gildea and Jurafsky (2002) for identifying frame elements or semantic arguments in which Recall is calculated over only frame elements with matching parse constituents.

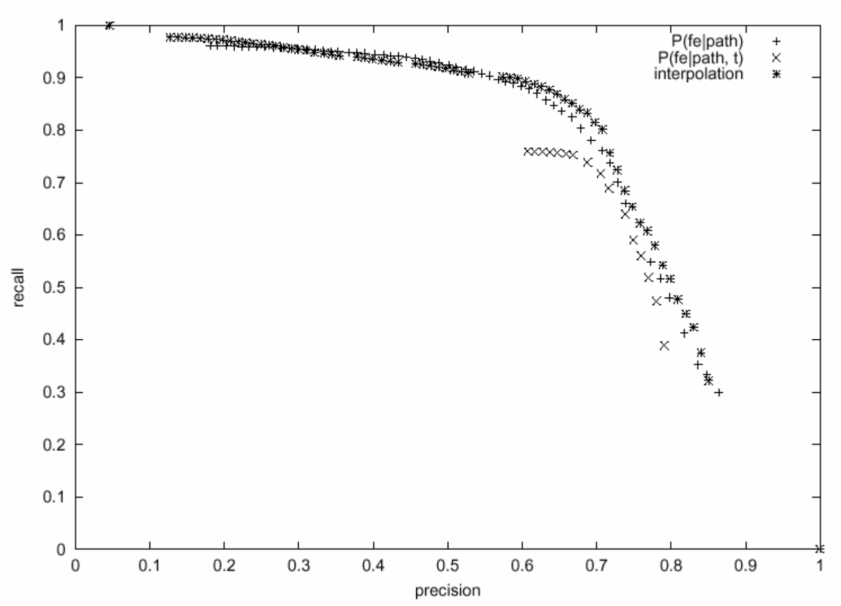


Figure 3.4. Illustration of Precision/Recall plot for various methods of identifying frame elements reproduced from Gildea and Jurafsky (2002).

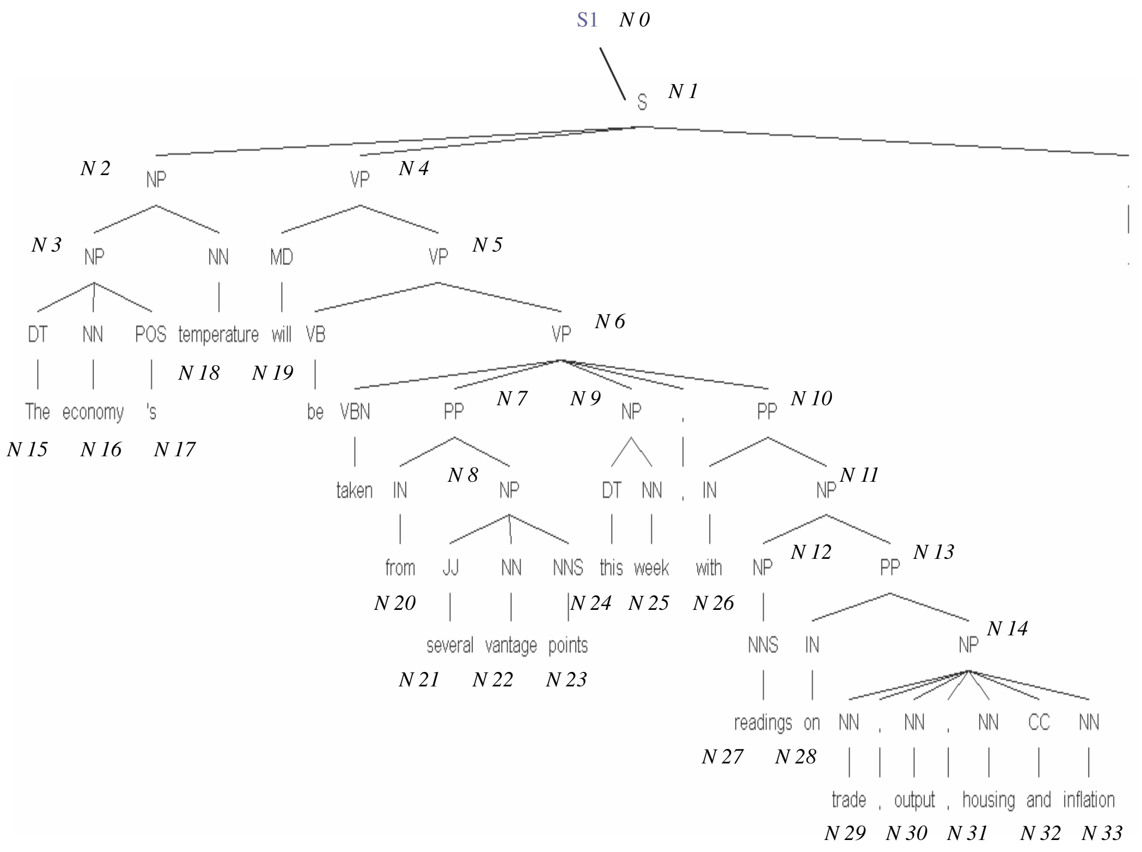


Figure 3.5. Graphic illustration of the example in Figure 3.3.

Figure 3.5 illustrates the graphic presentation for the same example in Figure 3.3. There are 34 individual potential arguments (or nodes): 15 phrases and 19 words out of 25 tokens. The punctuations, auxiliary words (like "be" in the fifth line) and the predicate are excluded. Each node can be a possible argument for the predicate, "take".

Table 3.3, including nodes, phrase type of headword, and selection type, shows an example of identification selection table for Figure 3.3. For example, the first line in Table 3.3 includes six nodes, N0, N1, N2, N3, N4 and N5, which correspond to the phrases, "S1", "S", "NP", "NP", "VP", and "NP" in Figure 3.5.

The system calculates the probability distribution learned from training data according to (F 3.2), and creates a selection table such as Table 3.3. The system calculates the probability distribution for each node in the parse tree, and checks if the probability of each node is bigger than a user-specified threshold. If not, an "**N**" letter is denoted in the third line of the node (like the "N0" node with "S1" phrase type). Before the probability calculation, a prior check is applied to see if the corresponding argument includes the predicate (indicated in the table by the letter ***P*** in the case of N4, N5, and N6). Once the probability of a node exceeds the threshold, a further check is applied if this valid Potential Argument is included in a larger-span valid node (indicated in the table by the letter ***O***). For example, N18 ("temperature") is included in N2 ("the economy's temperature") and N24 ("this") and N25 ("week") are included in N9 ("this week").

The reason for this check is that the CoNLL 2005 shared task does not allow overlapped or embedded arguments. If there exists unavoidable overlapping between overlapping (or covering) and overlapped (or covered) arguments such as "take *it* off", the covering argument, "take .. off", is split into two parts. One ("take") is before and the other ("off") is after the covered argument ("it"). A continuation label (defined in Section 2.3.2) is assigned to the latter argument based on the former semantic role. For example, the overlapping argument “take .. off” covering an overlapped argument, ”it” is split two arguments, ”take” and “off”. The second split argument is assigned the “C-V” due to the first one labelled with “V”. Then, semantic roles for this sentence with three arguments are “V” for “take”, “A1” for “it” and “C-V” for “off”.

After argument identification, the system outputs the five valid arguments(indicated in Table 3.3 by the letter **Y**), which are “N2”, “N7”, “N9”, “N10”, and “N19” in Table 3.3 or Figure 3.5 for further argument classification.

Nodes	N0	N1	N2	N3	N4	N5
Phrase or head	S1	S	**NP**	NP	VP	VP
Prob > Threshold	N	N	**Y**	N	***P***	***P***
Nodes	N6	N7	N8	N9	N10	N11
Phrase or head	VP	**PP**	NP	**NP**	**PP**	NP
Prob > Threshold	***P***	**Y**	N	**Y**	**Y**	N
Nodes	N12	N13	N14	N15	N16	N17
Phrase or head	NP	PP	NP	the	economy	‘s
Prob > Threshold	N	N	N	N	N	N
Nodes	N18	N19	N20	N21	N22	N23
Phrase or head	temperature	*will*	from	several	vantage	points
Prob > Threshold	***O***	**Y**	***O***	N	N	N
Nodes	N24	N25	N26	N27	N28	N29
Phrase or head	this	week	with	reading	on	trade
Prob > Threshold	***O***	***O***	***O***	N	N	N
Nodes	N30	N31	N32	N33		
Phrase or head	output	housing	and	inflation		
Prob > Threshold	N	N	N	N		

Table 3.3. An example of a selection table for argument identification.

3.3.3 Argument identification and classification

Gildea and Jurafsky (2002) merged argument identification and classification into one classifier. To integrate argument identification and argument classification for each constituent in a parse tree, Gildea and Jurafsky (2002) extended the formulation

$$\operatorname{argmax}_{r_{1..n}} \mathrm{P}(\{r_{1...n}\} \mid predicate) \prod_i \frac{\mathrm{P}(r_i \mid f_i, predicate)}{\mathrm{P}(r_i \mid predicate)}$$

where $f_i = \{pt_i, path_i, pos_i, voice_i, hw_i\}$, and r_i is each semantic role.

to include argument identification decisions:

$$\operatorname{argmax}_{r_{1..n}} \mathrm{P}(\{r_{1...n}\} \mid predicate) \prod_i \frac{\mathrm{P}(r_i \mid f_i, pa_i, predicate)\, \mathrm{P}(pa_i \mid f_i)}{\mathrm{P}(r_i \mid predicate)}$$

where pa_i is a binary variable indicating that a constituent is an argument and $\mathrm{P}(pa_i \mid f_i)$ is calculated using the formula (F 3.2) in Section 3.3.2.

When pa_i is true, role probabilities are calculated as before; when pa_i is false, r_i assumes an empty role, or NULL, with probability one and is not included in the selection list represented by $\{r_{1...n}\}$.

Table 3.4 shows the results of simultaneous argument identification and argument classification for the same example from Figure 3.3. All possible arguments are labelled, including the embedded ones with a gray background. N18 is labelled as A1, N20 as A2, N24 as A1, N25 as TMP, and N26 as MNR. After the probability calculation and classification, the system selects those arguments that are not covered by any other arguments, and outputs the labels for each argument, which are A1 for N2, A2 for N7, AM-TMP (TMP) for N9, AM-MNR (MNR) for N10, and AM-MOD (MOD) for N19. This procedure may spend time on some invalid arguments like N18 embedded in valid arguments like N2.

Nodes	N0	N1	N2	N3	N4	N5
Phrase or head	S1	S	**NP**	NP	VP	VP
Prob > Threshold	N	N	**Y**	N	***P***	***P***
Semantic role			**A1**			

Nodes	N6	N7	N8	N9	N10	N11
Phrase or head	VP	**PP**	NP	**NP**	**PP**	NP
Prob > Threshold	***P***	**Y**	N	**Y**	**Y**	N
Semantic role		**A2**		**TMP**	**MNR**	

Nodes	N12	N13	N14	N15	N16	N17
Phrase or head	NP	PP	NP	the	economy	‘s
Prob > Threshold	N	N	N	N	N	N
Semantic role						

Nodes	N18	N19	N20	N21	N22	N23
Phrase or head	temperature	*will*	from	several	vantage	points
Prob > Threshold	***O***	**Y**	***O***	N	N	N
Semantic role	A1	**MOD**	A2			

Nodes	N24	N25	N26	N27	N28	N29
Phrase or head	this	week	with	reading	on	trade
Prob > Threshold	***O***	***O***	***O***	N	N	N
Semantic role	A1	TMP	MNR			

Nodes	N30	N31	N32	N33
Phrase or head	output	housing	and	inflation
Prob > Threshold	N	N	N	N
Semantic role				

Table 3.4. An example of a selection table for argument identification and argument classification

3.3.4 Computational complexity

The computational complexity for calculating every possible combination of features is doubly exponential (Gildea and Jurafsky, 2002). Thus Gildea and Jurafsky (2002) used the back-off lattice from Figure 3.2 to represent some expected interaction between features. The linear interpolation method simply averages the probabilities given by each of the distributions. It is hard to calculate time complexity of using the back-off lattice because computation varies with different datasets, but the measured execution times exhibit a logarithmic behaviour as discussed in detail in Section 3.6.4 for the development data set WSJ 24.

3.4 Improvements

Palmer et al. (2005) reported the performance of their system on a preliminary release of the PropBank data. The preliminary data set used contained annotations for 72,109 propositions or predicate-argument structures including 190,815 individual arguments from 2,462 types of lexical predicates. In order to provide results comparable with the existing literature, Palmer et al. (2005) used the annotations from WSJ section 23 of the Treebank as the test set, and all other WSJ sections were included in the training set. They report the performance obtained by their system on WSJ 23 is 62.74 using the F_1 measurement, which indicates there is some room to improve compared to F_1: 76.46 by Surdeanu and Turmo (2005). Therefore, this subsection describes additional features and heuristics for pre-processing and post-processing that improve overall performance when applied to the system of Palmer et al., (2005).

The modified system implemented in this thesis uses additional features and heuristics to boost F_1 about 8 points on WSJ 23. The additional steps are itemized in Table 3.5. Additional features are used when determining argument roles. Instead of only using the preposition in a prepositional phrase (PP) as a headword (like "*in* the office"), we add as an additional feature for a PP the headword of the NP following the preposition (like “in the *office*”). For example, the role of “you” after the preposition “to” in the sentence, ”come to you” and the role of “school” in the sentence, ”come to school” can be identified as A2 and AM-LOC when adding the head word of the child NP in a PP in the role classification stage. If adding extra preposition feature results in zero frequency due to data sparseness, we can use the back-off lattice later to avoid this problem.

Moreover, instead of using a discrete position feature (before or after the predicate), we use a distance feature (i.e. the relative displacement from the predicate), which offers more precise location in an argument structure. Note that this feature has been widely used in existing different systems for the CoNLL 2005 shared task.

Additional features help to improve the performance of a system but they consume more computational resources. Another way to improve performance without sacrificing computational efficiency is to add heuristics when classifying arguments instead of adding features in the whole system. Thus we propose an Actor heuristic to reduce erroneous assignments when classifying arguments. This actor heuristic is implemented to filter out incorrect assignments for Agent or A0 by checking if a labelled argument with Agent or A0 is a subject when the predicate is in active voice, or checking if this labelled argument has a preposition (like “by”) prior to the argument when the predicate is in passive voice.

There are also simple cases for labelling such as AM-MOD and AM-NEG for which heuristics can be used. A single-word argument with a Modal POS tag can immediately be assigned an “AM-MOD” tag. In the same manner, a single-word argument of “not” or “n’t” can directly be assigned an “AM-NEG” tag without any probability estimation. These simple rule-based pre-assignments also appear in other existing systems (e.g. Che et al. 2005).

Additional features
Preposition – This feature is the preposition of a PP.
Distance – The relative displacement from the predicate (negative if the constituent is to the left, positive if it is to the right of the predicate).
Heuristics
Actor constraint – "A0" or "Agent" is assigned to an argument only if the argument is a subject before the predicate and the predicate is in active voice, or a preposition (such as "by", "with", and so on) appears prior to the argument after the predicate when the predicate is in passive voice.
Pre-Processing
Modal – "AM-MOD" tags are directly assigned to arguments with POS of "MOD", such as "will", "should", "going to" and so on. **Negation** – "AM-NEG" tags are directly assigned to one-word arguments with "not" or "n't". **Discourse** – "AM-DIS" tags are directly assigned to one-word arguments with POS of "CC".
Post-Processing
Duplication – if there are duplicate core roles in the same argument lists (for example, "A0 V A1 A1"), the system will keep the roles that are the closest to the predicate as ("A0 V A1 X") (Sang et al. 2005).

Table 3.5. Additional features and heuristics applied to the system of Palmer et al. (2005)

Post-processing is used to avoid duplicate core roles such as [A0 V A1 A1]. A simple way to implement this is to delete duplicate core roles that are far from the predicate as proposed by (Sang et al. 2005)[1]. This method can potentially reduce the coverage in role classification, but Section 3.6.3 shows that this method provides better performance in overall.

3.5 Data and evaluation

In order to compare the modified version of Palmer et al ‘s system (2005) with other techniques, we use the datasets and evaluation measures from the CoNLL 2005 shared task. Section 3.5.1 describes the data and Section 3.5.2 defines the evaluation measurements used in the CoNLL 2005 shared task.

Words	POS	Chunks	Clauses	Full trees	Name Entities	Targets	Propositions	
And	CC	*	(S *	(S *	*	-	(AM-DIS *)	(AM-DIS *)
to	TO	(VP *	(S *	(S(VP *	*	-	*	(AM-PNC *
attract	VB	*)	*	(VP *	*	attract	(V *)	*
younger	JJR	(NP *	*	(NP *	*	-	(A1 *	*
listeners	NNS	*)	*)	*))))	*	-	*)	*)
,	,	*	*	*	*	-	*	*
Radio	NNP	(NP *	*	(NP *	(ORG *	-	(A0 *	(A0 *
Free	NNP	*	*	*	*	-	*	*
Europe	NNP	*)	*	*)	*)	-	*)	*)
intersperses	VBZ	(VP *)	*	(VP *	*	intersperse	*	(V *)
the	DT	(NP *	*	(NP(NP *	*	-	*	(A1 *
latest	JJS	*)	*	*)	*	-	*	*
in	IN	(PP *)	*	(PP *	*	-	*	*
Western	JJ	(NP *	*	(NP *	(MISC *)	-	*	*
rock	NN	*	*	*	*	-	*	*
groups	NNS	*)	*	*))))	*	-	*	*
.	.	*	*)	*)	*	-	*	*)

Figure 3.6. An example of an annotate sentence.

[1] Sang et al. (2005) do not offer any justification for this post-processing.

3.5.1 Data

The data consist of sections of the Wall Street Journal (WSJ) from the Penn TreeBank (Marcus et al. 1993), with information on predicate-argument structures extracted from the PropBank corpus (Palmer et al. 2005). The experiment followed the standard partition used in syntactic parsing: WSJ sections 02-21 for training, WSJ 24 for development, which is a test data for developing systems, and WSJ 23 for testing. In addition, the test set used in the CoNLL2005 shared task includes three sections of the Brown corpus (namely, ck01-03). The predicate argument annotations of the latter test material are very valuable as they allow evaluating learning systems on a portion of data that comes from a different source than the training data.

Section 2.5 has described the data format and listed an example from the CoNLL 2005 shared task. Figure 3.6 shows another example of a fully-annotated sentence. A brief explanation for each column is as follows. The input consists of words (in the first column); POS tags (second); base chunks (third); clauses (forth); full syntactic tree (fifth) and named entities (sixth). The seventh column marks target verbs (predicates). Their propositions (i.e. resulting role assignments) are found in the remaining two columns, with one column for each predicate. According to the PropBank Frames for "attract", A0 annotates the attractor, and A1 the item attracted; for "intersperse", A0 is the arranger, and A1 the entity interspersed.

PropBank-1.0 was used in the CoNLL2005 Shared Task. Most verbs were annotated as predicates, but not all of them. For example, most of the occurrences of the verb "to have" and "to be" were not annotated. The description of the CoNLL 2005 shared task did not explain why these verbs are not included. The input data was annotated by pre-processing systems. The annotations are part-of-speech (POS) tags, chunks, clauses, full syntactic trees, and named entities. The following state-of-the-art systems werer used for pre-processing: UPC processors, consisting of a POS tagger (Gimenez and Marques, 2003) and a base chunker and clause recognizer (Carreras and Marquez, 2003), full parses from the Collins (1999) parser (with model 2), full parses from the

Charniak parser (2000), and named entity recognition (Chieu and Ng, 2003). Table 3.6 shows the number of sentences, tokens, annotated propositions, distinct verbs, and arguments in the three data sets. As Table 3.6 shows, there are 39,832 sentences in the training data with 3101 different verbs, 90,750 predicate-argument structures (propositions), 239,858 individual arguments, and 950,028 words (tokens).

	Train.	Development	Test WSJ	Test Brown
Sentences	39,832	1,346	2,416	426
Tokens	950,028	32,853	56,684	7,159
Propositions	90,750	3,248	5,267	804
Verbs	3,101	860	982	351
Arguments	239,858	8,346	14,077	2,177
A0	61,440	2,081	3,563	566
A1	84,917	2,994	4,927	676
A2	19,926	673	1,110	147
A3	3,389	114	173	12
A4	2,703	65	102	15
A5	68	2	5	0
AA	14	1	0	0
AM	7	0	0	0
AM-ADV	8,210	279	506	143
AM-CAU	1,208	45	73	8
AM-DIR	1,144	36	85	53
AM-DIS	4,890	202	320	22
AM-EXT	628	28	32	5
AM-LOC	5,907	194	363	85
AM-MNR	6,358	242	344	110
AM-MOD	9,181	317	551	91
AM-NEG	3,225	104	230	50
AM-PNC	2,289	81	115	17
AM-PRD	66	3	5	1
AM-REC	14	0	2	0
AM-TMP	16,346	601	1,087	112
R-A0	4,112	146	224	25
R-A1	2,349	83	156	21
R-A2	291	5	16	0
R-A3	28	0	1	0
R-A4	7	0	1	0
R-AA	2	0	0	0
R-AM-ADV	5	0	2	0
R-AM-CAU	41	3	4	2
R-AM-DIR	1	0	0	0
R-AM-EXT	4	1	1	0
R-AM-LOC	214	9	21	4
R-AM-MNR	143	6	6	2
R-AM-PNC	12	0	0	0
R-AM-TMP	719	31	52	10

Table 3.6. Statistics for the data sets of the CoNLL 2005 Shared Task.

The semantic roles covered by PropBank are as followings:

- Numbered arguments (core roles): Arguments defining verb-specific roles including A0, A1, A2, A3, A4, A5, AA
- Adjuncts (adjunctive roles): General arguments that any verb may take optionally including AM-ADV, AM-CAU, AM-DIR, AM-EXT, AM-LOC, AM-MNR, AM-MOD, AM-NEG, AM-PNC, AM-PRD, AM-REC, AM-STR, AM-TMP, AM;

Section 2.3.2 covers these semantic roles in detail.

3.5.2 Evaluation

Evaluation in the CoNLL 2005 shared task is performed on a collection of unseen test sentences with all syntactic information and target predicates as input annotation. For example, the first seven columns in Figure 3.6 are provided as input information (including word, POS, chunks, clauses, full trees, name entities and target predicates). A system must then create columns 8 and 9 for evaluation. A system is evaluated with respect to precision, recall and $F_{\beta=1}$ (or F_1) measure. Precision (p) is the proportion of arguments classified by a system that are correct. Recall(r) is the proportion of correct arguments that are predicted by a system. Finally, the $F_{\beta=1}$ measure computes the harmonic mean of precision and recall, and is the final measure to compare the performance of systems. It is given by:

$$F_1 = \frac{2*p*r}{(p+r)} \qquad \text{from} \qquad F_\beta = \frac{(1+\beta)*p*r}{(p*\beta + r)}$$

For example, assume a proposition is [A0 V A1 TMP] and the system's output is [A0 V A2 X]. (X denotes an unlabelled argument.) There are two classified arguments (excluding the predicate), and one has the correct label, which is A0. The *precision* (the proportion of classified arguments that are correct) is 1/2 (or 50%). There are three arguments in the proposition (excluding the predicate) and only one is labelled correctly. The *recall*, (the proportion of correct arguments that are predicted by a system) is 1/3 (or 33.33%). Therefore, F_1, the harmonic mean of precision and recall, is 0.4. For an argument to be correctly recognized, the words spanning the argument as well as its semantic role have to be correct.

An official script written in Perl, *srl-eval.pl*, that can be used to evaluate the performance of a system is obtained from the CoNLL 2005 shared task web site. This script evaluates the result from a system and outputs a table of precision, recall, and F_1 for each semantic role. It also outputs an overall performance. More details can be found at the website of the CoNLL 2005 shared task[2].

3.6 Empirical Results

The following three aspects of SRL can be tested: argument classification, argument identification and argument identification and classification. This section uses Palmer et al., (2005) as a baseline system and Sections 3.6.1 and 3.6.2 show experimental results for the baseline system and its variants. Section 3.6.3 outlines feature performance and analysis, and Sections 3.6.4 and 3.6.5 illustrate other results obtained from different sizes of training data including performance in terms of execution time. Sections 3.6.6 and 3.6.7 show the results of argument identification and classification on WSJ 24, which contain additional results including the impact of auto parses and gold (hand-corrected) parses. Sections 3.6.8 and 3.6.9 show overall performance on WSJ 23 and part of the Brown corpus from the CoNLL 2005 shared task. A comparison with other systems is shown in Section 3.6.10.

Argument Classification

As explained in Section 3.3.1, the goal of this task is to classify known arguments for each predicate. We use the proposition sets of the CoNLL2005 Shared Task and extract all arguments of each predicate as input to test the performance of a system. A

[2] http://www.lsi.upc.edu/~srlconll/

re-implementation of the system of Palmer et al. (2005) described in Section 3.3, with basic features, phrase type, path, head word, voice, and position is used as the baseline. The additional features and heuristics explained in Section 3.4 are also evaluated. The systems are given the exact semantic arguments for each predicate, and the task for each system is to find suitable semantic labels for each given argument but without identifying the arguments for each predicate.

This research uses the same probability estimation but modifies the probability estimation with additional features, distance and preposition as follows:

$$\begin{aligned}
&\mathrm{P}\,(r \mid hw, pt, pp, path, pos, dis, voice, predicate) = \\
&\quad \lambda_1{*}\mathrm{P}(r \mid pp, predicate) + \lambda_2{*}\mathrm{P}(r \mid pt, pp, predicate) + \\
&\quad \lambda_3{*}\mathrm{P}(r \mid pt, pp, path, predicate) + \lambda_4{*}\mathrm{P}(r \mid pt, pp, pos, dis, voice) + \\
&\quad \lambda_5{*}\mathrm{P}(r \mid pt, pp, pos, dis, voice, predicate) + \lambda_6{*}\mathrm{P}(r \mid hw, pp) + \\
&\quad \lambda_7{*}\mathrm{P}(r \mid hw, pp, predicate) + \lambda_8{*}\mathrm{P}(r \mid hw, pt, pp, predicate)
\end{aligned}$$

where $\sum_i \lambda_i = 1$ and $\lambda_i = \lambda_{i+1}$.

It uses the modified back-off lattice shown in Figure 3.7. There are two modifications. One is adding the “distance” feature, which is used in conjunction with the “position” feature, and the other is adding the preposition feature (denoted as “*pp*”) to each probability distribution.

Since there is more than one semantic role, the task of argument classification is a multi-class classification problem. Sections 3.6.1 to 3.6.6 contain detailed results for this multi-class classification problem.

Argument Identification

As mentioned in Section 3.3.2, the goal of this task is to identify arguments for each predicate from a sentence using syntactic information. This means that systems for this task are only concerned with finding correct semantic argument from a parse tree.

They do not need to take into account the kind of semantic labels for each argument. The classifiers for this task implement binary classification rather than multi-class classification.

For this research, the argument recognition of Gildea and Jurafsky (2002) was implemented with three features: path, head word, and predicate. All nodes in a parse tree can be considered as a Potential Argument, ("*pa*"), excluding punctuation and auxiliary words in the sentence. Section 3.3.3 has details and examples for this task.

Argument Identification and Classification

As mentioned in Section 3.3.3, this is the combination of the above two tasks, where the system first identifies probable constituents or phrases that represent semantic arguments of a predicate, and then assigns the most likely semantic labels. Features used in the three different tasks are listed in Table 3.7.

