

Using Paraphrases for Information Extraction

by

Shilpa Arora

BE in Computer Engineering
Nanyang Technological University, 2000-04

**SUBMITTED TO THE SMA OFFICE IN PARTIAL
FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF**

MASTER OF SCIENCE IN COMPUTER SCIENCE
AT THE
SINGAPORE-MIT ALLIANCE
July 2005

© 2005 National University of Singapore. All rights reserved.

Signature of author: _____

Shilpa Arora
June 30th, 2005

Certified & Accepted by: _____

Assoc. Prof. Ng Hwee Tou
SMA CS Programme Co-Chair, NUS
Dissertation Supervisor

Certified by: _____

Prof. Leslie Kaelbling
SMA CS fellow, MIT

Accepted by: _____

Prof. Charles Leiserson
SMA CS Programme Co-Chair, MIT

Acknowledgements

Firstly, I would like to thank Prof. Ng Hwee Tou and Prof. Leslie Kaelbling for their supervision in this project. I am grateful to them to have introduced me to the field of Natural Language Processing.

Prof. Ng Hwee Tou has always encouraged me to come up with my own ideas and approach towards solving a problem and he always appreciated and supported my interest and enthusiasm towards the project.

I would also like to take this opportunity to thank my colleague in SMA, Mr. Chieu Hai Leong, who helped me understand his Information Extraction system which I have used as a reference in this project.

Also, I would like to thank my colleague Mr. Low Jin Kiat for providing me with his software to use in the project.

Table of Contents

Acknowledgements.....	2
Table of Contents	3
List of Figures.....	4
Abstract	5
CHAPTER 1 INTRODUCTION	6
1.1 Objective	6
1.2 Background	6
1.3 Organization of the report	8
CHAPTER 2 LITERATURE REVIEW	9
2.1 Applications of paraphrases	9
2.2 Techniques for Paraphrase Acquisition	10
2.2.1 Selection of Corpus	10
2.2.2 Acquisition of Paraphrases.....	11
CHAPTER 3 SYSTEM ARCHITECTURE AND DESIGN	13
3.1 Introduction to ALICE	13
3.2 Architecture Design and flow	15
3.3 Module Description.....	20
3.3.1 Pre-processing Modules	20
3.3.2 Feature Extractor	22
3.3.3 Template Reader	25
3.3.4 Maxent Classifier	25
3.3.5 Template Generator.....	26
3.3.6 Maximum Entropy training and prediction model.....	27
CHAPTER 4 IMPLEMENTATION DETAILS	28
4.1 External software used	28
4.2 Implementation of String Slots	29
4.3 Using weakly labeled data	36
CHAPTER 5 RESULTS AND DISCUSSION	38
5.1 Scorer used.....	38
5.2 Tests conducted.....	39
CHAPTER 6 CONCLUSION AND FUTURE WORK.....	48
6.1 Conclusion	48
6.2 Possible Improvements	48
CHAPTER 7 REFERNCES	51
Appendix A.....	54

List of Figures

Figure 1: Alice black box architecture	13
Figure 2: ALICE - Sample Template	14
Figure 3: ALICE - the classifier model.....	17
Figure 4: ALICE - Training Module.....	18
Figure 5: Weakly labeled data Model	20
Figure 6: Average F-measure Vs No. of Training documents TST3.....	40
Figure 7: Average F-measure Vs No. of Training documents for TST4	41
Figure 8: Performance for TST3 files with MLD & WLD.....	42
Figure 9: Performance for TST4 files with MLD & WLD.....	43

Using Paraphrases for Information Extraction

By

Shilpa Arora

Submitted to the SMA office on June 30, 2005
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Science

ABSTRACT

Maximum Entropy has been used in several statistical classification problems in Natural Language Processing (NLP). This project uses the Maximum Entropy approach to build an Information Extraction (IE) system for extracting information about events such as terrorist events. Here, the Information Extraction (IE) task is modeled as a classification problem where information extracted from the document is used to fill information slots in a template. Manually generated templates are used as the training set for this system, but the amount of manually labeled data that can be provided is very limited. So, in our approach we use weakly labeled data, which essentially contains news articles describing same event, and are thus a rich source of paraphrases. The weakly labeled data is freely available and hence provides ample training examples. A basic Maximum Entropy based IE system was developed and improvement in performance using weakly labeled data was investigated.

Keywords:

Maximum Entropy Approach, Information Extraction, weakly labeled data

Dissertation Supervisors:

1. Dr. Ng Hwee Tou, SMA CS Co-Chair
2. Dr. Leslie Pack Kaelbling, SMA Fellow, MIT

CHAPTER 1 INTRODUCTION

1.1 Objective

This project aims at building an information extraction system using maximum entropy techniques. The information extracted from an article is used to fill in a template which represents the important information contained in the article.

The project also aims at using paraphrasing techniques to enhance the performance of the information extraction system. The source of paraphrases are the weakly labeled data, which is a set of documents describing same event as the manually labeled data used in the system. Documents describing same event would contain similar information hence would provide a rich source of paraphrases.

This project concentrates on building a classifier for the string slots¹ of a template and exploring the possibility of using weakly labeled data to improve the performance.

1.2 Background

Natural Language Processing has many applications based on statistical classification where diverse pieces of contextual evidence are combined in order to estimate the probability of a certain linguistic class occurring with them [7]. Maximum entropy is one of the approaches used for predicting linguistic classes using linguistic contexts.

¹ String slots are the slots of the template which are filled with a string taken from the article

The principle of Maximum Entropy Modeling as described by Berger et. al. [3] is simple:-
“*model all that is known and assume nothing about that which is unknown*”. In other words, the model chosen is such that it is consistent with what is known and for the unknown it is uniform.

Information Extraction (IE) task can also be modeled as a classification problem. In an IE system, information extracted from the document is used to fill information slots in a template called template slots. Thus, IE task can be modeled as a classification problem in which a classifier is built for every slot in the template.

The training data used in this system are the manually generated templates. Each training document has one or more manually generated templates associated with it. However, the amount of manually labeled data that can be provided is very limited.

For terrorist events, various news articles describing the same event would contain similar information. Hence, we can retrieve these documents and use them as something called the weakly labeled documents for training. These documents are called weakly labeled because the human generated template cannot be directly associated with it. But since it describes the same event as the manually labeled document, some of the information will be preserved.

With weakly labeled data, we have a richer source of training data set which will help us improve the system performance.

1.3 Organization of the report

In chapter 2, a literature survey on application of paraphrases and methods for acquiring paraphrases is presented. In chapter 3, we discuss the design approach used for this system and describe in detail the different modules in the architecture and the roles they play. Chapter 4 highlights the implementation details of the project, the external software used, difference phases of the system and algorithms for key modules. Results and discussion on performance and observations is presented in chapter 5. The last chapter provides a brief conclusion and a list of further improvements that can be made in the system.

CHAPTER 2 LITERATURE REVIEW

In this project, the literature review included an investigation of several applications of paraphrasing and how paraphrases are obtained.

2.1 Applications of paraphrases

Paraphrases i.e. different ways of expressing the same information has found many useful and intuitive applications in Natural Language Processing (NLP). This diversity in expression in natural language presents many challenges in NLP but it has also found many disparate applications.

Paraphrasing can be used to identify sentences that convey same information and hence can be used in Multiple Document Summarization to avoid repetition of information while summarizing. Paraphrasing can also be used in text generation, to create varied text.

Molla et al. [9] present a very unique work on exploiting use of paraphrasing in a question answering system. They present a question answering systems for technical domain which makes an intelligent use of paraphrases to increase the likelihood of finding the answer to the user's questions.

Another paraphrasing technique by Shinyama et al. [10] generalizes the paraphrases obtained as templates such as “*NUMBER* people die in *LOCATION*” where *NUMBER* and *LOCATION* are the slots that are filled with different values depending on the event. These generalized templates can be used in Information Extraction systems.