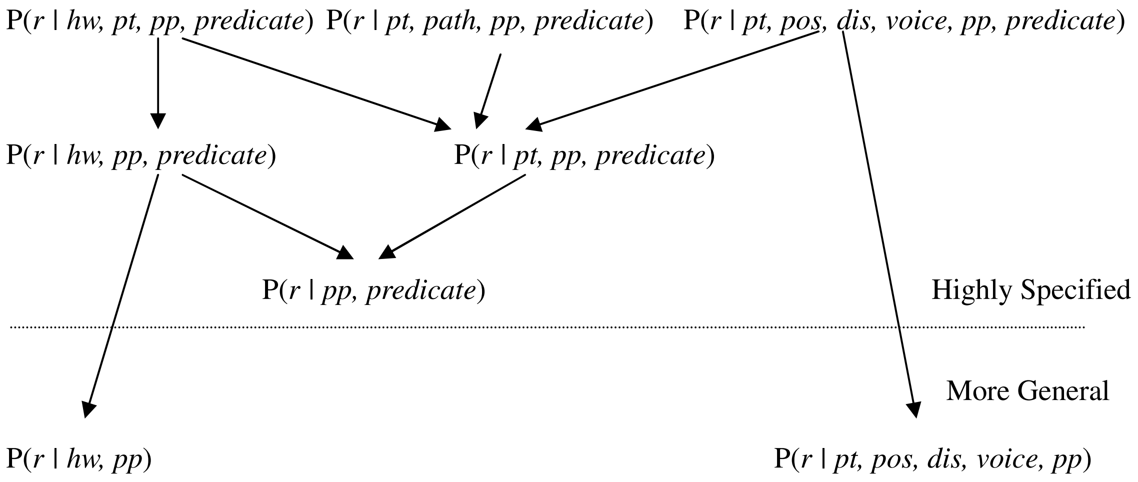

$$\begin{tikzcd}
P(r \mid hw, pt, pp, predicate) \arrow[d] \arrow[drr] & & P(r \mid pt, path, pp, predicate) \arrow[d] & P(r \mid pt, pos, dis, voice, pp, predicate) \arrow[dl] \arrow[dddd] & \\
P(r \mid hw, pp, predicate) \arrow[dr] \arrow[ddd] & & P(r \mid pt, pp, predicate) \arrow[dl] & & \\
& P(r \mid pp, predicate) & & & \text{Highly Specified} \\
& & & & \text{More General} \\
P(r \mid hw, pp) & & & P(r \mid pt, pos, dis, voice, pp) &
\end{tikzcd}$$

Figure 3.7. The modified back-off lattice from the Palmer et al. (2005) used for argument classification proposed in this thesis.

Task	Features
Argument identification	Path, Head word, Predicate
Argument classification	Head word, Phrase type, Path, Voice, Distance, Predicate, Preposition
Identification and classification	Head word, Phrase type, Path, Voice, Distance, Predicate, Preposition

Table 3.7. Featured used in different tasks.

3.6.1 Performance of the Palmer et al.'s system (2005)

There are 39,832 sentences in the training data with 3101 different verbs, 90,750 predicate-argument structures (propositions), 239,858 individual arguments, and 950,028 words (tokens). Table 3.8 shows the results of the baseline system on the task of argument classification. The overall performance is 79.43 in F_1 measurement and the performance scores for each role are also listed in Table 3.8.

Generally, the performance for core roles such as A0, A1 and so on, is better than for adjunctive roles such as AM-ADV, AM-CAU and so on. This suggests that it is harder to classify adjuncts than core roles. This is likely because there are more training examples found for core roles than for adjuncts. Another factor is that there is low performance on simple labels such as "AM-MOD" and "AM-NEG". Since the assignments for modals and negations are actually simple cases, the results show that the statistical approach makes errors in labelling these simple arguments by probability estimation. Systems such as the one used by Che et al. (2005) in the CoNLL 2005 shared task include corresponding simple heuristic rules in the classification stage. The system proposed in this thesis also applies rule-based pre-processing, shown in Section 3.6.2, for labelling modal, negation and discourse tags.

WSJ24	Precision	Recall	$F_{\beta=1}$
Overall	**79.23%**	**79.64%**	**79.43**
A0	86.60%	91.30%	88.89
A1	82.89%	90.78%	86.66
A2	72.35%	76.97%	74.59
A3	70.45%	54.39%	61.39
A4	56.63%	72.31%	63.51
A5	100.00%	50.00%	66.67
AA	0.00%	0.00%	0.00
AM-ADV	42.55%	28.67%	34.26
AM-CAU	20.00%	11.11%	14.29
AM-DIR	76.92%	55.56%	64.52
AM-DIS	77.40%	67.82%	72.30
AM-EXT	55.00%	39.29%	45.83
AM-LOC	51.28%	30.93%	38.59
AM-MNR	62.84%	38.43%	47.69
AM-MOD	84.32%	98.42%	90.83
AM-NEG	86.11%	59.62%	70.45
AM-PNC	65.45%	44.44%	52.94
AM-PRD	0.00%	0.00%	0.00
AM-TMP	64.96%	59.23%	61.97
R-A0	83.64%	94.52%	88.75
R-A1	84.38%	65.06%	73.47
R-A2	57.14%	80.00%	66.67
R-AM-CAU	0.00%	0.00%	0.00
R-AM-EXT	0.00%	0.00%	0.00
R-AM-LOC	75.00%	66.67%	70.59
R-AM-MNR	50.00%	16.67%	25.00
R-AM-TMP	74.29%	83.87%	78.79

Table 3.8. Results of argument classification by the baseline system (Palmer et al. 2005) for known arguments on WSJ 24.

3.6.2 Performance of Improvements

Table 3.9 shows the results of the improvements proposed in this thesis, including additional features and heuristics, on the task of argument classification. Compared to the baseline system by Palmer et al. (2005) in Table 3.8, the result shows an improvement of 6.16 in F_1. These improvements are gained using the new features and constraints explained in 3.4.1. The results are computed for auto (Charniak's) parses with known arguments, which means the system is given the correct arguments for each predicate in a tree. In other words, the performance of argument identification is 100%, and the testing task is only focusing on the accuracy of reclassifying these given arguments with the hand-corrected parses. Most of the results for the different roles increase except for four reference roles, “R-A0”, ”R-A2”, “R-AM-MNR”, and “R-AM-TMP”. This suggests that adding additional features does not guarantee an increase in performance for all semantic roles.

WSJ24	Precision	Recall	$F_{\beta=1}$	
Overall	**85.53%**	**85.65%**	**85.59**	
				to Baseline
A0	92.47%	91.49%	91.98	+ 3.09
A1	87.48%	92.92%	90.12	+ 3.46
A2	83.90%	80.53%	82.18	+ 7.59
A3	80.73%	77.19%	78.92	+17.53
A4	84.13%	81.54%	82.81	+19.30
A5	100.00%	50.00%	66.67	
AA	0.00%	0.00%	0.00	
AM-ADV	67.21%	58.78%	62.16	+27.90
AM-CAU	83.87%	57.78%	68.42	+54.13
AM-DIR	71.88%	63.89%	67.65	+ 3.13
AM-DIS	81.67%	72.77%	76.96	+ 4.66
AM-EXT	70.59%	42.86%	53.33	+ 7.50
AM-LOC	62.22%	72.16%	66.83	+28.24
AM-MNR	64.10%	51.65%	57.21	+ 9.52
AM-MOD	100.00%	99.68%	99.84	+ 9.01
AM-NEG	96.26%	99.04%	97.63	+27.18
AM-PNC	70.67%	65.43%	67.95	+15.01
AM-PRD	100.00%	66.67%	80.00	+80.00
AM-TMP	74.32%	72.71%	73.51	+11.54
R-A0	84.91%	92.47%	88.52	- 0.23
R-A1	84.06%	69.88%	76.32	+ 2.85
R-A2	42.86%	60.00%	50.00	-16.67
R-AM-CAU	0.00%	0.00%	0.00	
R-AM-EXT	0.00%	0.00%	0.00	
R-AM-LOC	100.00%	77.78%	87.50	+16.91
R-AM-MNR	50.00%	16.67%	25.00	
R-AM-TMP	65.00%	83.87%	73.24	-5.55

Table 3.9. Results of argument classification by the improved system for known arguments on WSJ 24.

3.6.3 Feature / Rule Performance & Analysis

Table 3.10 shows the effect on the argument classification task of adding different features individually to the system proposed by Palmer et al. (2005) again using known arguments. The preposition feature gives the most improvement over the baseline system and both the distance feature and the post-processing step have the second highest impact. The post-processing step, which mainly handles duplicate core roles as described in Section 3.4.1, shows its contribution to performance by an F_1:0.60 increase in Table 3.10. Rule-based pre-assignment for modal, negation, and discourse tags reveals improvement in performance especially on the modal tags. This pre-assignment also constrains the role classifier not to assign "AM-MOD" to any argument. This helps to increase the precision in the "AM-MOD" tags from 92.15% to 100.00%. Finally, the Actor heuristic exerts a small (0.36) contribution to the overall performance of the modified system.

After the addition of all the new features, performance from removing individual features is investigated. A degradation in performance is observed; evaluated by leaving out one feature or heuristic at a time.

Feature / Rule	**F_1**	**Contribution**
Baseline	79.43	
+ Actor heuristic	79.79	+0.36
+ Preposition	82.91	+3.48
+ Distance	80.03	+0.60
+ pre-assignment	80.00	+0.57
+ post-process	80.03	+0.60

Table 3.10. Effect of each feature and heuristic on the argument classification task when added to the baseline system.

Feature	F_1	Contribution
All	85.59	
- path	84.70	- 0.81
- position	85.59	
- distance	85.14	- 0.45
- preposition	81.34	- 4.25
- phrase type	85.25	- 0.34
- head word	81.90	- 3.69
- voice	85.11	- 0.48
- Actor heuristic	85.34	- 0.25
- Pre-processing	84.95	- 0.64
- Post-processing	85.53	- 0.06

Table 3.11. Performance of feature combinations for the task of argument classification on WSJ 24.

Table 3.11 shows the effect that each feature has on the argument classification task when it is deducted individually from the baseline system on WSJ 24. Removing the preposition feature for PP degrades the classification performance most. Instead of taking the preposition of a PP as the head word (as done in Palmer et al. 2005), we keep this preposition as a separate feature and add the head word of the NP after the PP to the role classification (as explained in Section 3.2), which helps to boost the overall performance. For example, the argument “in the school” with the preposition “in” and the head word “school” is easily labelled as a locative role. Apparently adding the preposition feature of a PP and using the head word in the NP after this PP as an additional feature is a good combination for the task of argument classification.

Removing the “head word” feature led to the second greatest decrease in performance.

It is because the head word plays a valuable role when recognizing adjunctive roles such as a temporal role with head words like week, yesterday, and so on. The position feature does not affect the overall performance when removed from the feature set, which implies that this feature can be skipped and is subsumed by the distance feature. Thus, the probability, P(*r* | *pt, pos, dis, voice, pp, predicate*) can be changed to P(*r* | *pt, dis, voice, pp, predicate*) without influencing the system's performance**.** The following formula is the probability distribution used in the rest of experiments with the modified statistical role classifier.

(F 3.3) $$\begin{aligned} P(r \mid hw, pt, path, dis, voice, predicate) = {} & \lambda_1 {*} P(r \mid predicate) + \lambda_2 {*} P(r \mid pt, predicate) + \\ & \lambda_3 {*} P(r \mid pt, path, predicate) + \lambda_4 {*} P(r \mid pt, dis, voice) + \\ & \lambda_5 {*} P(r \mid pt, dis, voice, predicate) + \lambda_6 {*} P(r \mid hw) + \\ & \lambda_7 {*} P(r \mid hw, predicate) + \lambda_8 {*} P(r \mid hw, pt, predicate) \end{aligned}$$

where $\Sigma_i \lambda_i = 1$ and $\lambda_i = \lambda_{i+1}$, and "*r*" denotes semantic roles (such A0, A1, AM-TMP, and so on).

3.6.4 Learning Curves

What is the effect on performance when using different amounts of training data? Table 3.12 shows the performance for different sizes of training data again, with known arguments, meaning that systems are given the correct arguments to classify. D*n*K in Table 3.12 denotes the first *n* k (i.e. *n*-thousand) sentences from the training data sets, WSJ 02 to 21, for example, D10k denotes the first 10k sentence from the training datasets. Using the whole training data (WSJ 02-21), the overall performance increases by about 9 in F_1 measurement when compared to using D1k only. *Upper_bound* is the proportion of arguments which are found in the system's probability distribution before selecting the maximum label. For example, an argument with its probability distribution can be as follows: "A1":0.65, "A2": 0.20, "AM-ADV":0.15, and others:0.00. Assume the correct label is "A2", but the system chooses "A1" due to the strategy of highest probability. In this case, the argument is counted as a wrong role when computing the F_1 measurement but as a correct role when computing *Upper_bound*.

Sentences	**Precision**	**Recall**	**F_1**	**Upper_bound**	**T (sec/s)**
D1k	77.17	75.71	76.44	92.19%	0.259
D5k	81.63	81.19	81.41	96.01%	0.478
D10k	82.78	82.55	82.67	97.15%	0.564
D20k	84.36	84.22	84.29	97.60%	0.652
D30k	85.11	85.05	85.08	97.98%	0.712
D40k	85.53	85.65	85.59	98.14%	0.785

Table 3.12. Performance results for the development dataset (WSJ 24) using different sizes of training data.

Upper_bound shows how much coverage (or recall) a system can achieve when selecting the right label among all possible roles in the probability table. In Table 3.12, the "*Upper_bound*" for D1k is 92.19%, which means 92.19% of the correct roles can be found during probability estimation. But the "Recall" is only 75.71. This means only about 82% of the cases are right using this system.

Additionally, Table 3.12 illustrates the average processing time (T) in seconds per sentence during the testing process. As training data increases, the average time for each sentence also increases. The average processing time with D40k is 0.785 second per sentence.

Although the system is given semantic arguments before classification, there are some unlabelled arguments and wrong continuation roles such as "C-A0", which causes Recall to be different from Precision. For example, if the correct label set is [A0 V

C-A0], the evaluated set is [A0 V A1]. It seems there is one correct label "A0" (excluding the predicate labelled as "V") but this A0 must be joined with the continuation label "C-A0" as one label according to the evaluation script used in the CoNLL 2005 shared task. Therefore, the number of correct labels is zero

Table 3.12 also shows that the improvement in performance is not proportional to the size of training data. Figure 3.8 show a learning curve characterised by the logarithmic formula: y = 2.4578Ln(x) + 59.87, where 'y' denotes F_1 measurement, and 'x' the size of training data. The six actual data points from Table 3.12 are also shown.

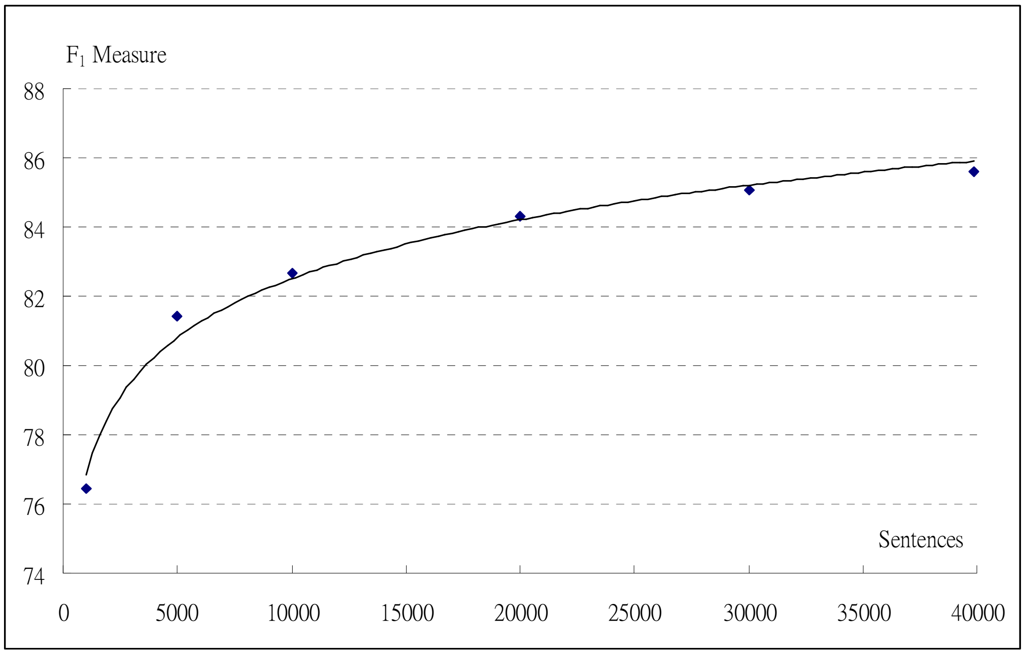


Figure 3.8. Learning curve for different sizes of training data.

3.6.5 Execution time for Classification

Time complexity is used for measuring how classification time varies as the size of the input increases. Figure 3.9 shows that the classification time exhibits logarithmic characteristics (y = 0.1606Ln(x) - 0.9324 , where 'y' denotes classification time in seconds per sentence, and 'x' denotes the size of training data), implying that the computational complexity for the modified statistical role classifier (M_SRC) (based on the Palmer et al. 2005) is logarithmic, such that $T_{M\_SRC} = O(m * \log n)$ where *m* is the number of features and *n* denotes the size of the training data.

The shape of the curve is a result of the back-off lattice described in Section 3.3.1, for which Figure 3.7 shows two types of searching: highly specified and more general. In Figure 3.7 the highly specified search means probability distributions are just based on training data containing a specific predicate without searching all training data. This specified calculation includes computation of P(*r* | *hw, pt, pp, predicate*), P(*r* | *pt, path, pp, predicate*), P(*r* | *pt, pos, dis, voice, pp, predicate*), P(*r* | *hw, pp, predicate*), P(*r* | *pt, pp, predicate*), and P(*r* | *pp, predicate*). These probability distributions include the predicate feature in their feature sets. The more general search (used for probability distributions without the predicate feature) is applied for P(*r* | *hw, pp*) and P(*r* | *pt, pos, dis, voice, pp*).

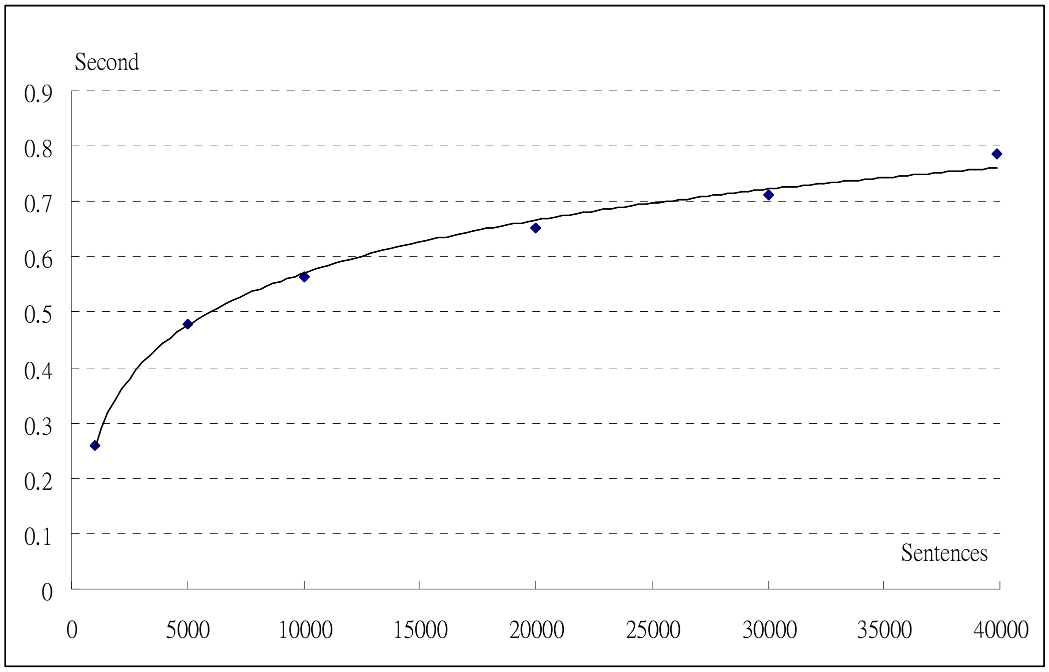


Figure 3.9. Classification time relative to the size of the training data.

When highly specified searching finishes the calculation (or finds a solution for role classification), there is no further calculation required for general searching, and this results in logarithmic complexity. This logarithmic complexity provides practical and acceptable execution time as the training set size increases.

Results of execution time in Table 3.12 are based on machines with P4 3.0G CPU, 1G RAM, and Linux platforms.

3.6.6 Argument Identification on WSJ 24

Here we use the proposition sets of the CoNLL2005 shared task and extract all arguments of each predicate as input to test the argument identification accuracy of the systems. We re-implement the statistical argument recognizer (based on Palmer et al., 2005) described in Section 3.3.2. Table 3.13 lists results for argument identification with gold (hand-corrected) parses based on different sizes of training data. Although the Recall increases as the training data increases, the Precision decreases. The F_1 measurement for all training data does not appear much improved compared to the one based on one section of training data D1. This reveals that the increasing size of training data does not create a significant improvement for argument identification.

Training	**Precision**	**Recall**	**F_1**	**T (sec/s)**
D1	91.98	86.35	89.08	0.358
D4	91.57	88.38	89.95	1.603
D20	90.59	89.31	89.95	5.643

Table 3.13. Argument identification with gold parses before role classification.

Table 3.14 shows results for argument identification based on auto parses. The overall results compared to those with gold parses in Table 3.13 drop by about 12 points in F_1 measurement. This shows that there is a real need for a state-of-the-art parser to achieve good performance in argument identification. The data in Table 3.14 also demonstrate that the size of training data does not significantly influence the performance of argument identification. But the execution time of argument identification increases significantly when training data increases. The effect appears to be linear. Meanwhile, the execution time for argument identification is about seven times that of role classification. Consequently Chapter 4 will introduce a faster approach for argument identification.

Training	Precision	Recall	F_1	T (sec/s)
D1	79.74	75.10	77.35	0.361
D4	79.68	76.65	78.14	1.589
D20	78.84	77.58	78.21	5.582

Table 3.14. Argument identification with auto parses (Charniak's) before role classification.

3.6.7 Gold parses vs. Auto parses

Table 3.15 shows the performance on argument identification and classification investigated in this chapter, using the PropBank corpus with gold-parses or "hand-corrected" parses, (i.e. the parses from the hand-corrected Penn TreeBank), and Table 3.16 shows the results for auto parses (Charniak's). Results for argument identification (ID), and the combination of argument identification and classification are indicated by "ID", and "ID + Classification" respectively.

Task with gold parses	P	R	F	T
ID	90.59	89.31	89.95	5.643
ID + Classification	78.14	76.46	77.29	6.342

Table 3.15. Performance for unknown arguments with gold-standard parses on WSJ 24.

Task with auto parses	P	R	F	T
ID	78.84	77.58	78.21	5.582
ID+Classification	69.70	68.03	68.86	6.382

Table 3.16. Performance for unknown arguments with auto (Charniak's) parses on WSJ 24.

Looking at these two tables (3.17 and 3.18), performance for auto parses in Precision (P), Recall (R) and F_1 measurement (F) decreases when compared to the results for gold parses. Compared to using gold parses, the overall performance measurement drops by about 10 points in F_1 when using auto parses. The average execution time with the 20 training datasets is similar for gold parses and auto parses, being about 6.3 seconds per sentence.

3.6.8 Overall Performance on WSJ 23

This subsection concerns the overall performance on WSJ 23, which can provide results for comparison with related research. Table 3.17 shows the performance of the F_1 measurement of semantic role classification for known arguments (when constituents that are semantic arguments are given) using gold-standard parses. The overall performance improves more than 6.0 in F_1. The contribution for each feature is similar to Table 3.11 shown in Section 3.6.3.

System on WSJ 23	F_1
Palmer et al., (2005)	80.47
Improvements	86.55

Table 3.17. Performance of Semantic Role Labelling for known arguments with gold parses.

Table 3.18 shows the results for argument identification and classification with Charniak's parses on WSJ 23. The system is not given any arguments and tries to recognize all possible arguments and label them with semantic roles. Table 3.18 shows that the overall performance for argument identification (ID) is F_1: 79.94 which is about 1.7 better than the one on WSJ 24. The overall performance on WSJ 23 increases F_1:2.18 compared to WSJ 24.

Task on WSJ 23	**P**	**R**	**F**
ID	80.10	79.78	79.94
ID+Classification	71.18	70.90	71.04

Table 3.18. Performance for unknown arguments with auto parses (Charniak's) on WSJ 23.

Table 3.19 shows the results on WSJ 23 by Palmer et al. (2005) and the improved system based on Palmer et al. (2005).

System	**P**	**R**	**F**
Palmer et al. (2005)	68.60	57.80	62.74
Palmer et al. (2005) – with Charniak's parses	68.04	65.89	66.95
The improved system	71.18	70.90	71.04

Table 3.19. Performance for different systems on WSJ 23.

We re-implement the system of Palmer et al. (2005) using the test data with Charniak's parses from the CoNLL 2005 shared task. Compared to Palmer et al. (2005) with Collins' parses, the modified version has an F_1: 4.21 increase. This modified system further improves about F_1:4.1 than the re-implemented system by additional features (preposition and distance) and Actor constraint, Pre-processing and Post-processing described in Section 3.4.

Table 3.20 shows details of performance of this system including performance of precision, recall, and F_1 measurement in each semantic role and overall. Generally, performance for core roles is better than that for adjunctive roles. The reason for better performance in core roles is that there are more training examples for core roles than adjuncts for each predicate.

WSJ24	**Precision**	**Recall**	$\mathbf{F}_{\beta=1}$
Overall	71.18%	70.90%	71.04
A0	91.75%	84.28%	87.86
A1	65.60%	72.94%	69.08
A2	66.22%	61.98%	64.03
A3	66.22%	56.65%	61.06
A4	62.16%	67.65%	64.79
A5	100.00%	40.00%	57.14
AM-ADV	49.17%	41.11%	44.78
AM-CAU	62.07%	49.32%	54.96
AM-DIR	46.91%	44.71%	45.78
AM-DIS	34.36%	38.44%	36.28
AM-EXT	71.43%	46.88%	56.60
AM-LOC	55.29%	50.41%	52.74
AM-MNR	50.32%	45.64%	47.87
AM-MOD	98.67%	94.56%	96.57
AM-NEG	95.18%	94.35%	94.76
AM-PNC	38.94%	38.26%	38.60
AM-PRD	50.00%	20.00%	28.57
AM-REC	0.00%	0.00%	0.00
AM-TMP	63.38%	58.60%	60.90
R-A0	73.28%	85.71%	79.01
R-A1	66.90%	60.90%	63.76
R-A2	60.00%	37.50%	46.15
R-AM-CAU	33.33%	25.00%	28.57
R-AM-EXT	0.00%	0.00%	0.00
R-AM-LOC	66.67%	47.62%	55.56
R-AM-MNR	25.00%	33.33%	28.57
R-AM-TMP	59.70%	76.92%	67.23

Table 3.20. Results on WSJ23 for argument identification and classification.

3.6.9 Overall Performance on the Brown corpus

This section details the overall performance on part of the Brown corpus, which can provide results to compare with related research. Table 3.21 shows that on the task "ID + Classification", there is an approximately F_1:11 degradation compared to that of WSJ 23 in Table 3.18. Section 2.5.5 describes the reason for this decrease as being due to processing propagating errors in Natural Language Processing applications when moving to different domains.

Task on Brown corpus	**P**	**R**	**F**
ID	77.20	74.18	75.66
ID+Classification	61.62	59.21	60.39

Table 3.21. Performance for unknown arguments with auto (Charniak's) parses on part of Brown corpus.

Table 3.22 shows the comparison of this modified system with the re-implemented one from Palmer et al. (2005) using the CoNLL test data with the Charniak's parses. There is a gain of about 4 points in F_1 by additional features and heuristics than the re-implemented system when moving from the WSJ domain to the Brown corpus domain.

System	**P**	**R**	**F**
Palmer et al. (2005) – re-implemented	59.14	53.93	56.42
The improved system -	61.62	59.21	60.39

Table 3.22. Performance for different systems on the Brown Corpus.

3.6.10 Comparison with Different Systems

Table 3.22 shows the results on WSJ 23 for the task of argument classification only using different classifiers, Support Vector Machines (SVM), Decision Tree, and Gildea and Palmer's system (G & P) with the basic features reported in Pradhan (2004). The system investigated in this chapter can obtain 80.47 in F_1. With the extra two features (distance and preposition) and the proposed heuristics, this system can reach 86.55 in F_1, which is 1.45 behind the SVM. Although this statistical approach does not perform as well as the SVM, it is competitive. More improvements will be described in Chapter 4 for argument identification and in Chapter 5 for argument classification.

System	Accuracy (F_1)
SVM	88
Decision Tree	79
G & P	77

Table 3.23. Comparison with different systems using the same features reported by Pradhan, (2004).

3.7 Summary and Remarks

A statistical approach to Semantic Role Labelling (SRL) was introduced by Gildea and Jurafsky (2002). Since then, many new approaches have been proposed. SRL is divided into two sub-problems, argument identification and argument classification. Features for argument identification used in this system are head word, path, and predicate. Three extra features, phrase type, voice and preposition, are used for role classification. Statistical classifiers (like the one proposed by Palmers et al., 2005) make good use of these features extracted from parses and produce 80.47 in F_1 measurement on WSJ 23 with known arguments. This research adds some improvements including additional features, an actor heuristic, pre-assignment for modal, discourse, and negation tags, and post-processing mainly for duplicate roles. These improvements help to increase the performance of the baseline approach from Palmer et al. (2005) by about 6.0 in F_1 measurement. Data used in these experiments are from the CoNLL 2005 shared task including a formal evaluation script, *srl-eval*, available from the CoNLL's website.

When looking at the contribution of the proposed features, it can be seen that the preposition feature influences the overall performance significantly when it is removed from the system. The head word is the second most influential feature, but gives less coverage, as also mentioned in Gildea and Jurafsky (2002). The size of training data has a big impact on role labelling but far less so on argument identification. One factor in this is probably that argument identification is a binary classification but role classification is a multi-class problem. There are enough training data for argument identification but sparseness of data stands out when moving to multi-classification. Of note here is that adjunctive roles reveal more shortage of training data than core roles. Execution time is also considered in this research. Illustration of learning curves when using different sizes of training data graphically reveals the logarithmic computational complexity for the back-off lattice. Experimentally, execution time for argument identification is about seven times slower than that for role classification.

Gold (hand-corrected) parses provide a 12 point (F_1 measurement) improvement in system performance compared to auto parses (Charniak's), revealing accumulating errors in natural language processing applications. The final result using the modified statistical system on WSJ 23 for standard comparison in SRL is 71.04 in F_1 measurement, evaluated by the Perl script from the CoNLL 2005 shared task. This is an improvement of about 9 points compared to the baseline system proposed by Palmer et al., (2005). When moving from the WSJ domain to a different domain such as the Brown corpus, system performance drops by about 3 points for argument identification and 10 points for the combination of argument identification and argument classification. Pradhan et al. (2004) also shows the degradation in a different domain.

Although the modified system improves on the performance of Palmer et al. (2005), there is still room for improvement, as can be seen by comparing the performance of this system (F_1:71.04) to that of Surdeanu and Turmo (2005) (F_1:76.46) using Charniak's parses. This insufficiency in performance provides room and motivation for more research in SRL as far as the problem of argument identification and argument classification is concerned.

Chapter 4 will introduce a new pre-processing algorithm, called Predicate-Argument Recognition Algorithm (PARA), for argument identification to reduce computational time without sacrificing system's performance. Chapter 5 will introduce a new methodology called Pattern-Matching (PM) for Multi-Argument Classification (MAC) which estimates all arguments of a predicate rather than individual arguments. The combination of PARA and PM together with the modified role classifier described in this chapter not only outperforms systems that use the same syntactic information in the CoNLL 2005 shared task but also provides a competitive performance compared to the best one.

Chapter 4

Predicate-Argument Recognition

We feel that our results show that statistical parsers, although computationally expensive, do a good job of providing relevant information for semantic interpretation. Not only the constituent structure but also head word information, produced as a side product, are important features. Parsers, however, still have a long way to go. Our results using hand-annotated parse trees show that improvements in parsing should translate directly into better semantic interpretations.
Gildea and Palmer (2002), *The Necessity of Parsing for Predicate Argument Recognition*. (page 246)

Machine learning methods are more or less universal in existing algorithms for performing argument identification or Predicate-Argument Recognition as the first stage in Semantic Role Labeling. This chapter demonstrates how a heuristic algorithm for Predicate-Argument Recognition (PARA) offers competitive performance compared to ML-based methods by naively assuming a direct mapping from parse trees to unlabelled semantic argument assignments. The parse trees are obtained from commonly-used, state-of-the-art parsers such as those developed by Charniak or Collins. When tested on the WSJ 23 sample from the CoNLL 2005 Shared Task, this training-free algorithm achieves an F_1 measure of 82.60 for argument identification in the CoNLL 2005 Shared Task when using the same syntactic information. Moreover, the execution time for each sentence is 0.002 seconds, which is much faster than Machine Learning approaches. This chapter further proposes other improvements for existing argument recognizers, and demonstrates how such improvements help to boost overall performance for an existing role classifier (Palmer et al., 2005). As PARA can achieve promising results on this problem, it may prove useful for other NLP tasks such as Information Retrieval, Question Answering, Machine Translation, and so on.

Section 4.1 outlines the background of the problem of argument identification. Section 4.2 describes related work on semantic argument identification, and Section 4.3 details the algorithm for Predicate-Argument Recognition. The data, evaluation software, and results are presented in Sections 4.4 and 4.5. Section 4.6 provides some conclusions, remarks and a summary for this chapter.

4.1 Introduction

The CoNLL 2005 Shared Task attracted extensive attention and the results given in the proceedings list a wide range of different ML approaches to the problem of Semantic Role Labeling (SRL). Most of those systems separated SRL into two parts: semantic argument identification, followed by role classification for the bounded arguments. That is, given the parse tree for a sentence, all arguments of predicates or verbs in a sentence must be identified and subsequently annotated with their respective semantic roles. Thus, the first task is simply to find the arguments of the predicate. This is the so-called semantic argument identification or Predicate-Argument Recognition problem. For example, the sentence “The dog bites a tennis ball” can be segmented into three basic syntactic constituents: a noun phrase (the dog), a verb (bites), and another noun phrase (a tennis ball). Once isolated, these arguments are subsequently labeled with appropriate semantic roles, such as “Agent”, “Verb”, and “Patient” respectively.

Predicate-Argument Recognition is important for Semantic Role Labeling (Baldewein et al., 2004). It is part of the problem of Semantic Role Labeling, in that automatic and exact recognition of all arguments for all predicates in a sentence can help to increase the performance of a Semantic Role Labeling classifier. Many researchers tackled Predicate-Argument Recognition using Machine Learning approaches. ML techniques that have been applied are the SNoW learning architecture (Punyakanok et al., 2004), Support Vector Machines (Moschitti et al., 2004; Park et al., 2004; Pradhan et al., 2004; Pradhan et al., 2003), Perceptron Learning (Carreras et al., 2004), EM-Based Clustering (Baldewein et al., 2004), Transformation-based Learning (Higgins, 2004), and Memory-Based Learning (Kouchnir, 2004). Their common characteristic is to retrieve features such as phrase type, path, voice, and so on, from a sentence in a chunking-based or tree-based format, then find arguments from all possible argument candidates for a specific predicate via Machine Learning. The systems are given predicates and their locations in the sentences as part of the input. By tracing the syntactic parses and extracting related features, they proceed to recognize all possible arguments related to the given predicates. For example, for the predicate "play" shown in Figure 4.1, a system can check each node or constituent in the parse tree and then recognize a valid NP argument, "he", and a valid PP argument, "with a dog". The two valid arguments are a NP "He" and a PP "with a dog". The rest nodes, which are S, the "IN" preposition and the NP "a dog", are not valid arguments according to PropBank as it is described in Chapter 2.

On the other hand, if the systems were not given all predicates, the additional task would be to find all possible predicates before any further steps for argument identification can be performed. In the CoNLL 2005 shared task, all predicates are given as input, but in a real case, targets or predicates are not always given.

In the last paragraph of Gildea and Palmer (2002), the authors mention the necessity of parsing for Predicate-Argument Recognition, which was implicitly revealed in the CoNLL 2004 shared task (Baldewein et al., 2004). This chapter provides evidence for the hypothesis, stated in Chapter 1, that there exists a direct mapping from syntactic structure to unlabelled semantic structure or semantic arguments in tree-based parses. This is implemented via a Tree-based Predicate-Argument Recognition algorithm (PARA). The algorithm, PARA, can convert syntactic parses to semantic arguments directly without training, which is different to conventional ML approaches to this problem.

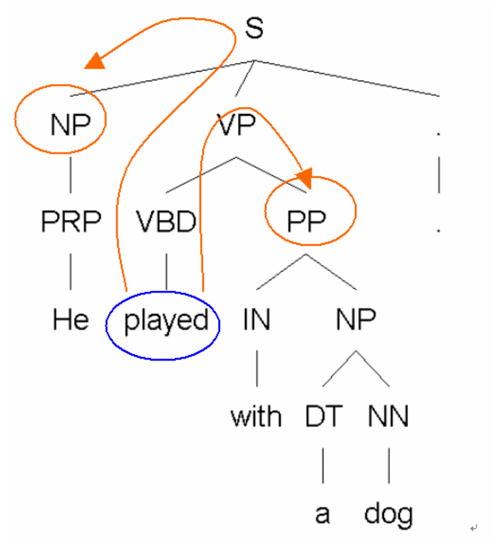


Figure 4.1. Illustration of an argument recognizer that checks all possible arguments when processing the predicate “play”.

4.2 Related Work

As described in Chapter 1, Semantic Role labeling is to find all arguments for all predicates in a sentence, which is done by argument recognizers, and forward these identified arguments to an argument classifier for role classification according to semantic roles such as AGENT, THEME, TIME, etc.; more commonly and abstractly referred to as A0, A1, AM-TMP and so on. Existing ML classifiers from the CoNLL 2005 Shared Task demonstrated many possible solutions.

4.2.1 A Conventional Argument Recognizer

Conventional argument recognizers, such as the one developed by Gildea and Palmer (2002), take all nodes in a parse tree, including each word in a sentence, as potential arguments. Whether a potential argument is classified as a valid semantic argument depends on a probability estimation such as that given by Maximum Likelihood Estimation (Gildea and Palmer, 2002) or similar. Such a recognizer is a binary classifier, utilizing the distribution observed in the training data to learn how to predict future novel semantic arguments. Information from a parse tree is forwarded as features to the argument recognizer to help formulate a model to make correct predictions. The most-frequently used features for semantic arguments are the Path, Headword, and Predicate itself, as summarized in Table 4.1.

A statistical argument recognizer from Palmer et al. (2005) utilizes the estimation formula (F 3.2), from Section 3.3.2 specifically:

$$P(pa| path, hw, predicate) = \lambda_1 * P(pa \mid path) + \lambda_2 * P(pa \mid path, predicate) + \lambda_3 * P(pa \mid hw, path)$$

where $\sum_i \lambda_i = 1$.

Predicate (*pr*) – The given predicate lemma (an uninflected, untensed verb).
Path (*path*) – The syntactic path through the parse tree from the constituent to the predicate being classified
Head Word (*hw*) – The syntactic head of the phrase. (The head is normally simply the last noun of the rightmost subordinate noun phrase).

Table 4.1. Features used in semantic argument identification.

This statistical argument recognizer was implemented for the research presented in this thesis as a means to compare a conventional ML recognizer with the heuristic direct-mapping algorithm proposed here. Details of this statistical argument recognizer are described in Chapter 3.

4.2.2 Pruning Strategy

Traditional argument recognizers have to spend time on each phrase and word to find possible semantic arguments. For example, consider the following sentence with its graphic tree parse shown in Figure 4.2:

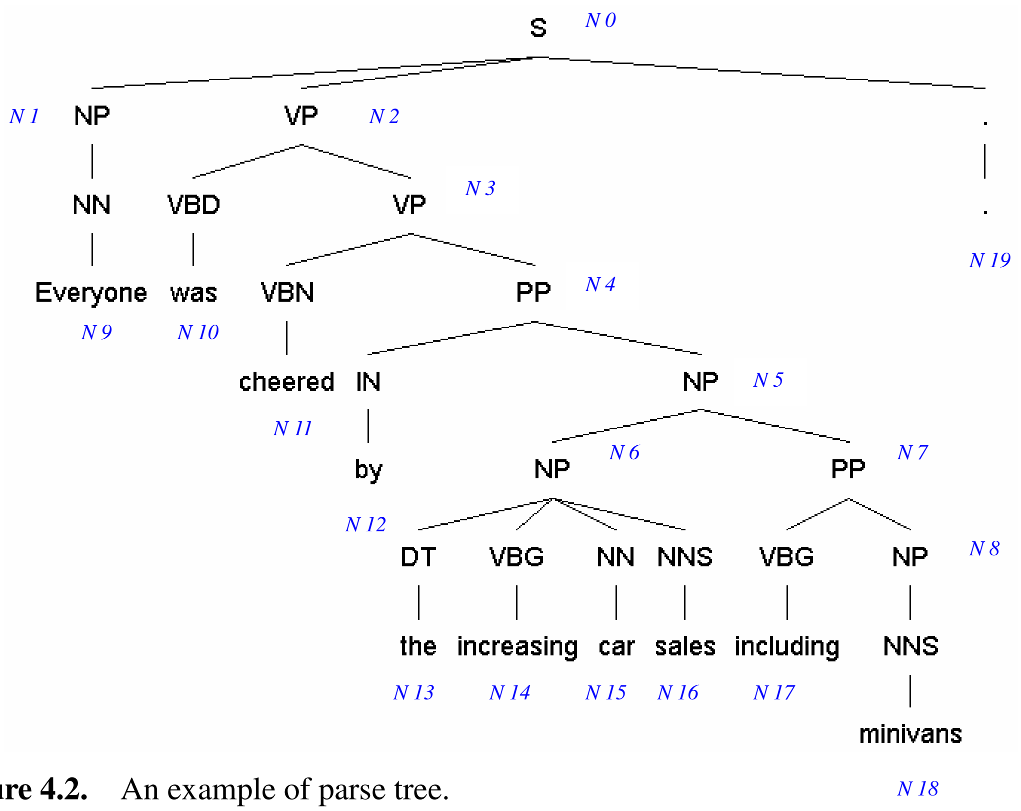


Figure 4.2. An example of parse tree.

(Ex. 4.1) "Everyone was cheered by the increasing car sales including minivans."

There are three predicates: "cheer", "increase", and "include" with 9 phrases (N 0 to N8) and 11 leaves (N9 to N19). Traditional argument recognizers need to check these 20 constituents for each predicate to identify semantic arguments.

In order to reduce computational time, Xue and Palmer (2004) describe a pruning strategy by filtering out constituents that are clearly not semantic arguments to the predicate. Then they classify the candidates derived from the pruning strategy as either semantic arguments or non-arguments. Finally they run a role classifier to label the constituents that are identified as arguments with semantic roles (Xue and Palmer, 2004).

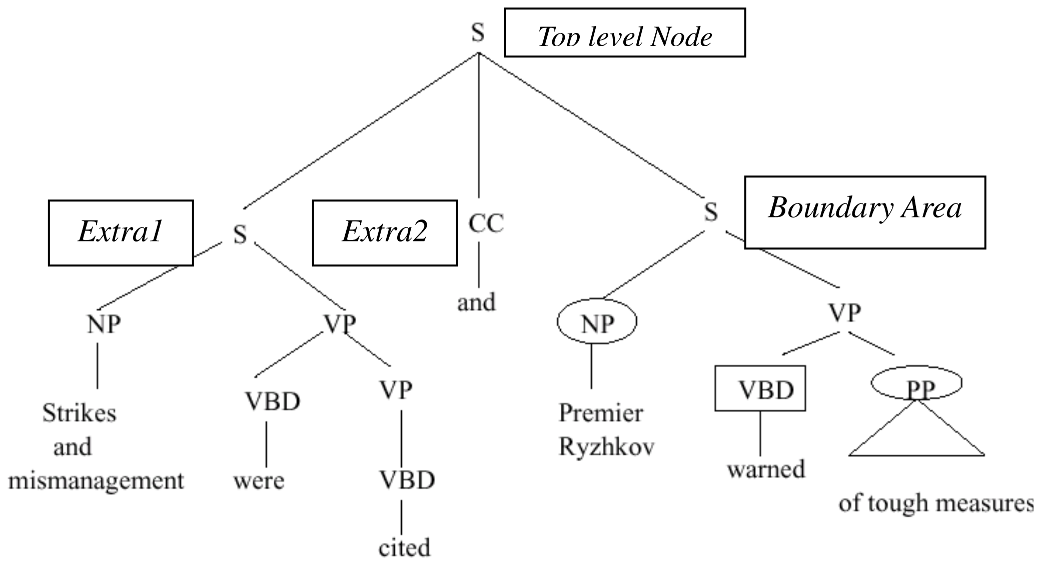


Figure 4.3. The example of the Pruning Strategy reproduced from Xue and Palmer (2004).

The pruning algorithm consists of two steps. Step 1 is to designate the predicate as the current node and collect the sisters or constituents at the same level as the predicate. If a sister is a PP, also collect its immediate children. The second step is to reset the current node to its parent and repeat Step 1 until the process reaches the top level node (Xue and Palmer, 2004). Figure 4.3 shows an example from Xue and Palmer, (2004). By this pruning algorithm, argument recognizers can focus on the possible argument candidates , which are a circled PP, a circled NP, S with the "*Extra1*" label, and CC. There are still two extra arguments included denoted as *Extra 1* and *Extra 2*. Such extra arguments can be avoided by confining the searching area under the nearest S clause denoted as *Boundary area.*

Going back to Example 4.1, there are three predicates, "cheer", "increase", and "include" with 9 phrases (N 0 to N8) and 11 words (N9 to N19). Using the pruning strategy, the possible arguments for "cheer" are *N4, N12, N5, N10, N1, N19* when tracing the algorithm upward to the top level node "N0", for "increase", *N13, N15, N16,* *N7, N17, N8, N12, N11, N10, N1, N19,* and for "include", *N8, N6, N12, N11, N10, N1, N19.* The numbers of possible arguments for each predicate are 6, 11, and 7, compared to 20 for each predicate with no pruning. Using the pruning strategy can save about 60% of the computation for this sentence.

This pruning strategy has been widely used in some systems in CoNLL2005 like Punyakanok et al., (2005); Tsai et al., (2005); Yi, et al., (2005); and Cohn et al., (2005) in order to reduce training and testing time for SRL systems. For example, results in Tsai et al. (2005) show this pruning strategy helps systems to eliminate large portion about 61% of the training data, while it does not sacrifice system's performance. Tsai et al. (2005) claim their systems with the pruning strategy achieve 93% of the correct arguments in training sets.

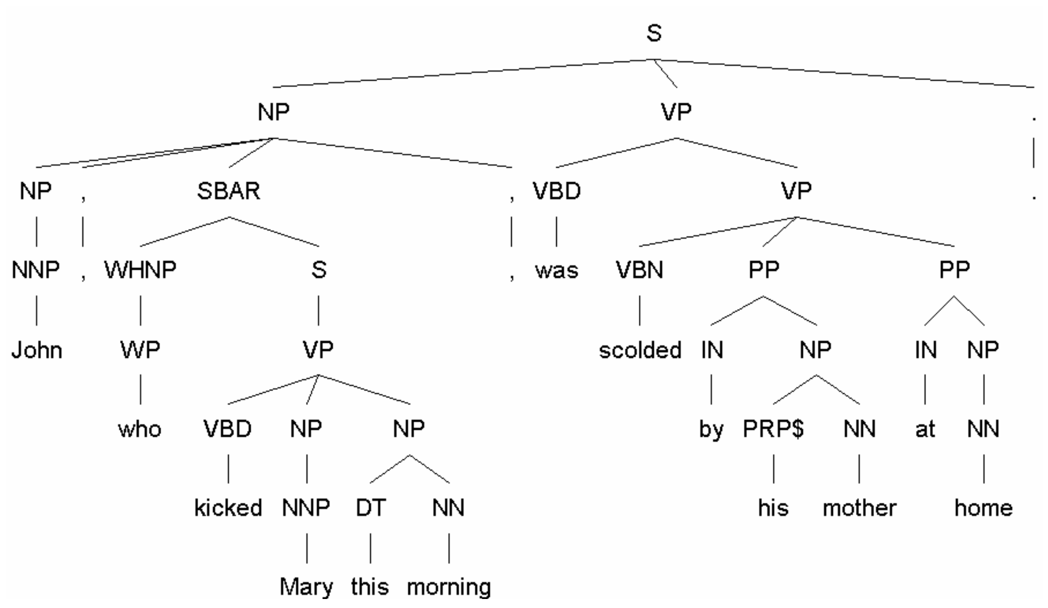


Figure 4.4. Graphic illustration of (Ex. 4.2).

4.2.3 Motivation

The pruning strategy of Xue and Palmer (2004) helps to cut down invalid nodes without sacrificing coverage but there are two ways to improve this strategy. One is, if we define the search area as under the closest S clause instead of the top level node, the invalid nodes will be reduced more, like two arguments (*Extra1* and *Extra 2*) in Figure 4.3. By cursory observation, most of all valid arguments are under the closest S clause except some cases like no subjects inside the closest S clause or a SBAR parent of the closest S. Methods for this problem are described in Section 4.3. The other point we find is a special case, an argument with several individual words for a predicate in NP.

The following sentence (Ex. 4.2) with its graphic tree parse shown in Figure 4.4 illustrates the first point.

(Ex. 4.2) “John, who kicked Mary this morning, was scolded by his mother at home.”

In the pruning strategy by Xue and Palmer (2004), the top level node for the predicate “kick” in (Ex. 4.2) is the S clause at the root in Figure 4.4. This pruning strategy includes another invalid constituent, VP under the top S clause.

The other case is for the predicate “increase” in Figure 4.2. According to the CoNLL 2005 shared task, the valid arguments for the predicate “increase” in Figure 4.2 are “increase” and “car sales”. Excluding the predicate itself, the constituent “car sales” is the only valid argument, which is not included by this pruning strategy either. Instead of combining two words “car sales” as one valid argument, the pruning strategy collects two separate arguments because “car” and “sales” are sisters of the predicate. We will develop a rule-based algorithm that can be applied to solve the two problems identified above. They are arguments outside the top level node, and arguments of a predicate in a NP with more than one leaf.

Another concern for this pruning strategy is that it searches the same top level node for all predicates. Such a general method can be further decomposed to different categories to reduce computation time. For example, in Figure 4.2, the closest phrases to the predicates “cheer”, “increase”, and “include” are VP, NP, and PP respectively. For the predicate “cheer” under VP, all valid or correct arguments appear under the S clause. For the predicate “increase” under NP, there are no other valid arguments appearing outside this NP. For the predicate “include” under PP, there exists a valid argument NP appearing outside this PP. The modified pruning strategy is thus as follows. The top level node for a predicate under VP is the closest S clause which is the same with the original pruning strategy. The top level node for a predicate under NP is the NP itself. The top level node for all other phrases except VP and NP phrases is the phrase itself.

The following section introduces a direct mapping algorithm that maps from syntactic parses to semantic arguments. This algorithm is called Predicate-Argument Recognition Algorithm (PARA). It is based on three different types of boundary areas, employs a pruning strategy and heuristic rules to search valid arguments outside boundary areas and performs post-processing to correct punctuation errors generated by parsers.

4.3 Methodology of PARA

This section presents the Tree-based Predicate-Argument Recognition Algorithm (PARA) that is designed to achieve the following goals:

- find all predicates (if not given);
- find all arguments of the given or identified predicates;
- label each argument with information such as phrase type, voice (active or passive) and so on; and
- arrange output in a flat format as established for the shared tasks of CoNLL2004 and CoNLL2005.

PARA is composed of four steps for an input sentence with parses; i) create a predicate list; ii) find all arguments for each predicate; iii) if needed, label each argument with basic features; and iv) output arguments. This procedure is illustrated in Figure 4.5 below.

Procedure PARA

```
    Input a sentence in tree-based format (tree)
    Create predicate list by adding given predicates if given or
    Add all words with verbal POS format, such as VBZ,VNG, and so on, to the predicate list.
    For each identified or given predicate, predicate,
        Create argument list // Argument_list[predicate]= Create_arg_list(predicate, tree)
        Add information for arguments // Info_labeling(Argument_lists, predicate_list)
    Output Argument_lists
```

Figure 4.5. Predicate-Argument Recognition algorithm.

```
Create_arg_list(predicate, tree)
If the closest phrase type including the predicate == VP
    - find the boundary area ( the closest S clause)
    - find if SBAR as the parent of the closed S clause
            - find if WHNP in siblings within the boundary area, //what, which, that, who,…
                    - if found then add WHNP to argument list
                    - if the first word of the WHNP is not "what" then find the closest NP
                      from ancestors, add the NP to the argument list and mark this WHNP as
                      a reference of NP.
            - else // there is no WHNP in siblings
                    - find the closest NP from ancestors, and add the NP to the argument list.
    - add to argument list all validated upper-most arguments without including predicates in
        the child VP phrase of the S clause.    // the same with the pruning algorithm
    - Further check if there is no NP before the predicate, then find the closest NP from
        Ancestors.
If the closest phrase type including the predicate == NP
    - find the boundary area (the NP clause)
    - find RB(POS) before predicate and add to argument list.
    - Add this predicate to argument list.
    - Add the rest of word group after the predicate and before the end of the NP clause as a
        whole argument to argument list.
    - If there is no argument after the predicate, then add to argument list the constituent
        after the NP predicate.
Otherwise (PP and others)
    - find the boundary area ( the phrase itself)
    - find the closet NP from ancestors if the lemma of the predicate is "include", and add
        this NP to argument list.
    - Add this predicate to argument list.
    - Add the children of this predicate to argument list or add one closest NP outside the
        boundary area to argument list if there is no child for this predicate.
```

Figure 4.6. Algorithm of Create_arg_list for PARA.

The main part of PARA is the function used to identify all arguments or constituents in a parse. There are three cases in the function of "Create_arg_list" outlined in Figure 4.6, which are explained in the following sections, 4.3.1, 4.3.2, and 4.3.3.

4.3.1 Three categories and their Boundary areas

There are several types of phrases that can include predicates: Verb Phrases (VP), Noun Phrases (NP), Preposition Phrases (PP), Adverb Phrases (ADVP), Adjective Phrases (ADJP), and others such as INTJ. Empirically, this research has found the boundary area for a predicate in NP to be the NP itself. The boundary area for a predicate in PP can also be itself like a predicate in NP. But for the predicate, "include", there is an additional NP argument from its ancestors as the closest NP. The boundary areas for the rest of the phrases, ADJP, ADVP, and others, can be categorized into VP, NP or PP by tracing upward to their first ancestor labeled VP, NP or PP. Therefore, these six phrase types can be further classified empirically into just three different categories, VP, NP, and PP. The boundary area for a predicate in the VP category is the nearest level of clause, which is the closest S clause from ancestors for predicates in VP. The boundary area for NP is NP itself and the boundary area for a predicate in PP and others is the phrase itself.

These boundary areas cover most of the valid arguments of predicates. According to the pruning strategy, valid arguments can be identified by the upper-most arguments without including the predicate. However, there are valid arguments outside these boundary areas and these are described in Section 4.3.2, and 4.3.3.

4.3.2 Predicates in VP

Argument identification for a predicate under a VP can be implemented by searching children of the closest S clause, children of the closest VP, and constituents within the uppermost VP containing the predicate. Figure 4.7 illustrates a simple case for a predicate in the VP category. The valid arguments in this tree for the predicate "play" are the nodes labeled NP and PP and the modal “would”. Note that *valid arguments* do not include any argument with just punctuation marks, and not any arguments that overlap each other. In Figure 4.7, a valid argument of type NP before the predicate is a sibling of the “uppermost” VP ancestor (denoted as “*V1*”) or one of the children of S. Another valid argument of type PP after the predicate is one of the children of the closest VP ancestor (denoted as “*V2*”). This PP does not include the predicate. Under the top level S, there is another valid one-word argument (the modal “would”).

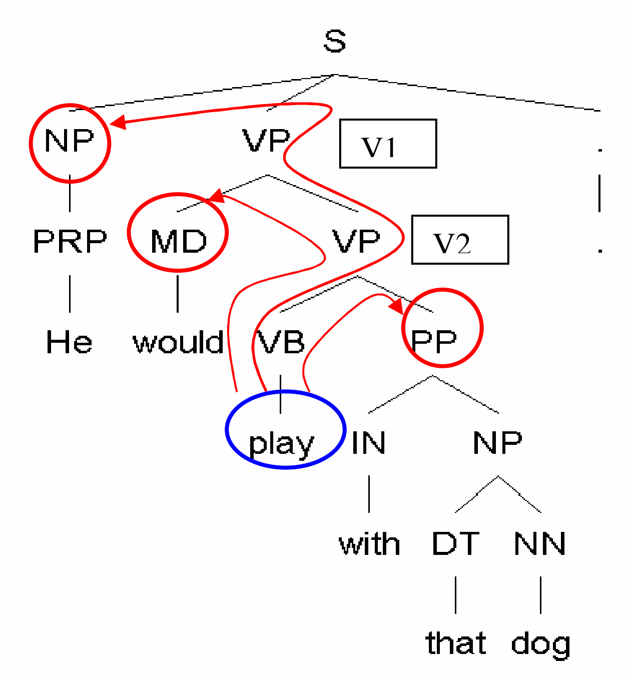


Figure 4.7. Illustration of a fully parsed tree for the VP category.