2.2 Techniques for Paraphrase Acquisition

The important phases of paraphrase acquisition can be classified as:

- 1. Selection of Corpus:** The first and foremost thing in Paraphrase acquisition or any other NLP application is selecting a corpus to work with. There are a number of corpus available online meant for several NLP applications.
- 2. Pre-processing:** Before beginning the paraphrase acquisition, Article/document level matching is necessary. It is important to identify the articles which convey same information and are hence the sources of paraphrases.
- 3. Sentence Alignment:** After selecting the correct article, we need to align the sentences which are probable sources of paraphrases and which will be processed to acquire paraphrases.
- 4. Extraction of Paraphrases:** Extraction of paraphrases involves identification of the part of sentences which convey same information and hence qualify as paraphrases.
- 5. Generalization:** The paraphrases acquired are generalized to templates which can then be used in a number of applications.

2.2.1 Selection of Corpus

For paraphrasing, selection of corpus is important. Paraphrases are expressions which although different, convey the same meaning and can be replaced for one another. Hence, the corpus needed for acquisition of paraphrases should contain documents that talk about the same thing i.e. where same information is expressed in more than one way.

Shinyama et al. in their work on paraphrases used Japanese news articles for a specific domain from two different news papers. The basic ideas is to search news articles from the

same day focusing on the fact that various newspapers describe a single event in different ways and these articles can be used as the source of paraphrases. Similarly, Barzilay et al. [1] in their unsupervised learning method used a collection of articles describing same event from two different news companies. Although these documents might be very different on surface, by nature of data, they are essentially descriptions of the same information.

Another example of data with similarity in nature would be multiple parallel English translations of novels as was used by McKeown et al. [2] in their work. Different translations may convey the meaning differently but the meaning of the original source is preserved through them.

2.2.2 Acquisition of Paraphrases

In the beginning, paraphrases were limited to lexical (single word) paraphrases. But with new approaches, we are now able to identify multi-word, syntactic and phrasal paraphrases etc. For example, McKeown et al., in their work, identified phrase level paraphrases in parallel translations of a single document. They used text alignment in which similarity between contexts surrounding two phrases was used as a measure to identify two phrases as paraphrases. Certain good paraphrase predicting contexts were identified and used further to extract paraphrase patterns from the corpus.

Shinyama et. al. used different news articles describing same event as a source for paraphrases. News articles dated near by are likely to provide articles describing same event. They used topic detection and tracking technique to find these comparable articles.

A threshold model was used for the time component to exploit the temporal relationship between the articles.

Barzilay et al, in their work in Multiple Sequence Alignment for paraphrasing, used clustering to group similar sentences and extract patterns from these groups. They used hierarchical complete-link clustering with a similarity metric based on word n-gram overlap (n=1, 2, 3, 4).

.

CHAPTER 3 SYSTEM ARCHITECTURE AND DESIGN

3.1 Introduction to ALICE

The diagram below shows the basic structure of ALICE (Automated Learning-based Information Content Extraction), the Information Extraction system being implemented in this project. The system is built on lines of the system developed by Chieu et. al. [5]. The project aims at developing a basic ALICE system and exploring the possibility of using weakly labeled data to enhance the system performance.

A high level design of the system is shown in Figure 1. The input to the ALICE system is a set of text documents describing terrorist events and the output is a set of templates filled with information extracted from the documents.

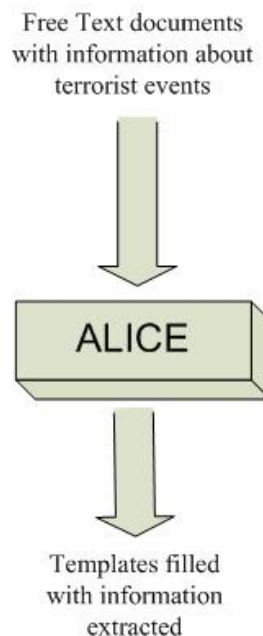


Figure 1: Alice black box architecture

An example of a template in ALICE is shown in figure 2.