(Ex. 4.2) from Section 4.2.3 shows another more complicated example for predicates under the VP category. In Ex 4.2, there are two predicates “kick” and “scold”. The arguments associated with the predicate “kick” are “John”, “who”, “Mary” and “this morning”. These arguments are the subject, reference of subject, object, and time modifier (Gildea and Jurafsky, 2002) respectively. Their semantic roles by the definition of PropBank (Palmer et al., 2005) are A0, R-A0, A1, and AM-TMP, respectively. All arguments associated with the predicate “scold” are “John, who kicked

Mary this morning,", "by his mother", and "at home", which are passive subject, passive object, and location modifier respectively. The semantic roles by the definition of PropBank are A1, A0, and AM-LOC respectively. The fully parsed tree for this sentence is shown in Figure 4.8.

Figure 4.8 shows all valid semantic arguments in circles. Traces for the search for valid arguments and the predicates' boundary areas are shown in rectangles. The boundary area for the predicate "kick" is the node "*ID 6*". There is a node "SBAR" "*ID 3*" as its parent. Under "*ID 3"*, there is one "WHNP" child ("*ID 4*"), which is the word "who". The algorithm then marks the "*ID 4*" constituent as a reference to the "NP" labeled "*ID 2*". This NP with "*ID 2*" is the closed NP that does not include the predicate. This NP with "*ID 2*" is a subject of the predicate "kick" because there are no other NP arguments prior to the predicate. The "WHNP" phrase is the reference of that subject. Once the argument search prior to the predicate is done, there are two NPs with "*ID 8*" and "*ID 9*" as arguments under the "VP" with "*ID 7*". These two NP arguments are the children of the closest VP ancestor under the boundary area. In this case, the uppermost VP ancestor is the same as the closest one, and no other valid argument is found between the uppermost VP and the predicate. Thus four semantic arguments apart from the predicate have been found in this argument search under the boundary area of S with "*ID 0*".

The same approach is applied to the predicate "scold". The boundary area for the predicate "scold" is the "S" node with "*ID 0*". There is one "NP" child with "*ID 1*" for the whole constituent, "John, who kicked Mary this morning," as an argument for the predicate "scold". There are two PPs, whose *ID*s are 12 and 13, under the closest VP ancestor containing the predicate "scold". There are two arguments. The uppermost VP ancestor is denoted as "*ID 10*". There is one auxiliary word, "was", between the uppermost VP ancestor and the predicate. This auxiliary word is excluded from semantic arguments in the CoNLL 2005 shared task[1]. Consequently, there are three arguments exclusive to the predicate that have been found in this argument search under

[1] The CoNLL 2005 shared task does not offer any explanations.

the boundary area of "S" with "*ID 0*".

Besides finding semantic arguments related to predicates, the algorithm can also identify the real entity or actor, who performs the action, in order to increase the precision for role classification. This is done according to the Actor heuristic described in Section 3.4. Formally, a grammatical subject is an actor if the action is performed in an active voice. But it would be different if a predicate is in a passive voice. Therefore, in Figure 4.6 the first NP prior to the passive predicate, "scolded" can be considered as a subject but not an actor. And the PP with "by" after the predicate can be identified as an actor according to the Actor heuristic.

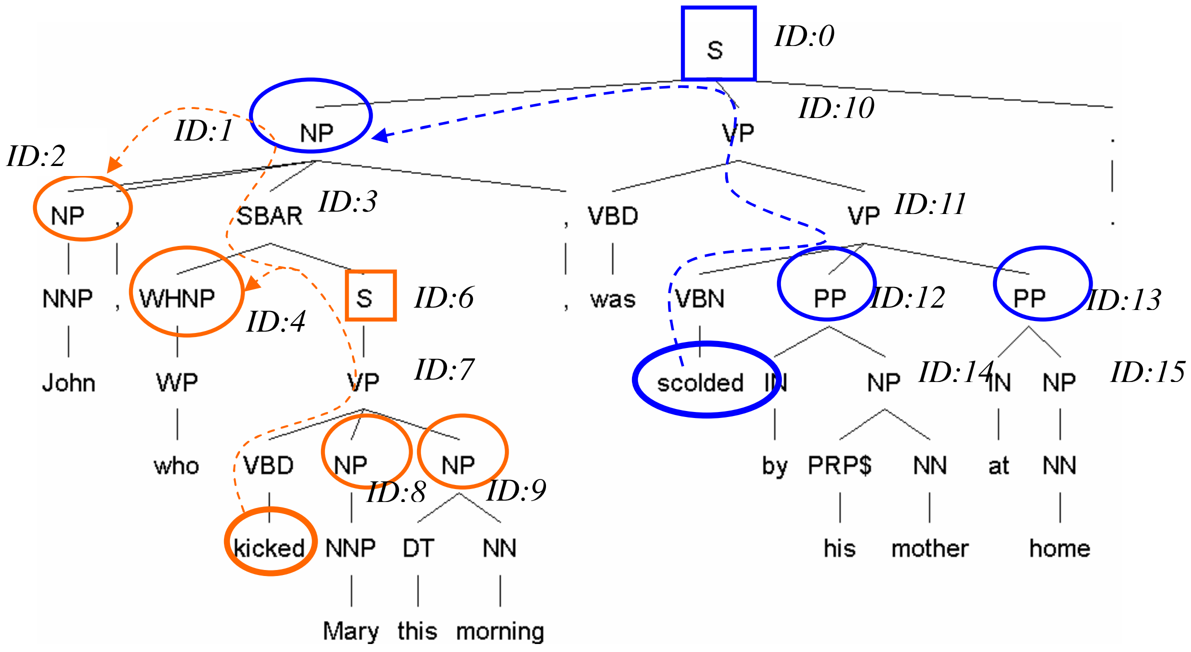


Figure 4.8. Illustration of steps for PARA with 15 sub phrases and 16 words.

4.3.3 Predicates in NP and PP

Different to the VP category, the boundary area for a predicate in the NP and PP categories is the phrase itself. There are some cases where valid arguments are outside the boundary areas such as in the case of the predicate "include" in the PP category. (Ex. 4.1) illustrates both categories. Figure 4.9 shows how PARA works for the two predicates in the Noun Phrase and Prepositional Phrase categories respectively from (Ex. 4.1). There are totally 9 sub phrases and 11 words including the punctuation. The boundary areas for the predicates "increase" and "include" are under the NP with "*ID 6*", and the PP with "*ID 7*" respectively. Only the POS, which is RB, is valid for the argument search before the predicate in a NP. The rest of the words after the predicate are considered as one argument, which consists of the words "car" and "sales". There is only one argument for "increase" excluding the predicate itself, which is "car sales".

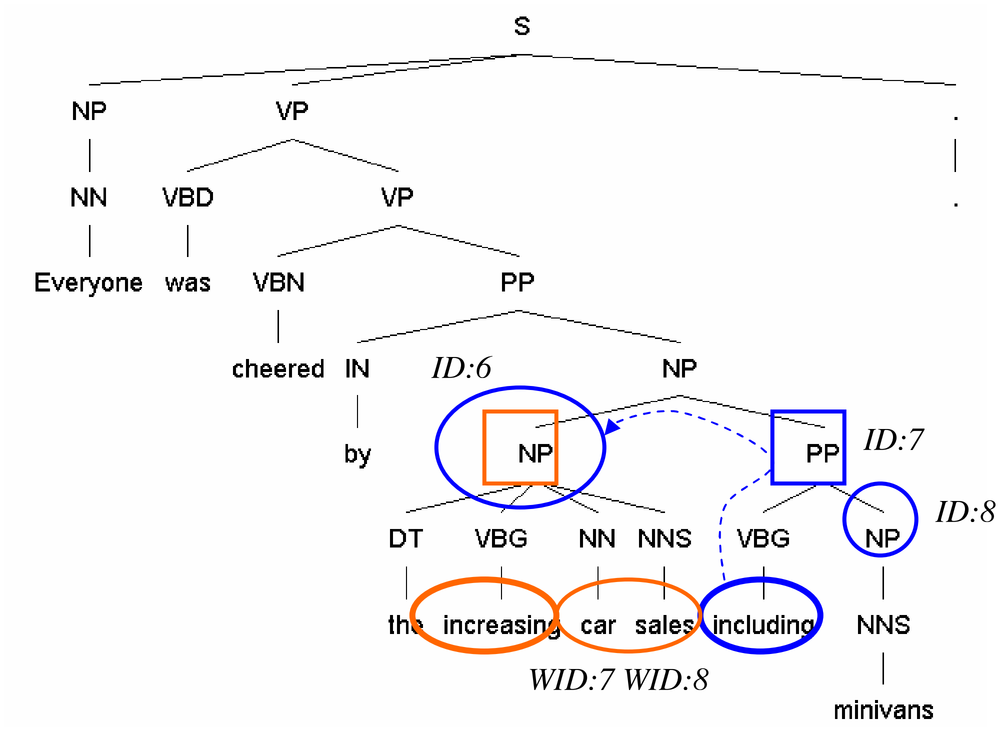


Figure 4.9. Illustration of diagrams for PARA on Example 1 with 9 sub phrases and 11 words.

4.3.4 Post-processing

PARA can identify all semantic arguments of predicates based on parses, but can not detect and correct any errors made by parsers. In order to increase performance, one can perform the following post-processing step for rectifying parse errors:

- skip one word arguments containing only punctuation, and
- skip quotation marks at both ends of an argument.

The first rule reduces errors of an argument recognizer to include arguments containing only punctuation. The second is used to correct errors by parsers that include quotation marks for constituents in different ways.

4.3.5 Output format of PARA

The main task for the direct mapping algorithm is to find the structure of predicates and semantic arguments, and leave role classification to other role classifiers. Therefore, the output format of PARA is defined as the following format, called [GPDV] format, for the procedure in Figure 4.5.

G: Grammatical function with Actor label–5 denotes a subject or an actor, 3 an object and 2, others;

P: Phrase type of the argument – 00 denotes ADJP, 01 ADVP, 02 NP, 03 PP, 04 S, 05 SBAR, 06 SBARQ, 07 SINV, 08 SQ, 09 VP, 10 WHADVP, 11 WHNP, 12 WHPP, and 13 Others

D: Distance of the argument from the predicate – a relative displacement from the predicate with either a "+" note if the argument is before the predicate or a "-" note if it is after the predicate.

V: Voice of the predicate, 0: active 1: passive

The algorithm can also offer notes of references as described in Section 2.3.2 for arguments with “WH” phrase type denoted as “RX”, in which “X” denotes the information on the closest NP prior to this “WH” phrase.

Figures 4.10, and 4.11 show the output for (Ex. 4.1) and (Ex. 4.2). In both figures, the first two columns contain the words and predicates in a sentence. The other columns are for the arguments of the predicates in column 2. The rest of columns specify the predicates’ arguments and their roles respectively in the [GPDV] format. For example, (302-11*) in the first line of Figure 4.10 represents the start and end of the first argument for the predicate “cheer” with [3 02 -1 1] label. It means this argument is an object (3) and a NP (02), and has a displacement prior to the predicate (-1) in the passive (1) voice of the predicate.

```
Everyone     -          (302-11 * )            *              *
was          -                 *               *              *
cheered      cheer          (V * )             *              *
by           -          (503+11 *              *              *
the          -                 *               *       (502-10 *
increasing   increase          *            (V *              *
car          -                 *       (302+10 *              *
sales        -                 *               *              * )
including    include           *               *          (V * )
minivans     -                 * )             * )   (302+10 * )
.            -                 *               *              *
```

Figure 4.10. Output of PARA for (Ex. 4.1).

```
John       -         (502-20 * )        (302-11 *
  ,        -                   *                *
Who        -        (R502-10 * )                *
kicked     kick            (V * )               *
Mary       -         (302+10 * )                *
this       -         (302+20 *                  *
morning    -                  * )               * )
was        -                   *                *
scolded    scold               *           (V * )
by         -                   *        (503+11 *
his        -                   *                *
mother     -                   *                * )
at         -                   *        (203+21 *
home       -                   *                *
.          -                   *                * )
```

Figure 4.11. Output of PARA for (Ex. 4.2).

Summarily, the same point of this algorithm and the pruning strategy is to find arguments based on the upper-most phrases; and the main differences are as follows:

1) Different definition of boundary areas for different predicate categories

   PARA categorizes three types, which are VP, NP and PP & others;

2) Arguments outside boundary areas

   In order to add arguments outside boundary areas, PARA adds a simple checking algorithm, which is to browse upwards for more arguments if a predicate under the VP category does not include a NP before the predicate or this boundary area of the predicate has a SBAR parent.

3) Post-processing

   PARA includes heuristics of post-processing to enhance system performance by skipping punctuation or quotation marks.

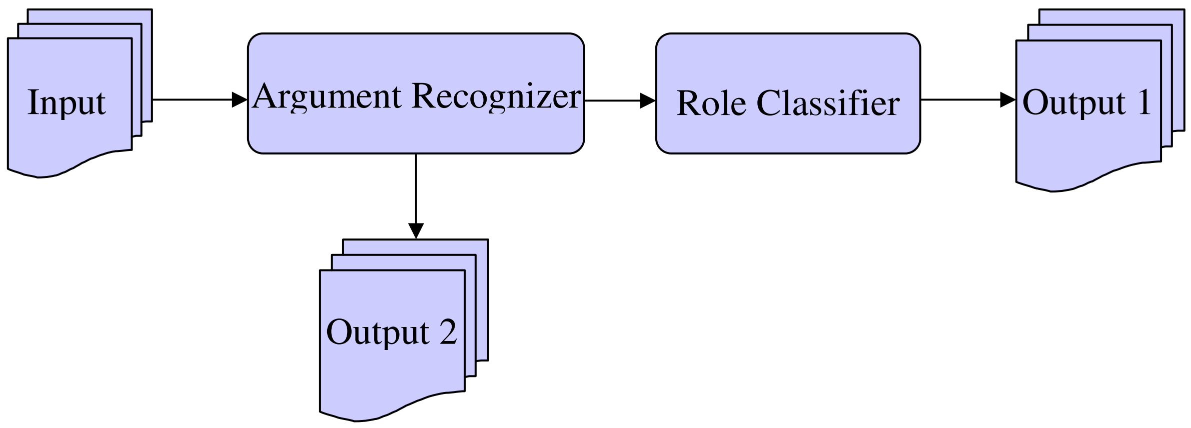


Figure 4.12. Illustration of system architecture.

4.3.6 System Architecture

The system presented in this section is designed to test consistency of performance in SRL when semantic arguments recognized by PARA are forwarded to a role classifier. Figure 4.12 illustrates the system architecture including Input, PARA, Role Classifier, and Output.

Input

The input format is the same as the one used for the CoNLL 2005 shared task, with all syntactic information as shown in Figure 4.13. The first column contains words or tokens for a sentence. The second column is the Name Entity information, which indicates whether the corresponding phrase is a person (PER), organization (ORG), and so on, as defined by the shared task. The third column has POS tags for each word, and partial syntactic annotations are listed in the fourth and fifth columns, composed of phrase chunks and clause brackets. A full syntactic parse, given by a Charniak or Collins parser, is in the sixth column. In the shared task, targets or predicates are also given as input, which are shown in the seventh column. For the testing stage, these seven columns offer sufficient information for an argument recognizer and role classifier to assign roles. The remaining columns, provided as input, vary according to the number of predicates. In this example, Columns 8 and 9 contain the arguments and roles for the predicates “face” and “explore” respectively. Each argument is annotated with the Start-End format described in Section 2.5.

Argument recognizer

PARA is used as the argument recognizer to identify predicates and their arguments, and forwards all of those to role classifiers. Details of PARA have been explained in Section 4.3.

WORDS	NE	POS	PARTIAL_SYNT		FULL_SYNT	TARGETS	PROPS	
The	*	DT	(NP*	(S*	(S(NP*	-	(A0*	(A0*
$	*	$	*	*	(ADJP(QP*	-	*	*
1.4	*	CD	*	*	*	-	*	*
billion	*	CD	*	*	*))	-	*	*
robot	*	NN	*	*	*	-	*	*
spacecraft	*	NN	*)	*	*)	-	*)	*)
faces	*	VBZ	(VP*)	*	(VP*	face	(V*)	*
a	*	DT	(NP*	*	(NP*	-	(A1*	*
six-year	*	JJ	*	*	*	-	*	*
journey	*	NN	*)	*	*	-	*	*
to	*	TO	(VP*	(S*	(S(VP*	-	*	*
explore	*	VB	*)	*	(VP*	explore	*	(V*)
Jupiter	(ORG*)	NNP	(NP*)	*	(NP(NP*)	-	*	(A1*
and	*	CC	*	*	*	-	*	*
its	*	PRP$	(NP*	*	(NP*	-	*	*
16	*	CD	*	*	*	-	*	*
known	*	JJ	*	*	*	-	*	*
moons	*	NNS	*)	*)	*)))))))	-	*)	*)
.	*	.	*	*)	*)	-	*	*

Figure 4.13. Input in flat format for the system.

Role classifier

In order to test the direct mapping algorithm, a modified version of the system described by Palmer et al. (2005) is implemented, which uses the basic features predicate, head word, phrase type, voice, and distance; plus an additional preposition feature and the actor heuristic introduced in Chapter 3. Pre-assignment and post-processing for duplicate roles as described in Section 3.4 are also included. The probability estimation (F 3.3) described in Section 3.3.1 is formulated as follows:

$$\begin{aligned} P(\,r \mid hw, pt, path, dis, voice, predicate) = {} & \\ & \lambda_1 {*} P(r \mid predicate) + \lambda_2 {*} P(r \mid pt, predicate) + \\ & \lambda_3 {*} P(r \mid pt, path, predicate) + \lambda_4 {*} P(r \mid pt, dis, voice) + \\ & \lambda_5 {*} P(r \mid pt, dis, voice, predicate) + \lambda_6 {*} P(r \mid hw) + \\ & \lambda_7 {*} P(r \mid hw, predicate) + \lambda_8 {*} P(r \mid hw, pt, predicate) \end{aligned}$$

where $\sum_i \lambda_i = 1$ and $\lambda_i = \lambda_{i+1}$, and "*r*" denotes semantic roles (such A0, A1, AM-TMP, and so on).

This ML classification approach is based on the "backoff" lattice. The lattice employed here is shown in Figure 4.14 according to (F 3.3). Less-specific distributions, going down the lattice, are used only when no data is found for any more-specific distribution (i.e. when the probability is zero at higher levels in the lattice). For example, P(*r* | *pt, pp, pr*) will be used only when any top-level distribution yields probabilities of zero, where the top-level estimates include P(*r* | *hw, pt, pp, pr*), P(*r* | *pt, pa, pp, pr*) and P(*r* | *pt, di, vo, pp, pr*). The unused probability is set to be zero so that the distribution sums to one.

More detail can be found in Chapter 3.

$$\begin{tikzcd}
P(r \mid hw, pt, pp, pr) \arrow[d] \arrow[dr] & P(r \mid pt, pa, pp, pr) \arrow[d] & P(r \mid pt, di, vo, pp, pr) \arrow[dl] \arrow[dd] \\
P(r \mid hw, pp, pr) \arrow[d] \arrow[dr] & P(r \mid pt, pp, pr) \arrow[d] & \\
P(r \mid hw, pp) & P(r \mid pp, pr) & P(r \mid pt, di, vo, pp)
\end{tikzcd}$$

Figure 4.14. Illustration of the modified "Backoff" lattice.

```
-          (A0*      (A0*
-             *         *
-             *         *
-             *         *
-             *         *
-             *)        *)
face        (V*)        *
-          (A1*         *
-             *         *
-             *         *
-             *         *
explore       *       (V*)
-             *       (A1*
-             *         *
-             *         *
-             *         *
-             *         *
-             *)        *)
-             *         *
```

Figure 4.15. An output example for the evaluation of role classification in the CoNLL 2005 shared task.

Outputs

There are two kinds of outputs: one (Output1) for role classification and the other (Output2) for argument identification. In order to have a fair comparison with existing ML approaches, this research uses the same output format as the CoNLL 2005 shared task for Output1. Output1 for evaluation only contains a column with all the predicates, and one column for each predicate. Each tag after the predicate column is in the *STARTS**ENDS*, format, which represents phrases that start and end at the corresponding words. When systems are evaluated for argument identification and role classification, the output format is the same as the standard format used in the shared task. For example, Figure 4.15 is the output for evaluation of role classification for the example in Figure 4.13.

Since the evaluation script offered in the shared tasks does not directly allow the calculation of performance for argument identification, Output2 for argument identification is different from Output1 for argument classification. All labels of arguments offered by the shared task are replaced with the A0 labels excluding the predicates, and the output data of systems is modified to replace all arguments with the A0 label, excluding the predicate arguments. Figure 4.16 is an output example of argument identification for the example in Figure 4.13. This prevents incorrect evaluation by the conll script due to the continual labels. For example, the number of correct arguments in the semantic list [A1 V A2] to the proposition [A1 V C-A1] is zero since the conll script counts word spans of A1 and C-A1 as a whole argument instead of two arguments.

```
-           (A0*     (A0*
-              *        *
-              *        *
-              *        *
-              *        *
-              *)       *)
face         (V*)       *
-           (A0*        *
-              *        *
-              *        *
-              *        *
explore        *      (V*)
-              *      (A0*
-              *        *
-              *        *
-              *        *
-              *        *
-              *)       *)
-              *        *
```

Figure 4.16. An output example for the evaluation of argument identification in the CoNLL 2005 shared task.

4.4 Data, Evaluation, and Parsers

The results in this thesis are based on the CoNLL 2005 dataset released in March 2005[2], which includes Wall Street Journal sections annotated with Charniak's (Charniak, 2000) and Collins' (Collins, 1999) parse trees. Wall Street Journal sections are part of the PropBank corpus[3]. The whole WSJ 02-21 is used as training data for role classification. This training data contains annotations for 90,750 predicate-argument structures or propositions with 239,858 individual arguments, and has 39,832 sentences with 950,028 words or tokens, and 3,101 lexical predicates (types). The WSJ23 test data contains 982 different predicates with 14,077 arguments and the Brown data has 351 predicates with 2,177 arguments. Table 4.2 summarizes these datasets, including training sets WSJ 02 to 21, the development set WSJ 24, the main test set WSJ 23, and the Brown corpus test set as another English domain.

[2] http://www.lsi.upc.edu/~srlconll/soft.html
[3] http://www.cis.upenn.edu/~treebank

	Train.	Development	Test WSJ	Test Brown
Sentences	39,832	1,346	2,416	426
Tokens	950,028	32,853	56,684	7,159
Propositions	90,750	3,248	5,267	804
Verbs	3,101	860	982	351
Arguments	239,858	8,346	14,077	2,177

Table 4.2. Summarization of data sets used in this research.

Evaluation is performed on a collection of test sentences with all input information like POS, parses described in Section 3.5 and target verbs or predicates given as input annotation. An official script, *srl-eval.pl* available from the CoNLL 2005 Shared Task website[4], evaluates a system with respect to precision, recall and $F_{\beta=1}$ (i.e. the F_1 measure) of the predicated arguments. This evaluation measurement has been described in Section 3.5.2. It is based on the formula: $F_1 = 2*p*r / (p+r)$, and is widely used in Natural Language Processing.

4.5 Experiments and Results

This section shows the results for argument identification. Section 4.5.1 shows the performance of PARA with and without given predicates. The differences between gold-standard or hand-corrected parses and auto parses are discussed in Section 4.5.2. The analysis of PARA in Section 4.5.3 shows the performance in three predicate categories and inside and outside boundary areas. Standard figures for comparison on WSJ 23 are in Section 4.5.4. Comparisons with conventional argument recognizers are in Section 4.5.5. The contribution of improvements for different recognizers is shown in Section 4.5.6. Results for the Brown corpus are compared in Section 4.5.7. The effect of punctuation is shown in Section 4.5.8. Section 4.5.9 discusses the runtime of PARA when compared to conventional argument recognizers. The joint model of

[4] http://www.lsi.upc.edu/~srlconll/soft.html

PARA with the ML approach increases improvements shown in Section 4.5.10, and the results of argument identification and role classification are tabled in Section 4.5.11.

4.5.1 Given vs. Identified

In the CoNLL 2005 shared task, all predicates are given to systems. Tables 4.3 and 4.4 show the effect on PARA if predicates must be identified before any other tasks are performed. In Table 4.3, based on hand-corrected parse trees, results are the same with given predicates (PARA-G) and without predicates (PARA-N) on WSJ 24. It is not surprising for this result because all predicates in the hand-corrected parses are correctly labeled with verbal POS tags such as VBN, VBZ, and so on.

There is an approximately 2-point drop (80.89 vs. 78.77) of overall performance in F_1 caused by errors in Charniak's parses, which means some predicates are not tagged as a verbal POS tag.

Note that the post-processing discussed in Section 4.3.4 is only used when the input parse trees are not hand-corrected.

	Precision	**Recall**	**F_1**
PARA-N	91.54	92.42	91.98
PARA-G	91.54	92.42	91.98

Table 4.3. Results for PARA with hand-corrected parses on WSJ Section 24.

	Precision	**Recall**	**F_1**
PARA-N	81.14	76.53	78.77
PARA-G	80.76	81.01	80.89

Table 4.4. Results for PARA with auto parses on WSJ Section 24.

4.5.2 Gold parses vs. Auto parses

Degradation of performance appears not only in Table 4.4, from given to identified, but also in the comparison of hand-corrected parses to auto parses. There is a gap of more than 11 points (in F_1) between results obtained with hand-corrected parses and auto parses with given predicates on WSJ 21 and 24 as shown in Table 4.5. The overall performance of hand-corrected parses on WSJ 21 is about 4.5 (93.44 vs. 89.88) over the Charniak's parses. But the overall performance of hand-corrected parses on WSJ 24 improves by about 11 points (91.98 vs. 80.89) on the Charniak's parses. Meanwhile, the big drop (89.58 to 80.89) on WSJ 24 shows the Charniak's parser performs variously on different data sets.

Parses	WSJ	Precision	Recall	F1
Hand-corrected	*21*	93.07	93.80	93.44
	24	91.54	92.42	91.98
Charniak	*21*	89.02	90.14	89.58
	24	80.76	81.01	80.89

Table 4.5. Results for PARA on WSJ Section 21, 24 with hand-corrected parses and Charniak's parses.

4.5.3 Analysis of Results

As mentioned in Section 4.3, there are three main categories of phrases, which are VP, NP, and Others. Tables 4.6 and 4.7 show the frequency of the three types of categories and their performance for hand-corrected parses and Charniak's parses respectively when excluding the other phrases. The majority of the predicates, the category with the biggest recall, in both Tables 4.6 and 4.7 are in the VP category, and dominate the overall performance. Precision in the VP category drops by about 11 points for Charniak's parses compared to the hand-corrected ones. It is mainly due to errors in assigning prepositional phrases after predicates, as shown in Figure 4.17.

The difference between hand-corrected parses and Charniak's parses in Figure 4.17 is whether the prepositional phrase "with us" is under the ADVP "here" or not. The hand-corrected parse includes it but Charniak's parse does not. This difference or error causes wrong assignments in the argument identification. There are 5 semantic arguments in the hand-corrected parse excluding the predicate, but 6 in Charniak's parse. This kind of error is beyond the scope of PARA, and might be solved by ML based recognizer in the future.

WSJ 24	**%**	**P**	**R**	**F1**
Overall		91.54	92.42	91.98
VP	93.72	91.56	89.81	90.68
NP	5.51	90.43	2.24	4.36
Others	0.77	93.94	0.37	0.73

Table 4.6. Results for different phrase types on WSJ 24 – hand-corrected parses by PARA.

WSJ 24	**%**	**P**	**R**	**F1**
Overall		80.76	81.01	80.89
VP	92.49	80.62	78.28	79.43
NP	6.37	86.40	2.33	4.54
PP & others	1.14	79.07	0.40	0.80

Table 4.7. Results for different phrase types on WSJ 24 – Charniak parses by PARA.

we	PRP	(S1(S(NP*)	-	(A0*)
may	MD	(VP*	-	(AM-MOD*)
not	RB	*	-	(AM-NEG*)
see	VB	(VP*	see	(V*)
them	PRP	(S(NP*)	-	(A1*)
here	RB	(ADVP*	-	**(AM-LOC***
with	IN	(PP*	-	*****
us	PRP	(NP*))))))))	-	***)**

we	PRP	(S1(S(NP*)	-	(502-30*)
may	MD	(VP*	-	(209-20*)
not	RB	*	-	(209-10*)
see	VB	(VP*	see	(V*)
them	PRP	(S(NP*)	-	(302+10*)
here	RB	(ADVP*)	-	**(200+20*)**
with	IN	(PP*	-	**(203+30***
us	PRP	(NP*)))))))	-	***)**

Figure 4.17. An example hand-corrected parse and Charniak's parse.

Tables 4.8 and 4.9 show the results on hand-corrected and auto parses respectively for inside and outside boundary areas, i.e. their performance inside (if arguments are located in the boundary area) and outside (if not in the boundary area). Most arguments appear inside the boundary area. In total, about 90 percent of the arguments are inside, and 10 percent outside. Precision and Recall inside the boundary area are better than outside the boundary area for both parses. This suggests that it is more difficult to trace arguments outside the boundary area, such as WH-phrases and their related constituents. For example, in Figure 4.8, two arguments outside the S clause (denoted as "ID: 2" for the predicate, "kick") are the NP with "John", and the WHNP with "who".

Boundary Area	*%*	**P**	**R**	**F1**
Overall		91.54	92.42	91.98
Inside	90.06	94.20	81.72	87.48
Outside	9.94	75.78	10.70	18.74

Table 4.8. Results inside and outside boundary areas on WSJ 24 – hand-corrected parses by PARA.

Boundary Area	%	**P**	**R**	**F1**
Overall		80.76	81.01	80.89
Inside	90.11	82.52	71.13	76.40
Outside	9.89	70.05	9.88	17.31

Table 4.9. Results inside and outside boundary areas on WSJ 24 – Charniak's parses by PARA.

WSJ 23	%	**P**	**R**	**F1**
Overall		82.90	82.30	82.60
VP	96.47	82.92	81.10	82.01
NP	2.33	83.97	0.77	1.52
PP &Others	1.20	76.54	0.43	0.86
Inside	90.08	84.39	72.53	78.01
Outside	9.92	73.26	9.77	17.24

Table 4.10. Allocation of predicates in PARA based on Charniak's parses on WSJ 23.

4.5.4 Results for WSJ 23

Table 4.10 shows the percentage of different boundary categories and performance based on Charniak' parses on WSJ 23. Compared to the results in Table 4.7, allocation of the three phrase categories, VP, NP, and Others are different. The precision of the main phrase category, VP, increases about 2%, but that of NP decreases about 4%. This reveals that the results by PARA based on Charniak's parses vary because of parser's different performance in different datasets. Generally speaking, the performance of argument identification on WSJ 23 is about 2 points (in F_1) better than on WSJ 24.

System	Corr.	Excess	Missed	Precision	Recall	F1	Synt
PARA	**11794**	2433	**2537**	82.90	**82.30**	**82.60**	cha
Moschitti	11652	2322	2679	83.38	81.31	82.33	cha
Che	11138	1710	3193	**86.69**	77.72	81.96	cha
Tjongkimsang	11009	1943	3322	85.00	76.82	80.70	cha
Yi	11196	2231	3135	83.38	78.12	80.67	cha
Surdeanu	10931	1943	3400	84.91	76.28	80.36	cha

Table 4.11. Comparison with other existing systems using only Charniak's parses.

4.5.5 Comparison with Other Systems

Table 4.11 shows the performance of PARA and some existing systems using the same syntactic information (Charniak's parses) from the CoNLL 2005 shared task as input. The best ML performance for argument identification, achieved by Moschitti et al., (2005) was 82.33 in F_1 measurement. PARA (82.60) outperforms this by 0.27 point. Although this is not much, it shows that even with a state-of-the-art Machine Learning approach, the result for argument identification is inferior to the one obtained by the direct mapping algorithm, PARA.

PARA not only exhibits the top performance in F_1, but also identifies the most correct (Corr.) and the fewest missed (Missed) arguments, as shown in the second and fourth column in Table 4.11. PARA also achieves the best performance in Recall (in column 6). Although PARA finds more correct arguments, it also gets more excessive (Excess) or wrong arguments, which results in poor Precision for PARA compared to other systems. It is because PARA uses general rules to recognizer arguments without any ML validation. Such ML validation can help to increase precision but reduce recall as discussed in Section 4.5.10.

Besides the Charniak's parses, other syntactic information used in the CoNLL 2005 shared task include Collin's parses (col), partial parses (upc) with chunks and clauses, and n-best parsings generated by re-trained parsers such as "6-cha" denoting the six top parses generated by Charniak's parser after re-training. Table 4.12 lists the top systems with multi-syntactic information in the shared task. By comparison, the best system, Punyakanok et al. (2005), using rich syntactic information outperforms PARA using singular Charniak's parse by 1.68 points. It will be the future work to include different parses in PARA.

System	Corr.	Excess	Missed	Precision	Recall	F1	Synt
Punyakanok	11668	1612	2663	87.86	81.42	**84.52**	6-cha,col
Marquez	**11752**	1869	**2579**	86.28	**82.00**	84.09	cha, upc
Haghighi	11886	2092	2445	85.03	82.94	83.97	5-cha
Pradhan2	11358	1458	2973	**88.62**	79.25	83.68	cha, col, chunk

Table 4.12. Comparison with other existing systems using rich syntactic information.

4.5.6 Contribution of Post-processing

The post-processing method discussed in Section 4.3.5 increases system performance. Table 4.13 shows improvements in some systems with Charniak's parses after applying the post-processing rules of skipping one-word punctuation arguments, and quotations at the end of arguments. The system by Moschitti et al. (2005) gains another 0.27 points in F_1 measurement, but that by Surdeanu et al. (2005) does not exhibit any change in performance. This means some systems in the CoNLL 2005 shared task have already taken the punctuation issue into consideration.

System	Precision	Recall	F_1	Improved
Moschitti	83.63	81.54	82.57	0.24
Che	86.76	77.78	82.03	0.07
Tjongkimsang	85.15	76.95	80.84	0.14
Yi	83.39	78.13	80.68	0.01
Surdeanu	84.91	76.28	80.36	0.00

Table 4.13. Comparison with other existing systems using only Charniak's parses.

4.5.7 Other Domain – the Brown Corpus

The CoNLL2005 shared task is not limited to the WSJ data, but also uses the Brown corpus for testing in another domain (i.e. other than WSJ). The results shown in Table 4.14 indicate a drop in F_1 of about 3 points for argument identification in the Brown Corpus domain in comparison to performance on WSJ 23. As is the case for WSJ 23, most predicates in the Brown Corpus are in the VP category, which influences the overall performance. The precision on predicates within PP increases significantly to 100 when PARA only includes PP in the Brown corpus. Meanwhile, we test Moschitti's performance on the Brown Corpus and find it drops 3.75 points compared to the one on WSJ 23 (from 82.33 to 78.58); thus PARA appears to perform comparably to Moschitti's ML method.

Brown 32	**%**	**P**	**R**	**F1**
Overall		80.60	78.51	79.54
VP	94.90	80.57	77.06	78.77
NP	3.98	80.56	1.32	2.59
PP only	1.12	100.00	0.13	0.27
Inside	91.85	80.89	70.01	75.06
Outside	8.15	78.24	8.50	15.33

Table 4.14. Results by PARA on some data in Brown Corpus with Charniak's parses.

4.5.8 Punctuation Effect

There are some incorrect arguments caused by punctuation, such as quotations (`) in an argument span. The evaluation script used in the shared task considers an argument valid if it exhibits the correct word span including punctuation. In order to check how much impact the punctuation has, an variant of the evaluation script, *my_eval()*, was developed for measuring performance without punctuation. This evaluation program evaluates a system by ignoring whether or not labels are correct because there is no need to verify labels at the stage of argument identification. Table 4.15 shows the results obtained by *my_eval()* on the same data with Charniak's parse trees. The results obtained by *srl-eval.pl* are almost the same as the results for "with-punctuation" (W/T Punc) datasets in this table. By skipping the punctuation, the performance in F_1 (W/O Punc) increases from about 0.44 (on Sec 23) to 0.90 (on Brown) compared to those with punctuation. It shows how punctuation influences Charniak's parses within 1 %.

		P	**R**	**F_1**
W/T Punc.	Sec 24	80.76	81.01	80.88
	Sec 23	82.89	82.29	82.59
	Brown	80.57	78.48	79.51
W/O Punc.	Sec 24	81.33	81.61	81.47
	Sec 23	83.29	82.76	83.03
	Brown	81.50	79.34	80.41

Table 4.15. Results with and without punctuation in PARA evaluated by *my_eval()* for WSJ Section 23, and 24, and Brown corpus with Charniak's parser.

4.5.9 Execution time in different sizes of Dataset

Table 4.16 includes test data sets (column 1), numbers of sentences for each data set (column 2), F_1 performance (column 5), total execution time in minutes for each data set (column 6) and average execution time in seconds for each sentence (column 7). This table shows the execution time by PARA on WSJ 20, 21, 23, 15 to 18, and all of 02 to 21 combined with Charniak's parses. The average time to process each sentence is 0.002 second. It is worth noting that the F_1 performance for WSJ 02-21 is 88.40, but only 80.70 for WSJ23. Performance drops severely from the first 20 WSJ sections to WSJ 23, which means the performance of Charniak's parser varies in different WSJ sections.

Test	No of sen	P	R	F_1	T1(min)	T2(sec/sen)
Sec 20	2012	89.32	90.69	90.00	0.080	0.002
Sec 21	1671	89.02	90.14	89.58	0.063	0.002
Sec 23	2416	82.90	82.30	82.60	0.087	0.002
Sec 15-18	8936	89.75	90.40	90.07	0.349	0.002
Sec 02-21	39832	89.35	90.17	89.76	1.485	0.002

Table 4.16. Execution time on different datasets for PARA.

Test	Palmer	Palmer-Pruning	PARA
Sec 23	4.913	0.987	0.002
Brown	5.561	1.157	0.002

Table 4.17. Execution time (sec / sen) using different approaches on WSJ 23 and Brown Corpus.

PARA not only achieves the best performance as shown in Table 4.11, but also runs much faster in execution time compared to the statistical argument recognizer in Table 4.17. To take the statistical approach based on Palmer et al., (2005) as an example, the execution time for argument identification is about five seconds per sentence without the pruning strategy (Palmer), and about one second with the pruning strategy (Palmer-Pruning). PARA needs only 0.002 seconds per sentence, which is much faster than traditional argument recognizers. Performance in terms of execution time will be critical for real-time NLP applications in the future as data sizes increase. From the results obtained in this research, the training-free algorithm PARA can serve as a good solution for the Predicate-Argument Recognition problem. All the above figures were obtained using computers with a Linux operating system, a P4 3.0GHz CPU, and 1G RAM.

4.5.10 Joint model of PARA+ML

An interesting question is whether performance can be improved by joining PARA with a ML approach. In Tables 4.8 and 4.9, the precision outside the boundary area is much less than that inside. In order to increase the precision for arguments outside boundary areas, this research combines PARA with an argument recognizer based on the one presented by Palmer et al. (2005) to validate each argument from PARA that is outside boundary areas. This is done by checking whether or not the outside argument is found by the statistical argument recognizer. Tables 4.18, 4.19, and 4.20 illustrate the results obtained with the stand-alone ML recognizer (Palmer-M), PARA, and the joint model (Joint) on WSJ 24, WSJ 23, and the partial Brown corpus respectively for argument identification.

On WSJ 24	**P**	**R**	**F**
Palmer-M	79.38	77.63	78.49
PARA	80.76	**81.01**	**80.89**
Joint	**81.51**	80.15	80.82

Table 4.18. Results for argument identification with the joint model of PARA and ML on WSJ 24.

On WSJ 23	**P**	**R**	**F**
Palmer-M	81.30	80.62	80.96
PARA	82.90	**82.30**	82.60
Joint	**83.60**	81.84	**82.71**

Table 4.19. Results for argument identification with the joint model of PARA and ML on WSJ 23.

On Brown	**P**	**R**	**F**
Palmer-M	79.25	76.19	77.69
PARA	80.60	**78.51**	**79.54**
Joint	**80.88**	77.83	79.32

Table 4.20. Results for argument identification with the joint model of PARA and ML on Brown Corpus.

Precision obtained by the joint model increases on the three test data sets, but recall decreases. The overall performance in F_1 measurement increases slightly on WSJ 23, but not on WSJ 24 and the Brown corpus. The reason is that the joint model may be too simple to gain enough additional Precision on WSJ 24 and the Brown corpus to compensate for the loss of Recall. More sophisticate ML joint model will be the future work.

4.5.11 Performance with Argument Classification

Table 4.21 shows the latest results obtained using the statistical approach to argument identification and role classification proposed by Palmer et al. (2005). The training datasets are WSJ 02 to WSJ 21, and WSJ 23 is the test data. Palmer et al. (2005) used Collins parses, hand-corrected parses, and hand-corrected parses with traces.

Palmer et al. (2005, p.97) states:

> … the gold-standard parses of the Penn Treebank include several types of information not typically produced by statistical parsers or included in their evaluation. One of these annotations is called "trace". Traces are used to indicate cases of wh-extraction, and antecedents of relative clauses. Traces are intended to provide hints as to the semantics of individual clauses.

Table 4.21 shows the traces do contribute significantly to performance, but they are not included as input in the CoNLL shared task.

On WSJ 23	**P**	**R**	**F**
Automatic parses	68.6	57.8	62.7
Gold-standard parses	74.3	66.4	70.1
Gold-standard parses with traces	80.6	71.6	75.8

Table 4.21. Latest results by Palmer et al. (2005) for the performance for unknown arguments with gold-standard parses on WSJ 23.

As was done in the implementation of the joint model consisting of PARA and the statistical argument recognizer in Section 4.5.9, this research also applies the concept of a joint model to the role classifier based on the one by Palmer et al. (2005) to test how much improvement is gained by this joint model. Tables 4.22, 4.23, and 4.24 show the results on WSJ 24, WSJ 23, and the partial Brown corpus. These results are obtained by different argument recognizers on the task of argument identification and role classification. All syntactic information offered by the shared task consists of Charniak's parses. There are results for both the performance for argument identification and role classification. Label accuracy (Lacc), the ratio of the F_1 in the role classification to the F_1 in the argument identification, is also shown. The overall performance on argument identification after role classification differs from results of the one before classification in Tables 4.18, 4.19, and 4.20. It is due to the post-processing for skipping duplicate roles, and errors in continual labels evaluated by the shared task's script. As for the same example in Section 4.3.6, the number of correct arguments in the semantic list [A1 V A2] to the proposition [A1 V C-A1] is zero since the conll script counts word spans of A1 and C-A1 as a whole argument instead of two arguments.

	Argument identification				**Role classification**		
System	**P**	**R**	**F**	**Lacc**	**P**	**R**	**F**
PML+ Palmer-M	78.95	76.99	77.96	88.34	69.75	68.02	68.87
PML + PARA	80.38	**80.69**	**80.53**	**88.70**	71.29	**71.57**	**71.43**
PML + Joint	**81.22**	79.67	80.44	88.55	**71.93**	70.55	71.23

Table 4.22. More performance results obtained by different argument recognizers on WSJ 24.

In Tables 4.22, 4.23, and 4.24, the joint model does not outperform PARA in F_1 measurement for argument identification and role classification. This indicates that PARA can offer suitable semantic arguments for the statistical role classifier to obtain better overall performance on the WSJ and the Brown corpus domain. The robustness of PARA shown in these tables therefore demonstrates the feasibility of a direct mapping from syntactic parses to unlabelled semantic arguments through a heuristic algorithm that fully relies on parse trees.

	Argument identification				**Role classification**		
System	**P**	**R**	**F**	**Lacc**	**P**	**R**	**F**
PML+ Palmer-M	80.17	79.81	79.99	88.87	71.24	70.92	71.08
PML + PARA	81.85	**81.91**	81.88	**89.75**	73.46	**73.51**	**73.49**
PML + Joint	**82.62**	81.27	**81.94**	89.35	**73.83**	72.62	73.22

Table 4.23. More performance results obtained by different argument recognizers on WSJ 23.

	Argument identification				**Role classification**		
System	**P**	**R**	**F**	**Lacc**	**P**	**R**	**F**
PML+ Palmer-M	78.91	75.79	77.32	79.70	62.89	60.40	61.62
PML + PARA	80.17	**78.00**	**79.07**	**80.68**	**64.93**	**62.93**	**63.80**
PML + Joint	**80.58**	77.22	78.86	79.89	64.38	61.69	63.01

Table 4.24. More performance results obtained by different argument recognizers on the Brown Corpus.

4.6 Summary and Remarks

Semantic argument identification, a sub-problem of Semantic Role Labeling (SRL), is the task of finding all arguments or constituents corresponding to predicates in a sentence. Traditional argument recognizers identify all arguments based on features extracted from syntactic information such as head word, path, phrase type and so on. Gildea and Palmer (2002) claim the necessity of parsing for predicate-argument recognition or semantic argument identification. This research further demonstrates the possibility of generating direct mapping from syntactic parses to unlabelled semantic arguments.

The proposed Predicate-Argument Recognition Algorithm (PARA) achieves the goal of a direct mapping from syntactic parses to unlabelled semantic arguments without any need for training by utilizing the output from a state-of-art parser like Charniak (2000). In gold-standard (hand-corrected) parses, the performance of argument identification with PARA without given predicates is the same as that with given predicates; but a 2 point drop occurs when using the auto (Charniak) parses. The overall performance difference between gold-standard parses and auto (Charniak's) parses is between 3 and 11 points in F_1 measurement, shown in Table 4.5. Results for argument identification in Sections 2 to 21 of the WSJ corpus are generally stable, but performance drops significantly on WSJ 23 and WSJ 24.

The major category containing predicates is VP, containing more than 90% of the predicates, and it dominates the overall performance of argument identification. Similarly, the frequency of arguments inside boundary areas influences the overall results as well. Precision inside boundary areas is better than outside boundary areas. This suggests recognition outside boundary areas needs more attention.

There is an approximate 0.5-point increase in F1 measurement when evaluating PARA disregarding punctuation. The execution time for PARA when performing argument identification is constant, at about 0.002 seconds per sentence on a P4 3.0 CPU, 1.0G

RAM, Linux-based machine. This algorithm is thus much faster than traditional argument recognizers, even when they employ a pruning strategy. The algorithm presented here also outperforms existing ML approaches when the same syntactic information is used as was through the CoNLL 2005 shared task. Moreover, the output of the heuristic algorithm also helps to boost the performance of statistical role classifiers when used in a joint model The joint model consisting of PARA and the statistical argument recognizer from Palmer et al. (2005) improves the performance of argument identification, but it does not seem to help the combined task of argument identification and role classification.

PARA is a fast and accurate algorithm, which can be used as a stand-alone pre-processor for the problem of Predicate-Argument Recognition or jointly with other ML recognizers to increase the overall performance.

This chapter has demonstrated that the solution to argument identification by PARA is better than the performance exhibited by existing systems with single syntactic information in the CoNLL 2005 shared task. Based on the performance obtained by PARA, the following chapter will introduce a new technique for multi-argument classification. This technique not only boosts the performance of the existing singular argument classifier (Palmer et al. 2005) described in Chapter 3, but also outperforms existing systems in the shared task with single syntactic information for the problem of Semantic Role Labeling.

Chapter 5

Multi-Argument Classification

This chapter describes a Multi-Argument Classification (MAC) approach to Semantic Role Labeling via a Pattern Base (PB). The goal is to exploit dependencies between semantic roles by simultaneously classifying all arguments as a pattern. Argument identification, as a pre-processing stage, is carried out using the Predicate-Argument Recognition Algorithm (PARA) from Chapter 4. Results using standard evaluation metrics show that multi-argument classification outperforms all existing systems that use a single parse tree for the CoNLL 2005 shared task. This chapter also describes ways to significantly increase the speed of multi-argument classification, making it suitable for any real-time language processing task that requires semantic role labeling

Section 5.1 outlines the results of existing approaches to SRL for the CoNLL 2005 shared task. The concept of MAC is explained in Section 5.2. Section 5.3 shows the system architecture and related work, and experimental results are shown in Sections 5.4. Section 5.5 summarizes this chapter.

5.1 Introduction

Approaches for Semantic Role Labeling (SRL) are described in detail in the proceedings of CoNLL 2004 and CoNLL 2005 shared tasks. Many have addressed SRL using Machine Learning (ML) approaches; as with the SNoW learning architecture (Punyakanok et al., 2004, 2005), Support Vector Machines (Moschitti et al., 2004, 2005; Park et al., 2004, 2005; Pradhan et al., 2003, 2004, 2005), the Perceptron Learning Algorithm (Carreras et al., 2004), EM-Based Clustering (Baldewein et al., 2004), Transformation-based learning (Higgins, 2004), and Memory-Based Learning (Kouchnir, 2004), and so on.

System	***P***	***R***	***F1***	***Features***	***Syn parses***
Punyakanok	82.28	76.78	**79.44**	18	**7**
Pradhan (*)	**82.95**	74.75	78.63	**44**	3
Haghighi	79.54	**77.39**	78.45	27	5
Marquez	79.55	76.45	77.97	22	2
Surdeanu	80.32	72.95	76.46	31	1
Tsai	82.77	70.90	76.38	25	1
Che	80.48	72.79	76.44	24	1
Moschitti	76.55	75.24	75.89	14	1
Tjongkimsang	79.03	72.03	75.37	30	1
Yi	77.51	72.97	75.17	14	1
Ozgencil	74.66	74.21	74.44	11	1
Johansson	75.46	73.18	74.30	14	1
Cohn	75.81	70.58	73.10	20	1
Park	74.69	70.78	72.68	25	1
Mitsumori	74.15	68.25	71.08	10	1
Venkatapathy	73.76	65.52	69.40	7	1
Ponzetto	75.05	64.81	69.56	14	1
Sutton	68.57	64.99	66.73	20	1
Palmer	68.60	57.80	62.74	6	1

Table 5.1. A list of results for systems on WSJ 23.

Table 5.1 summarizes the results of systems in the CoNLL 2005 shared task. The best system, by Punyakanok et al. (2005), achieves 79.44 in F_1 measurement using seven different syntactic parses (*Syn info*) (like Collins' and Charniak's parses). Pradhan et al.

(2005) use 44 features and gets the best precision (P) among the systems[1]. Haghighi et al. (2005) achieve the highest recall (R). The four top systems use more than one parse in order to achieve the optimum parsings before classification. Surdeanu et al. (2005) with 31 features achieve the best performance among systems that just use a single parse tree.

The general trend is to try to increase performance by adding more features. Instead of this philosophy, this thesis applies the concept of Multi-Argument Classification to reach the goal of better performance without additional features and without additional parsings. MAC is based on utilizing the relationship between roles in a semantic structure.

The CoNLL shared task in 2005 attracted wide attention in the SRL community. In the same year, Palmer et al. (2005) reported their system results on the same test data. All of these systems utilize Singular Argument Classification (SAC), which means classifiers label one argument at a time. One problem of singular argument classification is that such systems may produce duplicate roles. For example, a possible output list of such a role classifier is [A1 V A2 A2 TMP], in which there are two arguments assigned with the same semantic role, “A2”. According to PropBank, one of the duplicate core roles is incorrect. It can be hard to determine which one must be changed. If there is a pattern with role list [A1 V A4 A2] found in the training data, this problem can be avoided by applying the whole pattern to the system with a multi-argument classifier. The relationship in the predicate-argument structure exhibits **semantic role dependency**, manifest in the sequential order and juxtaposition of different core roles (like A0, or A1) in the predicate-argument list. Generally speaking, there is only one core role in each predicate structure[2]. This can serve as useful information for role classification.

Existing systems use post-processing to solve the problem of duplicate roles in a sentence fixing errors after initial classification. There are different heuristics

[1] (*) in Table 5.1 means a system’s result was updated after the conference.
[2] There are rare situations with more than one core role.

proposed, such as those developed by Surdeanu and Turmo (2005), Che et al. (2005), Sang et al. (2005), Ponzetto and Strube (2005), and Ozgencil and McCracken (2005). For example, Ponzetto and Strube (2005) use a simple technique for avoiding duplicate A0 or A1 role labels within the same proposition, based on argument position and predicate voice. In the case of duplicate A0 labels, if the verb is in the active form, the second assignment (after the predicate) is replaced with A1, otherwise the first one is replaced. In the same way, in case of duplicate A1 labels, if the predicate is in the active form, the first A1 assignment is replaced with A0, otherwise the second one is replaced[3].

Distinct from a post-processing approach, this research proposes a technique called Multi-Argument Classification (MAC) as an argument classifier eliminating the problem of duplicate roles and leveraging semantic role dependencies. This classifier improves the performance of the system mentioned in Chapter 3. The pre-processor, Predicate-Argument Recognition Algorithm (PARA), is used for argument identification.

5.2 Multi-Argument Classification

This section describes a new technique, Pattern-Matching (PM), as the core operation to Multi-Argument Classification (MAC). Unlike Singular-Argument Classification (SAC), where a classifier focuses on labeling each instance one by one, Pattern-Matching labels a set of arguments simultaneously.

5.2.1 Classification Model

Chapter 3 describes how Gildea and Jurafsky (2002) calculate the probability of the optimal role assignment *r** for each sentence as follows.

[3] Ponzetto and Strube (2005) do not explain what if replacing A0 (or A1) creates duplicates of A0 (or A1).

$$r^* = \operatorname{argmax}_{r_{1..n}} \mathrm{P}(r_{1...n} \mid f_{1,..n}, \mathit{predicate})$$

where $\mathrm{P}(r_{1...n} \mid f_{1,..n}, \mathit{predicate})$ represents the probability of an overall assignment of roles r_i to each of the n constituents or semantic arguments of a sentence, given the predicate, *predicate* and the various features f_i of each of the constituents.

They make the assumption that the features of the various constituents, the given predicate and each constituent's role are independent, then induce the final formula as follows.

$$r^* = \operatorname{argmax}_{r_{1..n}} \mathrm{P}(\{r_{1...n}\} \mid \mathit{predicate}) \prod_i \frac{\mathrm{P}(r_i \mid f_i, \mathit{predicate})}{\mathrm{P}(r_i \mid \mathit{predicate})}$$

$\mathrm{P}(r_i \mid f_i, \mathit{predicate})$ is the probability of a constituent's role given the above features for the constituent and the predicate. More detail is given in Chapter 3.

This is a typical ML approach for maximizing the probability of the optimal role assignment to assign roles for each sentence without utilizing role dependency learned from the training data. In Multi-Argument Classification, we apply the role dependency relationship learned from the training data to the optimal probability as follows.

(F 5.1) $$r^* = \operatorname{argmax}_{\{r_{1...n}\}} \mathrm{P}(\{r_{1...n}\} \mid f_{1,..n}, \mathit{predicate})$$

where $\{r_{1...n}\}$ is a sequential role list learned from the training data, $\mathrm{P}(\{r_{1...n}\} \mid f_{1,..n}, \mathit{predicate})$ represents the probability of an overall assignment of the role list $\{r_{1...n}\}$ to each of the n constituents or semantic arguments of a sentence, given the predicate, *predicate* and the various features f_i of each of the constituents.

We apply Bayes' rule to this probability as follows:

$$r^* = \underset{\{r_{1\ldots n}\}}{\text{argmax}} \ \mathrm{P}(\{r_{1\ldots n}\} \mid predicate) \ \frac{\mathrm{P}(f_{1\ldots n} \mid \{ r_{1\ldots n}\}, predicate)}{\mathrm{P}(f_{1\ldots n} \mid predicate)}$$

Then we make the assumption that the features of the various constituents of a sentence are independent, given the predicate and each constituent's role when the role list is applied and discard the term P($f_{1\ldots n} \mid predicate$), which is constant with respect to r:

$$r^* = \underset{\{r_{1\ldots n}\}}{\text{argmax}} \ \mathrm{P}(\{r_{1\ldots n}\} \mid predicate) \prod_i \mathrm{P}(f_i \mid \{r_i\}, predicate)$$

where $\{r_i\}$ is the i-the role in the role list of $\{r_{1\ldots n}\}$.

We apply Bayes' rule again,

$$r^* = \underset{\{r_{1\ldots n}\}}{\text{argmax}} \ \mathrm{P}(\{r_{1\ldots n}\} \mid predicate) \ \prod_i \ \frac{\mathrm{P}(\{r_i\} \mid f_i, predicate) \ \mathrm{P}(f_i \mid predicate)}{\mathrm{P}(\{r_i\} \mid predicate)}$$

Finally, we discard the feature prior P($f_i \mid predicate$) as being constant over the argmax expression:

$$r^* = \underset{\{r_{1\ldots n}\}}{\text{argmax}} \ \mathrm{P}(\{r_{1\ldots n}\} \mid predicate) \ \prod_i \ \frac{\mathrm{P}(\{r_i\} \mid f_i, predicate)}{\mathrm{P}(\{r_i\} \mid predicate)}$$

P($\{r_i\} \mid f_i$, $predicate$) is the probability of a constituent's role obtained from a role list $\{r_{1\ldots n}\}$ learned from the training data given the above features for the constituent and the predicate.

The role list $\{r_{1\ldots n}\}$ denotes there are n arguments in a test sentence, though the number of arguments in any training sentence may vary. Therefore, we need to add a mapping function to convert the role list $\{r_{1...m}\}$ of a training sentence to the role list $\{r_{1...n}\}$ of a test sentence as follows:

$$\text{M: } \{r_{1...m}\}_j \rightarrow \{r_{1...n}\}$$

where $\{r_{1...m}\}$ is the role list with *m* arguments of a training sentence *j* and $\{r_{1...n}\}$ is the role list with *n* arguments of the query sentence. The basic principle of this mapping function is to map *m* arguments of a training sentence to *n* arguments of the query.

Finally, by replacing $\{r_{1...n}\}$ with $\text{M}\{r_{1...m}\}_j$ and $\{r_i\}$ with $\{r_{ki}\}$ in the previous formula, we can obtain the probability formula.

$$\text{(F 5.2)} \quad r^* = \underset{\text{M}\{r_{1...m}\}_j}{\text{argmax}} \; \text{P}(\text{M}\{r_{1...m}\}_j \mid \textit{predicate}) \prod_i \frac{\text{P}(\{r_{ki}\} \mid f_i, \textit{predicate})}{\text{P}(\{r_{ki}\} \mid \textit{predicate})}$$

where $\text{M}\{r_{1...m}\}_j$ is the role list generated by the mapping function M from the *j*-th training sentences with *m* arguments of the training data to the role list $\{r_{1...n}\}$ for the test sentence with *n* arguments, and $\{r_{ki}\}$ denotes the *k*-th role of $\{r_{1...m}\}_j$ ($1 <= k <= m$) corresponding to the *i*-th argument of $\{r_{1...n}\}$.

In summary, each argument in the query is mapped to a corresponding argument in a knowledge pattern. The mapping function is described in the following section.

5.2.2 Mapping Algorithm

There are four considerations essential to the function that maps a knowledge pattern learned from the training data to a new query sentence: i) where to start matching two patterns; ii) how to deal with different numbers of arguments between the knowledge and query patterns; iii) how to compute similarity between an argument in a knowledge pattern and an argument in a query pattern and iv) how to measure the quality of the matching.

i) Where to start

The first consideration is simply solved by looking for the most common instance in a knowledge pattern and a query one, given the predicate. In this discussion, an instance is an argument in a predicate-argument structure.

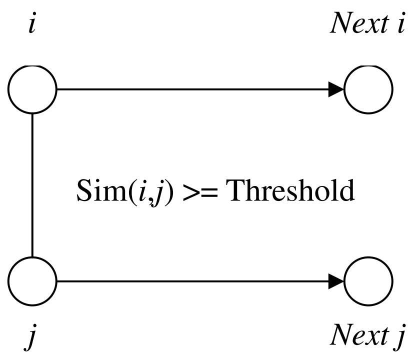
ii) Mapping of different arguments

Empirically there is very low coverage (about 0.46) to match query sentences with training sentences that have the same number of arguments. Instead of only matching two patterns with the same number of arguments, this thesis proposes an alternative way to increase the coverage. The principle is based on semantic role dependency, in which core roles (like A0 or A1) are more essential than adjuncts (like AM-TMP, or AM-LOC). We need to estimate similarity between an argument (or instance) in a knowledge pattern and an argument in the query. If two instances (one in the knowledge pattern and the other in the query) are considered highly similar (Case 1), we can try to match the next instances in both patterns. If two instances are not similar, there are two kinds of situations. One is to match the current instance in the knowledge pattern with the next instance in the query (Case 2). The other is to match the next instance in the knowledge pattern with the current instance in the query (Case 3). The other two final circumstances are unmatched instances in the query (Case 4), and unmatched instances in the knowledge pattern (Case 5).

These five cases are more formally described as follows:

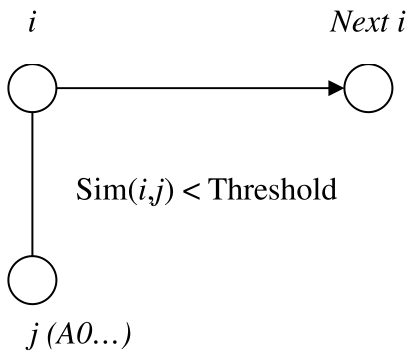
Case 1: if there exist a query instance *i* and a corresponding knowledge instance *j*, and both instances are similar (or their similarity is no less than a threshold), try to match the next instance in the knowledge pattern with the next instance in the query. This is the case when two instances are considered as highly similar then try to match the next instances in the query and knowledge patterns.

(Case 1)



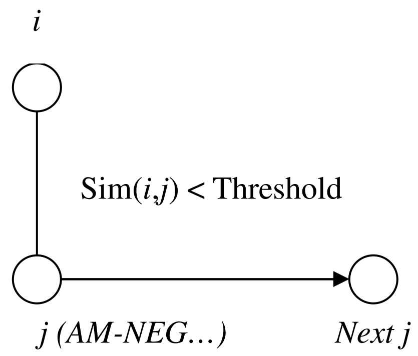
Case 2: if there exist a query instance *i* and a corresponding knowledge instance *j*, both instances are not similar (or their similarity is below a threshold) and the role of the knowledge instance *j* appears to be one of the core roles (i.e. A0 to A5 and AA) (as opposed to a non-core or adjunctive role), try to match the current instance in this knowledge pattern with the next instance in the query. The reason to keep the current knowledge instance is to try to increase the coverage. It is rare to have two patterns exactly the same due to data sparseness. For example, a query sentence "They will come" and a training sentence, "They come" is not matched due to different number of argument. But they can be considered as highly similar if the second argument "will" in the query sentence is skipped when matching two sentences. Such a skipped argument like "will" in the query can be labeled in advance by the pre-process step described in Section 3.4.

(Case 2)



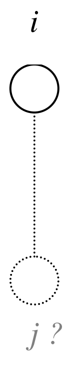
Case 3: if there exist a query instance *i* and a corresponding knowledge instance *j*, both instances are not similar and the role of the instance *j* in the knowledge pattern appears to be a non-core label (like AM-MOD AM-NEG or AM-DIS), try to match the next instance in the knowledge pattern with the current instance in the query. Such non-core roles are optional to a query pattern, which is to say that not all sentences have them. This means they can be skipped. This is the complement situation to Case 2 where all non-core or adjunctive roles in the knowledge pattern can be skipped in order to increase coverage.

(Case 3)



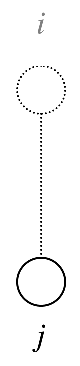
Case 4: if there does not exist an argument *j* in the knowledge pattern, keep a default probability to this query instance *i* to avoid zero frequency.

(Case 4)



Case 5: skip all extra corresponding knowledge arguments shown as the following figure.

(Case 5)



After mapping, all patterns from the pattern base are mapped to role lists with the same number of arguments in the query. It remains now to measure the similarity of each argument in the query role list with each corresponding argument in the role list of each pattern in the knowledge base.

iii) Similarity Function

The calculation of similarity between an instance in the knowledge pattern and its corresponding instance in the query is based on the feature space. The distance between two points or instances in the feature space is estimated by Euclidean distance, which is shown in (F 5.3)

Distance metric (Euclidean distance):

(F 5.3) $D(x_i, x_j) = \sqrt{\Sigma\,(a_r(x_i))-a_r(x_j))^2}$

where r =1 to n (n is the numbers of different classifications)

$a_r(x)$ means the r-th feature of an instance x.

If instances, x_i and x_j, are identical, then $D(x_i, x_j)=0$

Otherwise, $D(x_i, x_j)$ represents the vector distance between x_i and x_j

x_i is an instance in the query and x_j is an instance in the knowledge pattern.

Therefore the similarity function is defined as

$$\mathrm{Sim}(i, j) = (\text{number of features} - D(i, j)) / (\text{number of features})$$

If all features are the same between two instances, D(*i*, *j*) is zero and Sim(*i*, *j*) is 1.0. If there are different features between two instances (for example two features are not the same), the score of similarity by Sim(*i*, *j*) will be less than 1.0. For example, if there are five features for calculation and two different, D(*i*, *j*) is two and Sim(*i*, *j*) is 0.6, which is (5 – 2) / 5.

Threshold

The threshold utilized in the first three cases is initially set to 1.0, which means all features in the knowledge instance must be the same with the ones of the query pattern. This threshold value was arrived at through trial and error. Table 5.2 shows the results of label accuracy with two different thresholds on WSJ 24.

Thresholds	**Lacc**
1.0	89.38
>=0.856	82.31

Table 5.2. Results of different thresholds on WSJ 24 with known arguments

It includes a full similarity (threshold=1.0) (all features in a query and a knowledge must be the same), and a threshold allowing one feature different between both instances (threshold >= 6/7 or 0.856). The "1.0" threshold offers much better performance compared to the one by (threshold>=0.856). It suggests that all basic features are essential to PM to achieve better performance.

iv) What is the quality estimation

The fourth issue mentioned early in this section is to find the quality of a match between a pattern in a training sentence and a query pattern in the query sentence by a statistical method as follows:

$$\text{(F 5.4)} \qquad Qj = \mathrm{P}(\mathrm{M}\{r_{1\ldots m}\}_j \mid predicate) \prod_i \frac{\mathrm{P}(\{r_{ki}\} \mid f_i, predicate)}{\mathrm{P}(\{r_{ki}\} \mid predicate)}$$

where Qj denotes the quality estimation for the j-th training sentence.

This formula is effectively (F 5.2) except that argument maximization is not used. After all quality probabilities for patterns in the training data are calculated, the system can select a pattern with the highest quality probability in the training sentences. In order to avoid zero probabilities, the default probability for each argument in the test sentence is set 0.00001.

Figure 5.1 shows the algorithm of the mapping function. It is the function that measures the quality of the match between the query and a training pattern.

```
Quality (K,T, R)
{   Pattern K, T; // a knowledge pattern K and the query pattern T
    Instance i, j; // index i for the query pattern, and j for the knowledge pattern
    Role List R=””; // a role list mapped from K to T

    i = verb(T);    j = verb(K); // locate index to the predicates
    before_verb_done=0;
    i = next(i); j = next(j); // if before_verb_done then i = i--else i = i++;
    Quality = 1.0;

    While exist i and j
    {       Probij = Prob($r_j$ | $f_i$) if Sim(i , j) >= Threshold
                                else Probij = default; // default = 0.00001
            if Probij > default // Case 1
                  Quality*=Probij;
                  Addrole(R, $r_j$);
                  i = next(i); j=next(j);
            else if (core_role($r_j$)) // Case 2:
                  Quality*=Probij; // Sim(i , j) < Threshold
                  Addrole(R, “X”); // “X” denotes an unlabeled role
                  i = next(i);
            else // Case 3:
                  j = next(j);

            if ( ! before_verb_done && !i && !j )
            {     i = verb(T);    j = verb(K); // locate index to the predicates
                  before_verb_done=1;
                  i = next(i); j = next(j); // i = i++; j=j++;
            }
    }
    while (number_of_uncompared_pattern(i))
    {       Quality*= default; // Case 4:
            Addrole(R, “X”); // “X” denotes an unlabeled role
}
    // There is no action for Case 5:

    return Quality & R;
}
```

Figure 5.1. The Matching Algorithm (MA).

In the matching algorithm in Figure 5.1, there are two patterns, *K* denoting the knowledge pattern and *T* the query. The matching algorithm returns a role list *R*, and the quality of matching between *K* and *T*. Firstly, MA starts from both predicates in the knowledge and query patterns then tries to match arguments before and after the predicates. A flag *before_verb_done* is used for checking if the instances prior to the predicates have been matched or not. After the initialization, MA moves the current query instance *i* to the next one that is the instance before the predicate in the query and the current knowledge instance *j* to the instance prior to the predicate in the knowledge pattern. After setting the initial return value Quality to be 1.0, MA tries to match two instances *i* and *j*. If the similarity of *i* and *j* is no less than the threshold (set as 1.0), MA calculates the probability of Prob($r_j \mid f_i$) for the role r_j in the knowledge instance *j* with the features set in the query instance *i*.

The probability Prob($r_j \mid f_i$) is the short form of Prob ($r \mid pr, vo, pt, di, hw, pa, pp$), which is calculated using the formula in (F 5.5):

$$\text{(F 5.5)} \quad \text{Prob}\,(r \mid pr, vo, pt, di, hw, pa, pp) = \frac{\#\,(r, pr, vo, pt, di, hw, pa, pp)}{\#\,(pr, vo, pt, di, hw, pa, pp)}$$

Hence *r* denotes r_j and f_i = { *pr, vo, pt, di, hw, pa, pp* } for the feature set, predicate *pr*, voice *vo*, phrase type *pt*, distance *di*, headword *hw*, path *pa* and preposition *pp*.

After calculating Prob($r_j \mid f_i$) for the query instance *i* and the knowledge instance *j*, MA checks if the probability is greater than the default. If it is greater, which means the query instance *i* and the knowledge instance *j* are similar (Case 1), MA multiplies *Quality* with Prob($r_j \mid f_i$) according to (F 5.4), adds r_j to the role list *R* and moves the both current instances in the query and the knowledge pattern to the next ones.

If there is no similarity between the query instance *i* and the knowledge instance *j* and the label in the knowledge instance *j* is a core role (like A0, or A1) (Case 2), MA multiplies *Quality* with a default value (which is 0.00001) to show that the query instance *i* has no corresponding label for it, adds an unlabeled role (denoted as "*X*" here) to the return role list *R* and moves the current query instance to the next one. MA keeps the current knowledge instance because it has a core-role label, which is essential to the knowledge pattern.

If there is no similarity between the query instance *i* and the knowledge instance *j* and the label in the knowledge instance *j* is a non-core role (like AM-TMP) (Case 3), MA just moves the current knowledge instance to the next one for further matching with the current query instance.

After matching instances before the predicates, the *before_verb_done* flag is set true to indicate argument matching prior to the predicates is done. MA thus carries on matching instances after the predicates. .

Finally, MA adds an unlabeled role to the return role list *R* and multiplies a default value to the return value Quality for each unmatched instance in the query according to Case 4 to avoid zero frequency.

5.2.3 An Example of Mapping Algorithm

Example (5.1) is a partial sentence found in the training data for the predicate "resign" and (Ex. 5.2) is a test sentence with the predicate "resign".

(Ex. 5.1) "… [$_{A0}$ David A. Entrekin], [$_{R\text{-}A0}$ who] [$_{V}$ resigned] [$_{AM\text{-}TMP}$ Monday]."
(Ex. 5.2) "… [$_{Arg1}$ Entrekin] [$_{V}$ resigned] [$_{Arg2}$ in 1988]."

In the training example or knowledge pattern, (Ex. 5.1), there are three arguments apart from the predicate, "resign". The first argument, "David A. Entrekin" is labeled with "A0", the second, "who" with "R-A0" and the third, "Monday" with "AM-TMP". In the test example or query pattern, (Ex. 5.2), there are two arguments “Entrekin” denoted as *Arg1*, and “in 1988” *Arg2*. *Arg1* and *Arg2* are temporary labels, which must be replaced with proper semantic roles after classification. Instead of matching both patterns from the first arguments, the Mapping algorithm (MA) checks the similarity of these two patterns beginning from the predicates in both patterns. Then MA matches each argument before and after the predicates in the query pattern to the corresponding argument in the knowledge pattern.

According to the mapping algorithm (MA) in Figure 5.1, MA locates both initial indexes on the predicate instances, then moves the knowledge index to the instance of “who” (denoted as KI_2) and the query index to the instance of “Entrekin” (denoted as QI_1). MA calculates the similarity of the two instance, and finds Sim(KI_2, QI_1) is 0.0 because both headwords are different. Then MA moves the knowledge index to the instance of “David A. Entrekin” (denoted as KI_1) according to Case 3 because the role (R-A0) of KI_2 is not a core role. When comparing KI_1 and QI_1, MA finds Sim(KI_1, QI_1) =1.0 if just considering the headword and phrase type features as a feature space, then according to Case 1 MA multiplies the Quality function with the probability Prob(A0 | Entrekin, NP) , which is assumed as 1.0 in this example. There are no unmatched instances in the query and knowledge patterns prior to the predicates, so MA sets the *before_verb_done* flag true and moves the knowledge index to the instance of “Monday” (denoted as KI_3) and the query index to the instance of “in 1988” (denoted as QI_2). MA finds the Sim(KI_3, QI_2) is 0.0 again due to different headwords then multiplies the quality function with the default value (0.00001). Finally, MA returns the value of the quality function, Prob(A0 | Entrekin, NP)* default , which is 1.0*0.00001 in this example.

Figure 5.2 illustrates the final alignment of these two patterns according to MA. Each instance or argument box contains some features such as head word, phrase type and so on.

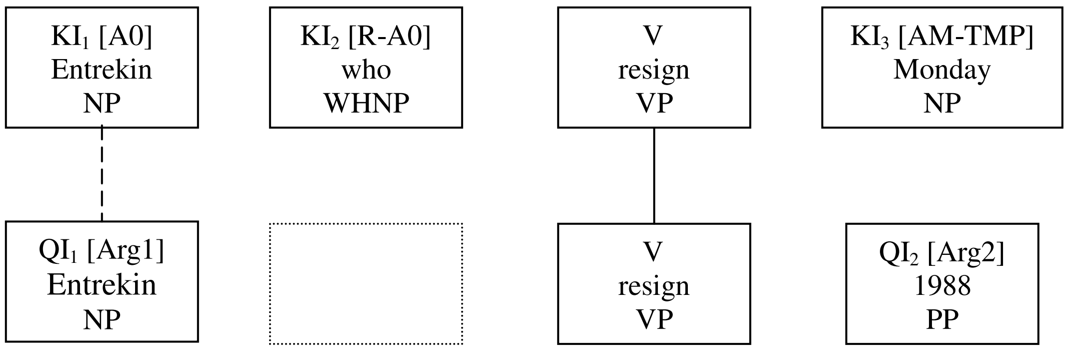


Figure 5.2. A related alignment of pattern boxes with the predicate for (Ex. 5.1) and (Ex. 5.2).

5.2.4 Unlabeled Arguments

MA is designed for matching two patterns with different arguments. It helps to increase the overall coverage from 0.46 (if only marching patterns with the same number of arguments) to 0.78. This is still not good enough compared to statistical singular-argument classifiers (like PML in Chapter 3). The reason for this low coverage is sparseness of data. For example, the return role list *R* for the query sentence in Ex. 5.2 is [A0 V X] matched with the training sentence with the role list [A0, R-A0, V, AM-TMP]. Apparently, there exists an unlabeled argument "in 1988", which can be further assigned with "AM-TMP" by other approaches like existing singular-argument classifiers.

We propose a simple argument labeller as follows to fill unlabeled arguments.

$$\underset{r}{\operatorname{argmax}} \quad P(r \mid f, predicate)$$

where P(*r* | *f*, *predicate*) represents the probability of an assignment of role *r* (except any core role appears in the labelled role list to avoid core role duplication) to each of the unlabeled arguments of a sentence after MA, given the predicate, *predicate* and the features *f* (including headword, distance, voice, preposition, phrase type and path) of the argument. P(*r* | *f*, *predicate*) is calculated using the formula in (F 5.5).

By handling unmatched arguments with this simple argument labeller, the recall reaches from 0.78 to 0.86.

5.2.5 Complete PM Model

The complete model for Pattern-Matching (PM) is thus a combination of MAC and SAC. PM tries to find all suitable patterns from the training data using the mapping algorithm described in Section 5.2.2, selects the best one from the pattern base according to the quality probabilities from the mapping algorithm using MAC, and classifies any unlabelled arguments in the best pattern with SAC like a simple argument labeller in Section 5.2.4.

The goal of selection is to find the knowledge patterns with the highest Quality, calculated by MA described in Section 5.2.2. The procedure for PM is shown in Figure 5.3.

```
Procedure of Pattern-Matching with SAC
For all knowledge patterns
    apply Mapping Algorithm for the query and knowledge patterns
Select the best knowledge pattern according to their quality probabilities
Use SAC to classify the unlabelled arguments
```

Figure 5.3. Procedure of the basic PM model and SAC.

5.3 System Architecture

There are two stages to the system proposed here, the building stage, which is comparable to training in a stochastic system or data preparation, and the testing stage. The building stage is only for storing all feature representations of the training instances in memory and no calculations are necessary.

The testing stage shown in Figure 5.4 classifies new instances by matching their feature representation to all instances in memory in order to find the most similar instances.

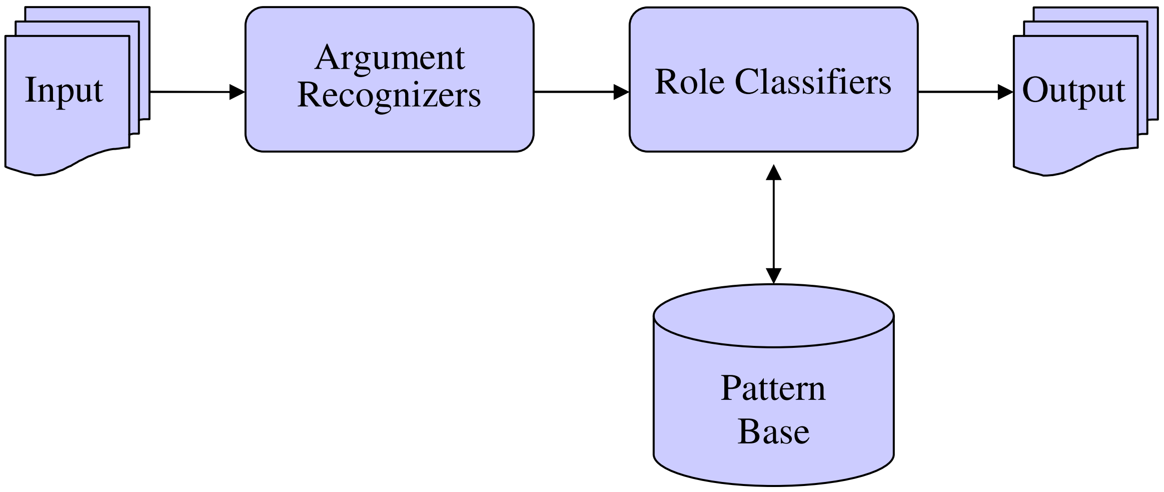


Figure 5.4. Illustration of System Architecture for the testing stage.

Predicate	–	The given predicate lemma.
Voice	–	Whether the predicate is realized as an active or passive construction.
Phrase Type	–	The syntactic category (NP, PP, S, etc.) of the phrase corresponding to the semantic argument.
Distance	–	The relative displacement from the predicate, measured in intervening constituents (negative if the constituent is to the left of or prior to, positive if it is to the right of or after, the predicate).
Head Word	–	The syntactic head of the phrase.
Path	–	The syntactic path through the parse tree, from the parse constituent to the predicate being classified.
Preposition	–	The preposition of an argument in a PP such as *during, at, with*, and so on.

Table 5.3. A list of features used in Pattern-Matching.

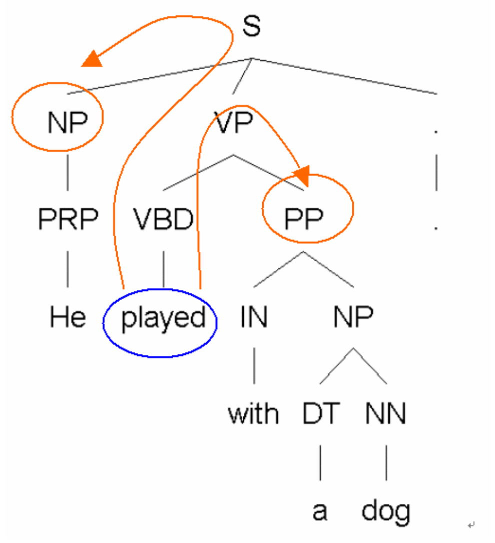


Figure 5.4. An example of a parse tree.

Pattern Base

Bosch et al. (2004) introduces a singular instance (SI) knowledge base. Each singular instance (SI) is stored in the following format $SI = (r, f_1, f_2,\ldots, f_n)$. "$r$" denotes the semantic role such as A0, A1, AM-TMP, and so on; and $f_1, f_2,\ldots, f_n$ denote different features, such as Predicate, Voice, Phrase type, Distance and so on. These features, Predicate (pr), Voice (vo), Phrase Type (pt), Path (pa), Distance (di), Head Word (hw), and Prepostition in a PP (pp), are introduced in Chapter 2 and summarized in Table 5.3. Here is an example instance for the first argument of the predicate "came" in Figure 5.4, with basic features:

(A0, play, active, NP, –1, he, NP ↑ S ↓ VP ↓ VBD, NULL)

The format maps each argument to an eight-dimensional vector space, which includes semantic role, predicate, voice, phrase type, distance, head word, path, and preposition. A NULL note is added if there is no preposition.

This research keeps the sequential order of predicates and their arguments as a pattern. Each pattern stores a set of instances with a set of features. The format of each pattern is $P = \{SI_1, SI_2, \ldots, SI_m\}$, where each instance $SIi = (r, f_1, f_2,\ldots, f_n)$, i =1 to m. This format is called multiple-instance format. It is different from that of Bosch et al. (2004) which assumes unordered bags with a single label used in instance-based learning,.

All training data is stored in the multi-instance format. The following is an example of the multi-instance knowledge base for Figure 5.4.

Pattern = { (A0, play, active, NP, –1, he, NP ↑ S ↓ VP ↓ VBD, NULL),
(V, play, active, VP, 0, play, VBD, NULL),
(A1, play, active, PP, +1, dog, PP ↑ VP ↓ VBD, with) }

The headword of a PP is the NP after a preposition such as "with" in Figure 5.6.

Argument Recognizers

In the proposed system, the Predicate-Argument Recognition Algorithm (PARA) described in Chapter 4 is used as an argument recognizer to identify predicates and arguments related to predicates. It forwards the predicates and their arguments to the next stage, which is for role classifiers. Details of PARA have been explained in Section 4.3.

Argument Classifiers

Argument classification in the complete model includes two role classifiers, a multi-argument classifier, Pattern-Matching (PM) described in detail in Sections 5.2, and a singular argument classifier, called PML modified from Palmer et al. (2005) described in Chapter 3.

Simple Role Labeling Algorithm (SLA)

The following simple algorithm, called simple role labeling algorithm (SLA), is used as a baseline for subsequent experiments.

1) Find the closest NP before the predicate; if the voice is active, then label this NP as A0, otherwise as A1.
2) Find the closest NP after a predicate; if the voice is active, then label this NP as A1.
3) Between a labeled role prior to the predicate, if there is an argument with WHNP, then label this argument as a reference of the labeled role, such as R-A0 in an active form or R-A1 in a passive form.
4) Label each MD argument as AM-MOD.
5) Label each negative argument, such as *not, n't*, and so on, as AM-NEG.

This algorithm is modified from the baseline used in the CoNLL shared tasks proposed by Erik Tjong Kim Sang (Carreras and Palmer, 2005, p.161). The difference is this modified algorithm does not tag target verbs and successive particles as a predicate "V" and does not label an argument after a predicate in passive voice as A0.

5.4 Experiments and Results

Data used in this chapter is the dataset released on March 2005 from CoNLL-2005[4], which includes Wall Street Journal sections with Charniak's (2000) and Collins' (1999) parse-trees. Charniak's parse tree is accepted as input to the system due to its better performance in WSJ (Carreras and Marquez, 2005). Evaluation is carried out using precision, recall and F_1 measure of the predicated arguments. For consistency, the performance of systems is tested using the official evaluation script from CoNLL 2005, *srl-eval.pl*.

This section shows experimental results on one part of the Brown corpus and Wall Street Journal (WSJ) Sections 21, 23, and 24 using different training data sets (WSJ 21, WSJ 15 to 18, and WSJ 02 to 21). It includes results for different approaches for role classification with known arguments in Section 5.4.1. Another singular-argument classifier, the Kth-Nearest Neighbour algorithm is described in Section 5.4.2 to offer another comparison to PM. Contribution of features for PM and learning curves using different sizes of training data are shown in Sections 5.4.3, 5.4.4. Section 5.4.5 shows test results on WSJ 24, WSJ 23 and the Brown corpus with PM and PARA. Execution times for different training data sets and speed improvement are described in Section 5.4.6. Section 5.4.7 shows a comparison with other systems.

5.4.1 Results for WSJ 24 with known arguments

Table 5.4 shows the results for several approaches, when used with known arguments (i.e. the systems are given the correct arguments for role classification). All training data (WSJ02-21) with Charniak's parses are included. The modified baseline or simple role labeling algorithm (SLA) described in Section 5.4.4 only reaches 71.97% in precision and 64.67% in recall. It suggests that about 72% of assignments for A0 and A1 can be simply finished without any ML approaches and such assignments cover about 65% of classification task.

[4] http://www.lsi.upc.edu/~srlconll/soft.html

Approach	**P**	**R**	**F1**
SLA(Baseline)	71.97	64.67	68.12
PML	85.53	85.65	85.59
PM without PML	87.67	85.85	86.75
PM	**88.89**	**87.65**	**88.27**

Table 5.4. Results obtained by different algorithms on WSJ Section 24 with known arguments.

The modified version of the classifier from Palmer et al. (2005) (*PML*) provides 85.59 in F_1 and the performance of the basic model (PM without PML) estimation is F_1: 1.16 improved compared to PML itself. The complete model (PM), combined with MAC and SAC, achieves the best results on Precision (88.89), Recall (87.65), and F_1 measurement (88.27) and offers the best solution on all test datasets compared to results in Chapter 3. It suggests PM, utilizing role dependencies existing in semantic roles, helps to increase F_1 by 3.0 over PML.

(System 1,System 2)	**(PM, PML)**
(C,C)	81.74%
(C,W)	6.32%
(W,C)	4.97%
(W,W)	6.97%

Table 5.5. Analysis obtained by different algorithms on WSJ Section 24 with known arguments.

Table 5.5 shows an analysis between PM and PML. The second row (*C,C*) shows where both systems have correct labelling, which is 81.74% of assignments. The third row (*C,W*) shows 6.32% of assignments for the case when the first system, PM, labels correctly but the second, PML, wrongly. There is 4.97% of assignments where PML offers better classification than PM shown in the fourth row (*W,C*). It is 6.97% of assignments in the fifth row where both classifiers fail to offer correct classification.

This table suggests the best improvement (4.97%) of labelling by merging PM with PML and another 6.97% improvement for more precise labelling using other ML approaches or adding more features in the systems. Both of these topics are of interest for future research.

5.4.2 KNN

In order to compare the multi-argument classifier and singular-argument classifiers, this research also includes another instance-based learning method, the kth-nearest neighbour algorithm, which uses the same distance metric, Euclidean distance in (F 5.3) for comparison of a multi-argument classifier and a singular-argument classifier.

In machine learning, the Kth Nearest Neighbor (KNN) algorithm is a method for classifying queried instance or objects based on the closest training data in the feature space. This algorithm is a type of instance-based learning. The Kth Nearest Neighbor (KNN) algorithm is suitable for instances mapped to points/classifications in n-feature dimensions. This research implements this algorithm to show another method for instance-base learning.

Generally, the three principal components for KNN are i) the instance base, which stores all instance or training data in the memory or knowledge base; ii) the distance matrix, which is used to store the distance between a query instance and all instances in the memory; and iii) the classification function to make the final decision.

The KNN algorithm starts with a point or a query, *q*, in the feature space, and assigns the class, *x*, that is the most frequent class label among the k nearest training samples. Usually the distance between two points or instances in the feature space is estimated by Euclidean distance, which is calculated using (F 5.3)

The training phase of the algorithm consists only of storing the feature vectors and class labels of the training data into a memory or knowledge base. In the test or classification phase, the same features as before are computed for the test sample (whose class is not known). Distances from the new vector to all stored vectors are computed and k closest samples are selected. The new point is predicted to belong to the most numerous class within the set. The classification function is defined in (F 5.6).

Classification function

Given a query/test instance x_q to be classified, let x_1,.. x_k denote the k instances from the training instances that are nearest to x_q. The Classification Function is

(F 5.6) $F\^(x_q) \leftarrow \text{argmax} \sum \delta (v,f(x_i))$

where i =1 to k, v =1 to m (the number of different roles);

$f(x_i)$ denotes the role assigned to x_i ;

δ (a,b)=1 if a=b, or δ (a,b)=0, otherwise;

v denotes a semantic role for each instance of training data

Instance base

All the training data is stored in the following format, similar to the one used by Bosch et al. (2004): "Role, Predicate, Voice, Phrase type, Distance, Head Word, Path". Here is an example instance for the second argument of a predicate "take" in the training data.

(A0, take, active, NP, –1, classics, NP $\uparrow$ S $\downarrow$ VP $\downarrow$ VBD)

This format maps each argument to six feature dimensions plus one classification.

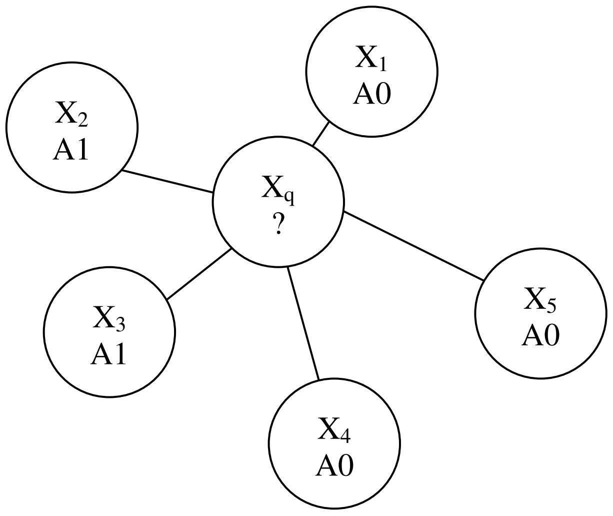


Figure 5.5. An example of KNN.

The basic algorithm is easy to implement for comparison of a singular-argument classifier and a multi-argument classifier based on the same distance metric.

Figure 5.5 depicts KNN in action. If there is a query X_q and five nearest training instances, X_1 with a semantic label A0, X_2 with A1, X_3 with A1, X_4 with A0, and X_5 with A0. Assume the distances for each training instance to the query are 1, 2, 2, 3, 3 respectively. It is calculated according to how many different features of each training instance compared to the query. For example, the number of different features between X_4 and X_q is 3. If we select the nearest neighbour (k=1), which is X_1, the role for the query is assigned with A0. If we select the three nearest neighbours (k=3), the majority of votes is A1 for the query. If we select the five nearest neighbours (k=5), the label for the query is A0 again.

Table 5.6 shows the results using KNN with different values of k. The basic Kth-Nearest Neighbour (KNN) algorithm with k=3 appears to be the best KNN approach, which achieves 81.13 in $F_{1,}$ but its performance is F_1: 5.62 beyond that of PM without PML. The comparison suggests that the role dependency by PM helps to improve performance of the most. Other SAC algorithms could be tried, but it is sufficient to see that PM is at least not the worst.

Approach	**P**	**R**	**F1**
KNN (k=1)	81.15	80.63	80.69
KNN (k=3)	81.24	81.02	81.13
KNN (k=5)	81.01	80.83	80.95
KNN (k=7)	80.59	80.45	80.52
KNN (k=9)	79.96	79.77	79.87
KNN (k=11)	79.43	79.28	79.35

Table 5.6. Results obtained by KNN algorithms on WSJ Section 24 with known arguments.

5.4.3 Contribution of Features for PM

Table 5.7 shows the result using all features (*ALL*), and the contribution of each feature in Precision (P), Recall (R), and F_1 measurements.

	P	**R**	**F1**
ALL	88.89	87.65	88.27
- *Voice*	88.52	87.02	87.77
- *Head Word*	85.52	83.93	84.72
- *Phrase Type*	85.21	**83.03**	84.11
- *Preposition*	**84.77**	**83.03**	**83.89**
- *Distance*	88.81	87.56	88.18
- *Path*	87.12	85.89	86.50

Table 5.7. Contribution of each feature on WSJ 24, with known arguments.

In contrast to the three main features Head word, Preposition, and Distance in Table 3.11 mentioned in Section 3.6.3, Phrase Type, Preposition, and Head word are the three features whose removal decreases the performance of the complete system by a large amount. The distance feature plays a key role in overall performance of PML but is the least influential in PM because of the usage of multi-argument classification. When using PM, the related distance is implicitly included when matching two patterns. The path feature is the fourth most influential factor on performance for role classification, and the voice feature has the least detrimental effect, along with the distance feature, on the performance of this system. Both features (path and voice) have the same influence in PM and PML.

5.4.4 Learning Curve

Table 5.8 shows the results for different sizes of training data. D*n*K in Table 5.8 denotes the first *n* k (i.e. *n*-thousand) sentences from the training data sets, WSJ 02 to 21. It indicates more training data can provide better performance for known arguments of WSJ 24. This happens not only in F_1 but also Precision (P), Recall (R), and Label accuracy (Lacc).

Training sets	**P**	**R**	**F_1**	**Lacc**
D1k	80.52	77.33	78.89	81.76
D5k	84.40	82.36	83.37	85.43
D10k	85.99	84.36	85.17	86.95
D20k	87.61	86.20	86.90	88.56
D30k	88.53	87.19	87.85	89.45
D40k	88.88	87.66	88.27	89.83

Table 5.8. Results for different training datasets on WSJ 24, with known arguments.

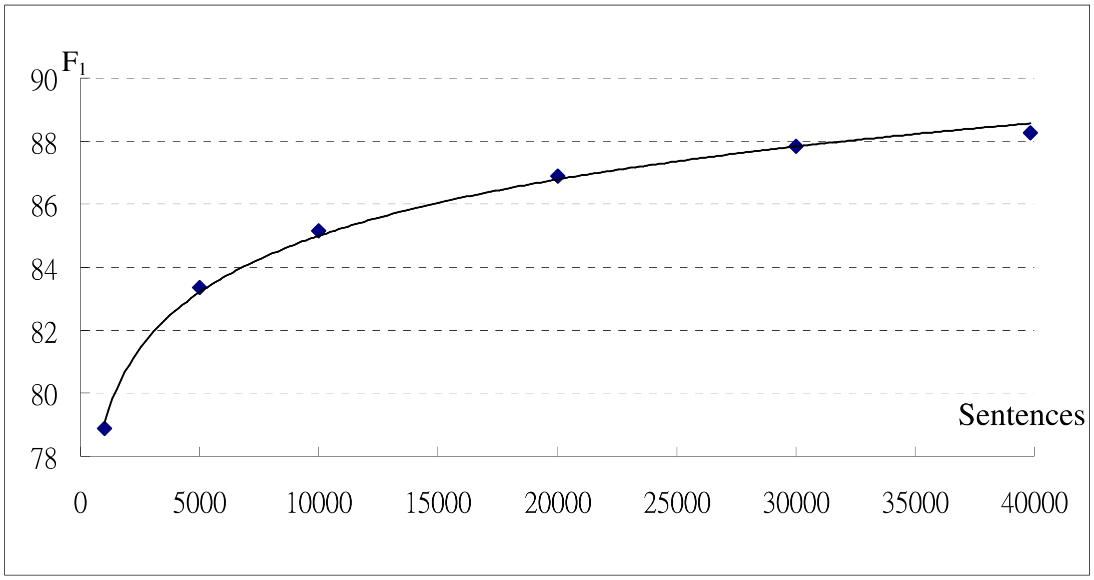


Figure 5.6. The learning curve for F$_1$ obtained with different sizes of training data on WSJ 24 for role classification.

The increase of performance is not proportional to the size of training data. The log-like curve for different sizes of training data is shown in Figure 5.6, and it is similar to curves for other systems presented in McCracken (2005)**.** This curve suggests the performance improvement decreases as training data increases.

5.4.5 Results for Test datasets with Auto parses & PARA

Table 5.9 shows performance (on WSJ 24, WSJ 23 and the Brown corpus) of the complete model (PM) using auto parses (Charniak's parser) and PARA as the pre-processor to recognize all related arguments. Compared to the results in Table 5.4, the performance on WSJ 24 drops by F$_1$:13.87 (from 88.27 to 74.40). The large drops in F$_1$ are mainly caused by the combined errors from the auto parser, the pre-processor PARA and the role classifier PM. This is well known as the propagating of error in Natural Language Processing. Errors in the previous stages are carried forward to subsequent processing steps. For example, the first degradation is from parsers. Carreras and Marquez (2005) claim the optimum achievable accuracy on these given parses from Charniak's parser is 87.49 in F$_1$ measurement on the WSJ 24 data before any SRL applications. The second one is from PARA. Table 4.7 (in Section 4.5.3) shows the performance by PARA on WSJ 24 using Charniak's parses is 80.89, which decreases another 6.6 before role classification. PM decreases another 6.49 after role classification. Carreras and Marquez (2005) conclude "beyond pipelines, what type of architectures and language learning methodology ensures a robust performance of processors".

Test dataset	P	R	F1	Lacc
WSJ 24	75.88	72.98	74.40	92.50
WSJ 23	78.04	75.20	76.60	93.34
Brown	69.33	63.44	66.25	84.67

Table 5.9. Results for different training datasets on WSJ 24 with Charniak's parses and PARA.

Table 5.9 also shows the results on WSJ 23 are about F_1:2.0 better than that by WSJ 24. This increase is because the performance by PARA on WSJ 23. It is about F_1:2.0 better than WSJ 24 described in Section 4.5.4. The results on WSJ 23 for each role are shown in Table 5.10. Generally speaking, performance on core roles is better than on adjuncts, except for the modal, and negation tags. This is because more training examples appear for core roles than for adjuncts.

The results on the Brown corpus show the performance drops by more than 10 points in F_1 compared to WSJ 23. It is mainly caused by processing propagating errors as described early. Table 5.9 also shows such errors affect results even more in the domain of the Brown corpus. Another area for future work can be looking for ways to minimize the impact of different domains.

WSJ 23	Precision	Recall	$F_{\beta=1}$
Overall	78.04%	75.20%	76.60
A0	84.31%	85.18%	84.74
A1	78.86%	76.98%	77.91
A2	70.83%	61.26%	65.70
A3	68.84%	54.91%	61.09
A4	66.67%	62.75%	64.65
A5	100.00%	60.00%	75.00
AM-ADV	59.07%	55.34%	57.14
AM-CAU	64.91%	50.68%	56.92
AM-DIR	35.53%	31.76%	33.54
AM-DIS	76.25%	76.25%	76.25
AM-EXT	50.00%	37.50%	42.86
AM-LOC	62.54%	51.52%	56.50
AM-MNR	59.33%	51.74%	55.28
AM-MOD	97.42%	95.83%	96.61
AM-NEG	95.18%	94.35%	94.76
AM-PNC	46.39%	39.13%	42.45
AM-PRD	0.00%	0.00%	0.00
AM-REC	0.00%	0.00%	0.00
AM-TMP	73.58%	72.49%	73.03
R-A0	85.84%	86.61%	86.22
R-A1	80.28%	73.08%	76.51
R-A2	80.00%	50.00%	61.54
R-A3	0.00%	0.00%	0.00
R-A4	0.00%	0.00%	0.00
R-AM-ADV	0.00%	0.00%	0.00
R-AM-CAU	0.00%	0.00%	0.00
R-AM-EXT	0.00%	0.00%	0.00
R-AM-LOC	73.68%	66.67%	70.00
R-AM-MNR	25.00%	16.67%	20.00
R-AM-TMP	62.69%	80.77%	70.59

Table 5.10. Details for each semantic role on WSJ 23, with Charniak's parses and PARA.

5.4.6 Time & Speed Improvement

Up to now, we have shown the accuracy performance by different approaches. This section discusses execution time and speed improvement for different approaches based on the same conditions.

Table 5.11 shows the execution time, in seconds per sentence, for Palmer et al. (2005), KNN, and PM (before speed improvement).

Approach	**T (sec/sen)**
PML	0.8
KNN (k=3)	1.5
PM (before speed improvement)	3.0

Table 5.11. Speed results for different approaches on WSJ 23 using Charniak's parses and PARA.

The execution speed of "PM before speed improvement" is the slowest one for the task of role classification. In order to increase execution speed without sacrificing accuracy, we propose a controlling strategy using Maximum Suitable Pattern (MSP) number. MSP limits numbers of suitable patterns for a query pattern. The formula is shows in F 5.7 as follows.

$$\text{(F 5.7)} \quad r^* = \underset{M\{r_{1\ldots m}\}_j}{\overset{\text{Suitable(j)} <= \text{MSP}}{\text{argmax}}} \; P(M\{r_{1\ldots m}\}_j \mid predicate) \prod_i \frac{P(\{r_{ki}\} \mid f_i, predicate)}{P(\{r_{ki}\} \mid predicate)}$$

i

where Suitable(j) denotes the number of suitable knowledge patterns found.

The difference between the basic formula (F 5.2) and the improved one (F 5.7) is the constraint Suitable(j) <= MSP. Once PM has found enough suitable patterns (Suitable(j) > MSP), PM stops matching knowledge patterns in the pattern base. A knowledge pattern with at least one instance that has similarity probability greater than the threshold is defined as a suitable one. For example, the role list [A0 V X] for the query sentence in (Ex. 5.2) described in Section 5.2.3 is considered as a suitable one, even there is an unassigned argument in the query.

MSP	**P**	**R**	**F1**	**T (sec/sentence)**
30000	89.68	89.20	89.44	2.949
20000	89.68	89.20	89.44	2.947
10000	89.68	89.20	89.44	2.943
5000	89.68	89.20	89.44	2.916
2000	89.68	89.20	89.44	2.433
1000	89.67	89.17	89.42	2.433
900	89.62	89.08	89.35	2.379
800	89.64	89.10	89.37	2.293
700	89.64	89.10	89.37	2.207
600	89.63	89.09	89.36	2.076
500	89.62	89.08	89.35	1.974
400	89.62	89.07	89.35	1.835
300	89.64	89.12	89.38	1.707
200	89.67	89.18	89.42	1.500
100	89.71	89.39	89.55	1.235
90	89.71	89.39	89.55	1.196
80	89.70	89.38	89.54	1.159
70	89.72	89.39	89.56	1.125
60	89.70	89.37	89.54	1.082
50	89.73	89.39	89.56	1.035
40	89.73	89.40	89.56	0.981
30	89.74	89.39	89.57	0.921
20	89.84	89.54	89.69	0.858
10	89.78	89.34	89.56	0.788
8	89.65	89.22	89.43	0.746
6	89.64	89.16	89.40	0.698
4	89.49	88.96	89.22	0.641
2	89.13	88.58	88.86	0.609
1	89.00	88.32	88.66	0.591

Table 5.12. Results for different MSP values obtained by the complete system on WSJ 23, with known arguments.

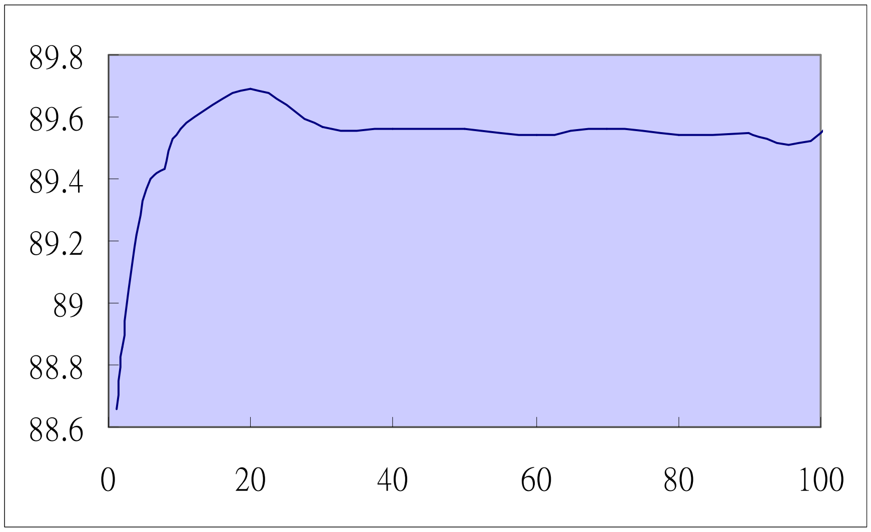


Figure 5.7. Results for MSP with values from 1 to 100.

The question is "how much is enough for suitable patterns". Table 5.12 shows different results for various values of Maximum-Suitable Pattern (MSP) and suggests that no improvement after 100 matches. Figure 5.7 shows the graphical curve with MSP values no greater than 100. Note that all these differences in Table 5.12 appear completely insignificant but the execution time increases a lot as the MSP value does. The key point will be no need to search all training data. What we need is just to find a MSP value by experiments. After applying MSP=20 in PM, the execution time for PM with speed improvement is less than 0.9 seconds per sentence. That is about 1.7 times faster than KNN.

Execution time with different datasets

The build-time of the pattern base on all training data (WSJ 02-21) is about 2.5 minutes. The training time ranges from about 10 hours to 60 hours depending on different systems shown in McCracken (2005). Figure 5.8 shows a graphical illustration with different sizes for the average classification time, seconds per sentence. The graph of execution for the system shows a log-like curve.

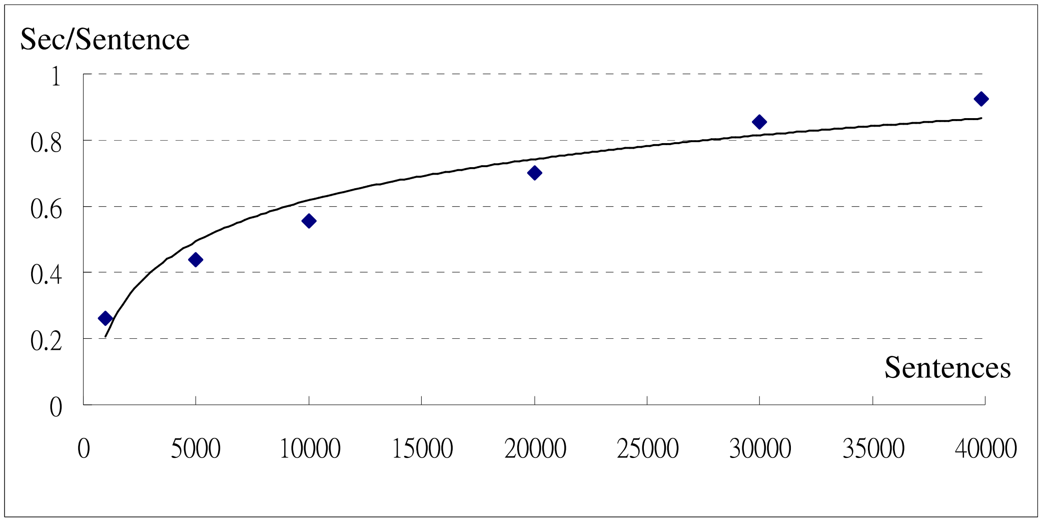


Figure 5.8. Curve of execution time for the complete system on WSJ 24, with known arguments.

System	T (sec/sen)
Palmer et al. (2005)	6.3
PARA+PM	0.9

Table 5.13. Execution time for argument identification and classification by different systems.

Table 5.13 shows the comparison of execution time for the system Palmer et al. (2005), and PARA+PM. This table suggests that the combination of PARA and PM is an efficient approach to solving the problems of argument identification and classification. It results in an overall system that offers benefits in terms of accuracy and speed, making it more effective for real time applications of SRL.

All execution time are calculated based on a P4 3.0 GHz CPU and 1G RAM Linux machine.

5.4.7 Comparison with other systems

Table 5.14 shows results for different systems using the same syntactic parses as input. Surdeanu et al. (2005), Tsai et al. (2005), Che et al. (2005), Moschitti et al. (2005) and Sang et al. (2005) are systems only using Charniak's parses listed in CoNLL 2005 shared task. The modified statistical classifier and PARA (*PARA+Palmer-Modified*) is the system described in Chapter 4.

System	***P***	***R***	***F_1***	***Features***
PARA+PM	78.04	75.20	76.60	**7**
Surdeanu	80.32	72.95	76.46	31
Tsai	82.77	70.90	76.38	25
Che	80.48	72.79	76.44	24
Moschitti	76.55	75.24	75.89	14
Sang	79.03	72.03	75.37	30
PARA+Palmer-Modified	71.18	70.90	73.49	7
Palmer et al. (2005)	68.60	57.80	62.74	6

Table 5.14. Results for different systems on WSJ 23 listed in the CoNLL 2005 shared task.

Even using fewer features, the combination of PARA and PM offers a faster and more accurate system for SRL compared to systems using the same input. It also becomes one of the top-performing systems in the CoNLL 2005 shared task compared to systems using more features and parses. It suggests that role dependencies between SRL help to yield accurate SRL.

5.5 Summary and Remarks

This chapter has shown that basic syntactic information is useful for semantic role labeling (SRL). The CoNLL 2005 shared task attracted a large amount of attention to SRL with different models. These systems have introduced more than 40 kinds of features with different combinations of syntactic parses. The best system achieved F1:79.44 on WSJ 23 with 7 syntactic parses and 18 features. The best system using only Charniak's syntactic parses provides F_1:76.46 performance on WSJ 23 (Surdeanu et al. 2005). All existing systems are based on Singular Argument Classification, which classifies an argument one by one without using other arguments. This often results in a problem of duplicate roles. Post-processing is used in systems to avoid wrong labels caused by duplicate assignment in a predicate-argument structure.

There exists a role relationship such as [A0 V A1] in predicate-argument structures, which is called role dependency. Such a relationship can be exploited by Multi-Argument Classification and Pattern-Matching to increase the performance of a typical statistical singular–argument classifier. The technique of Pattern-Matching, based on a pattern base, is a memory-based learner. In order to obtain a high precision, all basic features are essential to the system for matching a query pattern with a knowledge pattern from the pattern base.

Finding suitable knowledge patterns from the pattern base or the controlling strategy is implemented with a Maximum Suitable Pattern (MSP) constraint, which helps to reduce execution time to a log-like curve when using different sizes of training data. By this controller, the system is about 3.5 times faster than one without the controller.

The system architecture includes two stages, building and testing. In the building stage, only patterns extracted from the training examples are stored in the pattern base without any distribution calculation. All learning calculation is left to the testing stage, which includes the Predicate-Argument Recognition Algorithm (PARA) as a pre-processor to recognize all arguments for all predicates, role classifiers of Pattern-Matching, and the modified system (PML) from Palmer et al. (2005). This complete system outperforms existing singular argument systems from the CoNLL 2005 shared task using the same parses.

The results show that the preposition, phrase type and head word are the three features that influence the systems most. There is an F_1:13 difference when the system uses PARA and auto parses for the problems of argument identification and argument classification to recognize and label all arguments, compared to the results obtained given all arguments. The overall performance on WSJ 23 is about F_1:2.8 less than the best system (Punyakanok et al. 2005) in the CoNLL 2005 shared task. But both results, those from WSJ 23 and those from the Brown corpus, reveal this system can offer competitive performance for SRL even with fewer features.

This chapter has successfully demonstrated that the application of Multi-Argument Classification can not only increase the performance of an existing statistical system, such as Palmer et al. (2005), but also offer competitive performance compared to the best system in the CoNLL 2005 shared task and outperform systems with the same syntactic information. Results suggest that the combination of PARA and PM is an efficient approach to solving the problems of argument identification and classification. It results in an overall system that offers benefits in terms of accuracy and speed to the practitioner.

Chapter 6

Conclusions

This thesis addressed the problem of Semantic Role Labeling (SRL). Distinct from existing ML approaches that use Singular-Argument Classification for SRL, this thesis has described a combination of heuristic algorithms and Machine Learning (ML) techniques for Multi-Argument Classification that is faster and more accurate. Heuristic algorithms are rules constructed by humans, which offer some unlearned or general solutions to a problem. These heuristic and training-free algorithms can also provide systems with the added benefit of greater processing speed. Multi-Argument Classification is another technique that makes it possible to increase accuracy without using rich features. This technique utilises role dependencies, which are relationships in the predicate-argument structure. This kind of dependency or pattern structure can serve as useful information for argument classification.

This thesis demonstrates improved accuracy of argument classification by constructing a model of Multi-Argument Classification and Singular-Argument Classification to associate unlabelled semantic structure with semantic roles. Such unlabelled semantic structure can be identified via either a ML argument recogniser or a pre-processor to map syntactic structure to unlabelled semantic structure that is based on finding upper-most ancestor nodes in tree-based parses.

ML techniques, adopted when the heuristic algorithms do not perform well in particular cases, serve to predict which phrases of a sentence form arguments for predicates and which labels should be assigned to arguments in a sentence. The machine learning techniques studied in this thesis include:

1) an improved statistical approach based on Singular-Argument Classification

(SAC), as a base system (BS);

2) a new technique, Pattern-Matching (PM) for Multi-Argument Classification (MAC), which utilises role relationships in the semantic structures.

The new technique (PM) is based on the discovery of relationships between roles in a predicate-argument list or pattern.

The combination of heuristic algorithms with ML approaches (described in Chapters 4 and 5) constructs a faster and more accurate system. This combination offers competitive performance compared to the best system that uses rich syntactic information in the CoNLL 2005 shared task but outperforms systems with the same syntactic information.

Section 6.1 describes contributions of this thesis. Section 6.2 recapitulates all the techniques described in this research. Avenues for future work are also discussed in Section 6.3.

6.1 Contributions

This thesis has developed a framework for the problem of Semantic Role Labeling (SRL) in which syntactic structure (i.e. parse trees) can be automatically converted to semantic structure based on the application of learning and inference strategies. The proposed method offers benefits in term of accuracy and speed by using rule-based heuristic algorithms and machine learning approaches as the central mechanisms to solve the problem of SRL. The overall strategy presented in this thesis deconstructs SRL into two parts: argument identification and argument classification.

This thesis makes contributions in the form of new methodologies for:

1) argument recognition, namely the Predicate-Argument Recognition Algorithm (PARA); and

2) role classification, namely Pattern-Matching (PM) for Multi-Argument Classification;

Based on these algorithms the thesis makes several other contributions.

Contribution to the problem of argument identification:

- Empirical evidence that it is possible to design a deterministic algorithm for directly mapping syntactic parses to semantic arguments without any training.
- Empirical evidence that this algorithm outperforms existing systems in the CoNLL 2005 shared task using the same syntactic information.
- Empirical evidence that the proposed heuristic approach reduces execution time when compared to ML approaches.
- Evidence that this algorithm can be used as a pre-processor for argument identification to improve some existing ML role classifiers such as Priority Maximum Likelihood estimation.
- Evidence that it is possible to apply this algorithm in another corpus domain without decreasing performance significantly.

Contribution to the problem of argument classification:

- Empirical evidence showing that Multi-Argument Classification in SRL helps to boost the performance of argument classification compared to the statistical role classifier proposed by Palmer et al. (2005).
- Evidence that the proposed multi-argument technique with basic syntactic features, using the Predicate-Argument Recognition Algorithm (PARA) as a pre-processor, achieves better performance than existing systems that use rich syntactic features based on the same syntactic parses.

In general, building a competitive system involves issues related to (1) the type of architecture and model used for recognizing and labeling arguments; (2) the algorithm used to train the learning functions of the architecture; (3) the type of features used for representing the data; and (4) practical techniques and heuristics to develop and tune a learning system for a real-world natural language problem.

In view of these four points, this thesis presents a system that has been shown by evaluation to be amongst the top performing systems for SRL, and thus can be considered as a state-of-the-art system. For the problem of argument identification, it obtains results that make it the top-performing system among those that utilize the same syntactic information in the CoNLL 2005 shared task. On the problem of argument classification with known arguments, the system is one of the top-performing systems based on basic features. For the combination of argument identification and argument classification in the Semantic Role Labeling task, this system is also the best one in the CoNLL 2005 shared task when considering systems based on the same syntactic information.

The evaluations detailed in Chapters 3, 4, and 5 show that the system achieves F_1: 76.59 and 66.25 on the WSJ23 and partial Brown corpus test datasets from the CoNLL-2005 shared task. These results are better than those obtained by other systems used in the CoNLL 2005 shared task (Carreras and Marquez, 2005), that use the same syntactic information. The pre-processing algorithm, namely the Predicate-Argument Recognition Algorithm (PARA) (described in Chapter 4), is also better than the best argument recognition system (Moschitti et al., 2005) using the same syntactic information because it achieves F_1: 82.60 on this task. The execution time for the Multi-Argument Classification system is about 0.9 seconds, which is about 7 times faster than that of the re-implemented system from Palmer et al. (2005) (about 6.3 sec/sentence in Section 3.6.7) on a P4 3.0 G and 1G Ram Linux machine.

In summary, the results presented in this thesis show that the proposed system, combining rule-based heuristic algorithms and machine learning techniques, can successfully implement a direct mapping from syntactic parses to unlabelled semantic arguments and allows the application of Multi-Argument Classification via a pattern base in SRL. These fulfill the claims made in the Thesis Statement (Section 1.2). The criteria of accuracy and speed are both met by the proposed system.

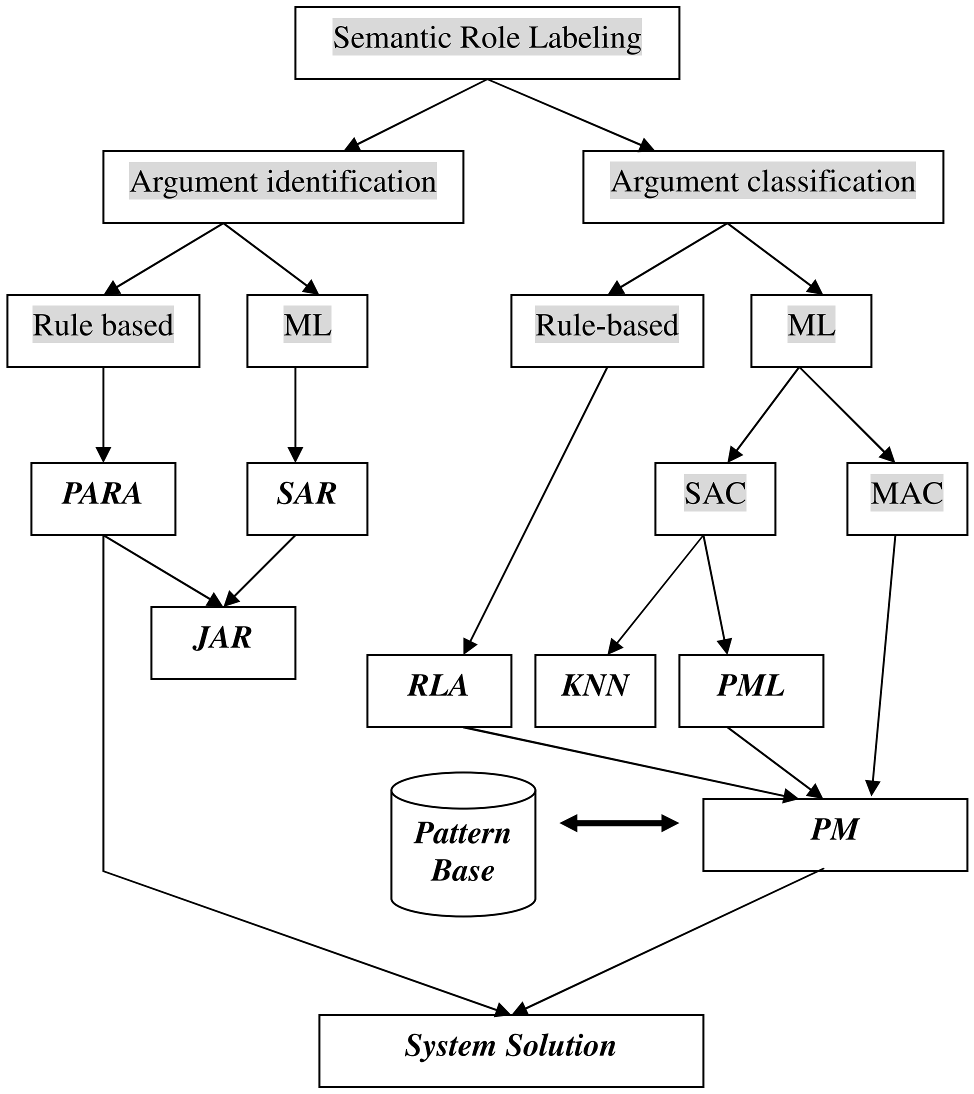


Figure 6.1. Relationships between techniques and categories of techniques (with a gray background) described in the thesis.

6.2 Review of Techniques

The fundamental techniques employed in this thesis and how they relate to Semantic Role Labeling (SRL) are illustrated in Figure 6.1.

These techniques consisted of:

- a modified statistical argument recognizer, (**SAR**), improving the performance of the existing statistical argument recognizer proposed by Palmer et al. (2005);
- a training-free argument recognizer or Predicate-Argument Recognition Algorithm (**PARA***)*, which implements a direct mapping algorithm from syntactic parses to unlabelled semantic arguments based on tree parses;
- a joint argument recognizer (**JAR**), which is the combination of PARA and SAR, and is used to compare the performance between the deterministic algorithm (PARA) and the Machine Learning (ML) approach (SAR);
- a basic role labeller (**RLA**), a modified version of the one used in the shared task (Carreras and Marquez, 2005), which is a basic role labeller for the task of argument classification;
- two singular role classifiers, the Kth-Nearest Neighbour algorithm (**KNN**) and an improved version of the statistical role classifier (**PML**) from Palmer et al. (2005), which are the ML approaches for argument classification used as fundamental classifiers for the system;
- a multi-argument classifier, Pattern-matching (**PM**), which uses Multi-Argument Classification to utilize role dependencies to boost the overall performance for different role classifiers;
- a **Pattern base**, which serves as a knowledge base with training information for the classifiers; and
- the final system solution for the problem of Semantic Role Labelling, which includes PARA as a pre-processor for argument identification, and the multi-argument classifier, PM, for argument classification for the problem.

Chapter 2 described the linguistic background for semantic role assignment, including domain-independent semantic role assignment, and domain-specific semantic role assignment. The CoNLL 2005 shared task, which belongs into the realm of domain-independent semantic role assignment, has attracted much attention and research efforts, in which new ML techniques, features, and combination of systems have been investigated and developed (Carreras and Marquez, 2005). Palmer et al.'s (2005) system is based on the one proposed by Gildea and Jurafsky (2002) and implements a statistical approach using basic features for shallow Semantic Role Labeling based on PropBank annotations (similar to the task in the CoNLL 2005 shared task). Chapter 2 explored the existing system proposed by Palmer et al. (2005), which is used as a baseline for the work presented in this thesis because of its simplicity.

Chapter 3 introduces improvements to the existing statistical system (Palmer et al., 2005). The system consists of the modified statistical argument recogniser (**SAR**), the modified statistical role classifier (**PML**) and other improvements such as using additional features, heuristics, pre-processing and post-processing methods. In addition, Chapter 3 also discussed the data and the evaluation script used to evaluate the system.

In Chapter 3, it was found that, although this modified system exhibits better performance than that of Palmer et al. (2005), there is still room for improvement when comparing the performance of this system to that of the one proposed by Surdeanu and Turmo (2005). Their system uses rich features and single syntactic information. Possible areas for improvement include, but are not limited to, better argument recognizers and different classification techniques. The results of this comparison encourage further research on the problem of Predicate-Argument Recognition, leading to the method presented in Chapter 4, and the problem of argument classification via Multi-Argument Classification, as presented in Chapter 5.

Chapter 4 focused on the problem of Predicate-Argument Recognition (or argument identification) and discussed a Predicate-Argument Recognition Algorithm (**PARA**). This algorithm (**PARA**) implements a direct mapping from syntactic parses to unlabelled semantic arguments without any need for training. Utilizing the output from a state-of-the-art parser like the one proposed by Charniak (2000), a number of the issues arising from complex parses (for example, whether a phrase is under or independent of the previous phrase) have been solved. This makes it possible to employ an algorithm that maps directly between syntactic parses and unlabelled semantic arguments.

In Chapter 5, a basic Role Labeling Algorithm (**RLA**), modified from the rule-based algorithm used in the CoNLL shared tasks, was proposed as a basic rule-based method for the task of Role Labeling. There are two generic ML strategies in argument classification, namely Singular-Argument Classification (SAC) and Multi-Argument Classification (MAC). All existing ML role classifiers in the CoNLL shared task implement SAC.

In general, solutions for accuracy improvement add more features to a system but that in turn impacts on running time. An alternative is to utilise role-dependencies without additional features by using a multi-argument classifier. The main technique for MAC presented in this thesis is called Pattern-Matching (**PM**). PM exploits characteristics present in the data due to inherent role dependency. Compared to two role classifiers (**KNN** and **PML**), **PM** achieves better performance by leveraging the role dependencies when classifying, as shown in Chapter 5.

In order to implement MAC, a **Pattern base** is built from training data. Additionally, in order to avoid increasing execution time when testing the system, the controlling strategy used by **PM** employs constraints based on maximum suitable patterns. This constraint reduces time complexity to a log-like curve and achieves better performance when using **PM** for unlabelled arguments.

In summary, a training-free tree-based algorithm, Predicate-Argument Recognition Algorithm (PARA), provides better results for argument identification compared to existing argument recognizers and can serve as a pre-processor for a role labeling system. The multi-argument classifier **PM** for argument classification presented in this thesis improves on the singular-argument classifiers **PML** and **KNN**. The best solution for converting syntactic structure to semantic structure uses the rule-based argument recognizer **PARA** in conjunction with **PM**. This combination is the top performing system, outperforming other systems using the same syntactic information in the CoNLL 2005 shared task.

6.3 Future work

Traditional techniques have contributed significantly toward the solution of SRL (Carreras and Marquez, 2005). These techniques label each argument one by one, a process that is called Singular-Argument Classification in this thesis. The successful demonstration of a Multi-Argument Classification system in this thesis opens an alternative pathway for solving SRL problems in addition to the traditional use of singular-argument classification.

The fundamental concepts and construction techniques for (a) the direct mapping from syntactic parses to unlabelled semantic arguments, and (b) Multi-Argument Classification, already achieve good performance when compared to existing techniques using the same syntactic information. However, there remain avenues for future exploration and these are discussed below.

SRL can be solved in two main steps: argument identification and argument classification. In this research, the PARA method for argument identification utilises techniques from state-of-the-art parsers (like Charniak, 2000) and outperforms existing ML techniques based on the same syntactic information. However, there are some potential areas for future research.

- More syntactic information

  Although the system presented in this research outperforms existing systems based on the same syntactic information in the CoNLL 2005 shared task, the performance obtained by this system does not reach that of the four top performing systems (Punyakanok et al., 2005; Haghighi et al., 2005; Marquez et al., 2005; Pradhan et al., 2005), which use various parses as rich syntactic information. Thus further gains in performance might be achieved if this system was enabled to exploit rich syntactic information.

This research proposes a preliminary structure based on basic features for demonstrating the possibility of Multi-Argument Classification in the problem of SRL. There are about 40 features that have been introduced in different systems in the CoNLL 2005 shared task. It would be worthwhile to utilize these additional features for Multi-Argument Classification by adding more features to the system such as the **NE** feature, which indicates if the target argument is, embeds, overlaps or is embedded in a named-entity along with its type (Punyakanok et al. 2005).

- Better algorithm to join PARA with ML techniques
  Chapter 4 introduces a simple joint algorithm combining PARA and a traditional statistical argument recognizer (SRB). However, it does not seem to offer better arguments for the role classifiers in the overall performance of SRL. A more sophisticated joining procedure or application of other argument recognizers may be able to improve on this.

- Different language domains
  This research is confined to the English domain for argument identification and argument classification. It would be interesting to apply PARA and Multi-Argument Classification to different languages (such as German, Chinese or Japanese) if possible.

- Other resources
  The CoNLL 2005 shared task offered a useful official resource: PropBank Frames[1], which are rolesets for English verbs. The PropBank Frame dataset defines related

[1] http://www.lsi.upc.edu/~srlconll/resources/pb-frames.tar.gz

role descriptions or rolesets for each predicate or verb. There are many other resources related to SRL such as the free Lexical Conceptual Structure (LCS[2]) knowledge database for semantic roles, the WordNet[3] database, and the verb-structure database VerbNet[4]. It would be worthwhile trying to add these resources to the system as additional features or knowledge bases.

- Other Machine-Learning techniques:
  Similarity- or probability- based techniques play an important role in SRL. There are different kinds of machine-learning based techniques. This thesis has demonstrated the application of two basic methods, Kth-Nearest Neighbour (KNN) and Priority Maximum Likelihood estimation (PML). Other probability-based methods such as Maximum Entropy (ME) might produce a better outcome when employed with Pattern-Matching.

- Different corpora and language domains:
  The results of the joint system consisting of PM and PML do not show consistency in performance when moving from WSJ to the Brown domain. This weakness, caused by propagating errors in Natural Language Processing applications and different domains, could be another research topic for researchers who are interested in SRL (like how to minimize the impact of different domains).

It has been shown that the system proposed in this thesis outperforms existing systems based on the same syntactic information in the CoNLL 2005 shared task. Distinct from traditional ML approaches to argument identification and existing singular role classifiers for argument classification, this thesis successfully demonstrates (a) an efficient approach to solving the problem of argument identification by using heuristic algorithms based on state-of–the-art tree parses, and (b) a method for solving the problem of argument classification with Multi-Argument Classification techniques. This results in an overall system that offers benefits in terms of accuracy and speed to

[2] http://www.umiacs.umd.edu/~bonnie/LCS_Database_Documentation.html
[3] http://wordnet.princeton.edu/
[4] http://www.cis.upenn.edu/~mpalmer/project_pages/VerbNet.htm

the practitioner.

References

Baker, C. F., Fillmore, C. J. and Lowe, J. B. (1998). The Berkeley FrameNet project. In *Proceedings of COLING/ACL*, Montreal, Canada, pages 86-90.

Baldewein, U, Erk, K, Padó, S. and Prescher, D. (2004). Semantic role labelling with similarity-based generalization using EM-based clustering In *Proceedings of Senseval-3* pp. 64-68

Baldewein, B, Erk, K, Padó, S. and Prescher, D. (2004). Semantic Role Labeling With Chunk Sequences. In *Proceeding of CoNLL'2004 Shared Task.*

Bosch, A. V. D., Canisius, S., Daelemans, W., and Sang, E. T. K. (2004). Memory-based semantic role labeling: Optimizing features, algorithm and output. In *Proceeding of CoNLL'2004 Shared Task.*

Carreras, X. and Marquez, L. (2003). Phrase recognition by filtering and ranking with perceptrons. In *Proceedings of RANLP-2003,* Borovets, Bulgaria.

Carreras, X and Màrquez, L. (2004). Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling. In *Proceeding of CoNLL'2004 Shared Task.*

Carreras, X., Màrquez, L. and Chrupała, G. (2004). Hierarchical Recognition of Propositional Arguments with Perceptrons. In *Proceeding of CoNLL'2004 Shared Task.*

Carreras, X. (2005). Learning and Inference in Phrase Recognition: A Filtering-Ranking Architecture using Perceptron. *Doctoral thesis in Universitat Politècnica de Catalunya (UPC).*

Carreras, X. and Marquez, L. (2005). Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *Proceedings of CoNLL-2005.*

Charniak, E. (2000). A Maximum-Entropy-Inspired Parser. In *Proceedings of NAACL-2000.*

Che, W., Liu, T., Li, S., Hu, Y., and Liu, H. (2005). Semantic role labeling system using maximum entropy classifier. In *Proceeding of CoNLL'2005 Shared Task.*

Chieu, H. L. and NG, H. T. (2003). Named Entity Recognition with a Maximum Entropy approach. In *Proceedings of CoNLL 2003,* Edmonton, Canada.

Cohn, T., and Blunsom, P. (2005). Semantic role labeling with tree conditional random fields. In *Proceeding of CoNLL'2005 Shared Task.*

Collins, M. (1999). Head-Driven Statistical Models for Natural Language Parsing. *PhD Dissertation, University of Pennsylvania*.

Dowty, D. R. (1991). Thematic proto-roles and argument selection. *Language,* Vol. 67, No. 3 (Sep., 1991), pp. 547-619.

Fillmore, C. J. (1968). The case for case. In Emmon W. Bach and Robert T. Harms, editors, *Universals in Linguistic Theory*. Holt, Rinehart & Winston, New York, pages 1-88.

Fillmore, C. J. and Baker. C. F. (2000). FrameNet: Frame semantics meets the corpus. Poster presentation, *74th Annual Meeting of the Linguistics Society of America*.

Gildea, D. and Jurafsky, D. (2002). Automatic Labeling of Semantic Roles. *Computational Linguistics*, 28(3):245-288.

Gildea, D. and Hockenmaier, J. (2003). Identifying Semantic Roles Using Combinatory Categorial Grammar . In *Proceedings of EMNLP-2003*, Sapporo, Japan.

Gildea, D. and Palmer, M. (2002). The Necessity of Parsing for Predicate Argument Recognition . In *Proceedings of ACL 2002*, Philadelphia, USA.

Gimenez, J. and Marquez, L. (2003). Fast and accurate part-of-speech tagging: The SVM approach revisited. In *Proceedings of RANLP-2003*, Borovets, Bulgaria.

Haegeman, L. (1991). Introduction to Government and Binding Theory. *Oxford:Blackwell*, 1991.

Haghighi, A., Toutanova, K. and Manning, C. (2005). A Joint Model for Semantic Role Labeling. In *Proceedings of CoNLL 2005 Shared Task.*

Higgins, D. (2004). A transformation-based approach to argument labeling. In *Proceeding of CoNLL'2004 Shared Task.*

Jackendoff, R. (1972). Semantic Interpretation in Generative Grammar. *MIT Press*, Cambridge, Massachusetts.

Jaynes, E. T. (1957). Information Theory and Statistical Mechanics. II. *The Physcal Review* Vol. 108, No. 2, p. 171-190, 1957.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proceedings: Workshop on Pattern Recognition in Practice*, Amsterdam, North Holland, pages 381-397.

Jurafsky, D and Martin J.H. (2000). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (contributing writers: Andrew Kehler, Keith Vander Linden, and Nigel Ward). Prentice-Hall, New Jersey.

Kouchnir, B. (2004). A Memory-Based Approach for Semantic Role Labeling. In *Proceeding of CoNLL'2004 Shared Task.*

Lin, C.S. A. and Smith, T. C. (2005). Semantic role labeling via Consensus in Pattern-matching. In *Proceedings of CoNLL-2005.*

Lin, C.S. A. and Smith, T. C. (2006a). A Tree-based Algorithm for Predicate-Argument Recognition. In *Bulletin of Association for Computing Machinery New Zealand (ACM_NZ), volumn 2, issue 1.*

Lin, C.S. A. and Smith, T. C. (2006b). Semantic Role Labeling via Instance-Based Learning. In *Proceeding of Conference on Empirical Methods in Natural Language Processing (EMNLP) '2006,* pp. 180-188, Sydney, Australia.

Liu, R-L. and Soo, V-W. (1993). An Empirical Study on Thematic Knowledge Acquisitin based on Syntactic Clues and Heuristics. In *Processing of 31st Annual Meeting of the ACL*, Columbus, Ohio, 1993, 243-250.

McCracken, (2005). On the Amount of Training Data in SRL. A talk at the CoNLL 2005 shared task session, http://www.lsi.upc.edu/~srlconll/st05/slides/mccracken.pdf.

Marcus, M.P., Santorini, B. and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* 19(2):313-330.

Marquez, L., Comas, P., Gimenez J., and Catala, N. (2005). Semantic role labeling as sequential tagging. In *Proceeding of CoNLL'2005 Shared Task.*

Mitsumori, T. Murata, M. Fukuda, Y. Doi, K. and Doi, H. (2005). Semantic role Labeling using Support Vector Machines. In *Proceedings of CoNLL 2005 Shared Task.*

Moschitti, A. and Bejan, C. A. (2004). A Semantic Kernel for Predicate Argument Classification . In *proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, Boston, MA, USA

Moschitti, A., Giuglea, A.-M., Coppola, B., and Basili, R. (2005). Semantic role labeling using support vector machines. In *Proceeding of CoNLL'2005 Shared Task.*

Ozgencil N. E., and McCracken, N. (2005). Semantic role labeling using libSVM. In *Proceeding of CoNLL'2005 Shared Task.*

Palmer, M., Gildea, D., abd Kingsbury, P., (2005). The Propostin Bank: An Annotated Corpus of Semantic Roles. In *Proceedings of ACL:* Volume 31, Number 1. p72-105.

Park, K.-M., Hwang, Y.-S. and Rim, H.-C. (2004). Two-Phase Semantic Role Labeling based on Support Vector Machines. In *Proceeding of CoNLL'2004 Shared Task.*

Park, K.-M. and Rim, H.-C. (2005). Maximum entropy based semantic role labeling. In *Proceeding of CoNLL'2005 Shared Task.*

Ponzetto, S. P., and Strube, M. (2005). Semantic role labeling using lexical statistical information. In *Proceeding of CoNLL'2005 Shared Task.*

Pradhan, S., Hacioglu, K, Ward, W., Martin, J. H., and Jurafsky, D. (2003). Semantic Role Parsing: Adding Semantic Structure to Unstructured Text . In *Proceedings of the International Conference on Data Mining (ICDM-2003)*, Melbourne, USA.

Pradhan, S., Ward, W., Hacioglu, K., Martin, J. H., Jurafsky, D. (2004). Shallow Semantic Parsing using Support Vector Machines, in *Proceedings of the Human Language Technology Conference/North American chapter of the Association for Computational Linguistics annual meeting (HLT/NAACL-2004)*, Boston, MA.

Pradhan, S., Hacioglu, K, Ward, W., Martin, J. H., and Jurafsky, D. (2005). Semantic Role Chunking Combining Complementary Syntactic Views. In *Proceeding of CoNLL'2005 Shared Task.*

Punyakanok, V., Roth, D., Yih, W., and Zimak, D. (2004). Semantic Role Labeling via Integer Linear Programming Inference . In *Proceedings of. the International Conference on Computational Linguistics (COLING),2004.*

Punyakanok, V., Koomen, P., Roth, D., and Yih, W. T. (2005). Generalized inference with multiple semantic role labeling systems. In *Proceeding of CoNLL'2005 Shared Task..*

Rosa, J. L. G. and Francozo, E. (1999). Hybrid Thematic Role Processor: Symbolic Linguistic Relations Revised by Connectionist Learning. In *Proceedings of the 16th International Joint Confrerence on Artificial Intelligence*-IJCAI'99 Stockholm, Sweden, July 31-Argust 6, 1999, Volume 2, 852-857.

Sang, E. T. K., Canisisus, S., Bosch, A. V. D., and Bogers, T. (2005). Applying spelling error correction techniques for improving semantic role labeling. In *Proceeding of CoNLL'2005 Shared Task.*

Surdeanu, M., Harabagiu, S., Williams, J., and Aarseth, P. (2003). Using Predicate-Argument Structures for Information Extraction. In *Proceedings of ACL 2003*, Sapporo, Japan, page 8-15.

Surdeanu, M. and Turmo, J. (2005). Semantic role labeling using complete syntactic analysis. In *Proceeding of CoNLL'2005 Shared Task.*

Sutton C. and McCallum A. (2005). Joint Parsing and Semantic Role Labeling. In *Proceeding of CoNLL'2005 Shared Task.*

Tsai, T.-H., Wu, C.-W., Lin, Y.-C. and Hsu, W.-L. (2005). Exploiting Full Parsing Information to Label Semantic Roles Using an Ensemble of ME and SVM via Integer Linear Programming. In *Proceeding of CoNLL'2005 Shared Task.*

Xue, N. and Palmer, M. (2004). Calibrating features for semantic role labeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Yi, S. and Palmer, M. (2005). The integration of syntactic parsing and semantic role labeling. In *Proceedings of CoNLL'2005 Shared Task.*