0. MESSAGE: ID	DEV-MUC3-0001 (NCCOSC)
1. MESSAGE: TEMPLATE	2
2. INCIDENT: DATE	03 JAN 90
3. INCIDENT: LOCATION	EL SALVADOR: APOPA (CITY)
4. INCIDENT: TYPE	ATTACK
5. INCIDENT: STAGE OF EXECUTION	ACCOMPLISHED
6. INCIDENT: INSTRUMENT ID	“MORTAR” ; “RIFLE”
7. INCIDENT: INSTRUMENT TYPE	MORTAR: “MORTAR”; RIFLE: “RIFLE”
8. PERP: INCIDENT CATEGORY	TERRORIST ACT
9. PERP: INDIVIDUAL ID	? “TERRORIST”
10. PERP: ORG ID	-
11. PERP: ORG CONFIDENCE	-
12. PHYS TGT: ID	“LAS CANAS BRIDGE”
13. PHYS TGT: TYPE	TRANSPORTATION ROUTE: “LAS CANAS BRIDGE”
14. PHYS TGT: NUMBER	1: “LAS CANAS BRIDGE”
15. PHYS TGT: FOREIGN NATION	-
16. PHYS TGT: EFFECT OF INCIDENT	-
17. PHYS TGT: TOTAL NUMBER	-
18. HUM TGT: NAME	-
19. HUM TGT: DESCRIPTION	“NATIONAL GUARD UNITS”
20. HUM TGT: TYPE	ACTIVE MILITARY: “NATIONAL GUARD UNITS”
21. HUM TGT: NUMBER	PLURAL: “NATIONAL GUARD UNITS”
22. HUM TGT: FOREIGN NATION	-
23. HUM TGT: EFFECT OF INCIDENT	NO INJURY OR DEATH: “NATIONAL GUARD UNITS”
24. HUM TGT: TOTAL NUMBER	-

Figure 2: ALICE - Sample Template

The slots in the template can be divided into following groups:

- **String fill slots:** These slots are filled using strings extracted directly from the text document (slot 6, 9, 10, 12, 18, 19).
- **Set fill slots:** This requires classification of a slot fill into one of predetermined classes – inferred from the document. E.g. effect: *dead, injured*, etc.
- **Text conversion slots:** These slots have to be inferred from strings in the document (slot 2, 14, 17, 21, 24). E.g. Date slot: e.g. *yesterday* to absolute dates and Number slot: e.g. *twenty-three policemen* to *23*

3.2 Architecture Design and flow

In figure 1, the ALICE system is shown as a black box. A more transparent view of ALICE system is shown in figure 3. The free text documents first pass through a pre-processing module which performs certain operation on the raw text document so that the ALICE system can work on it. This preprocessing module consists of several parts: sentence boundary detector, sentence tag appending, parser, tokenizer, named-entity recognizer. These modules are described in detail in the next sub-section.

The outcome of the pre-processing module is a document containing one parse-tree per sentence and a document containing named entity tags for the data. For each parse tree, a list of noun phrases is generated and for each noun phrase, feature-extraction module extracts a set of features such as the verb of that noun phrase or the prepositional phrase it is a part of. This module is explained in detail in the next sub section. The trained model in ALICE together with this set of features is used by the maximum entropy classifier to classify the features as positive or negative filler for the slot with a probability where positive filler means the noun phrase can be used to fill the slot. This outcome is then used to generate output templates.

The trained model used in ALICE structure in figure 3 is generated during the training phase. The design of training module is as shown in figure 4. The input to the training module is a set of articles and manually generated set of templates associated with them. The articles which are in free-text are first passed through the pre-processing module as explained in the next sub section. The parsed output from the pre-processing module is used by the feature extractor which gives as output a list of features for each noun phrase.

The template reader extracts clues from the templates. Clues are a list of slot fill values in the manually generated templates used for training. In the ALICE system, one classifier is built per slot or one classifier per group of slots. The slots are grouped intuitively based on the meaning of the slot. For example, human target name slot and human target description slots are grouped together and one classifier is built for these two slots together.

The classifier classifies the list of noun phrases, and their associated features, into sets of positive and negative examples. A noun phrase is a positive example if it fills a slot in the template i.e. it matches one of the clues extracted from the template. This list of positive and negative examples is used by the maximum entropy trainer to train a Maxent ALICE model which is a core component of the system that generates templates for the given articles.

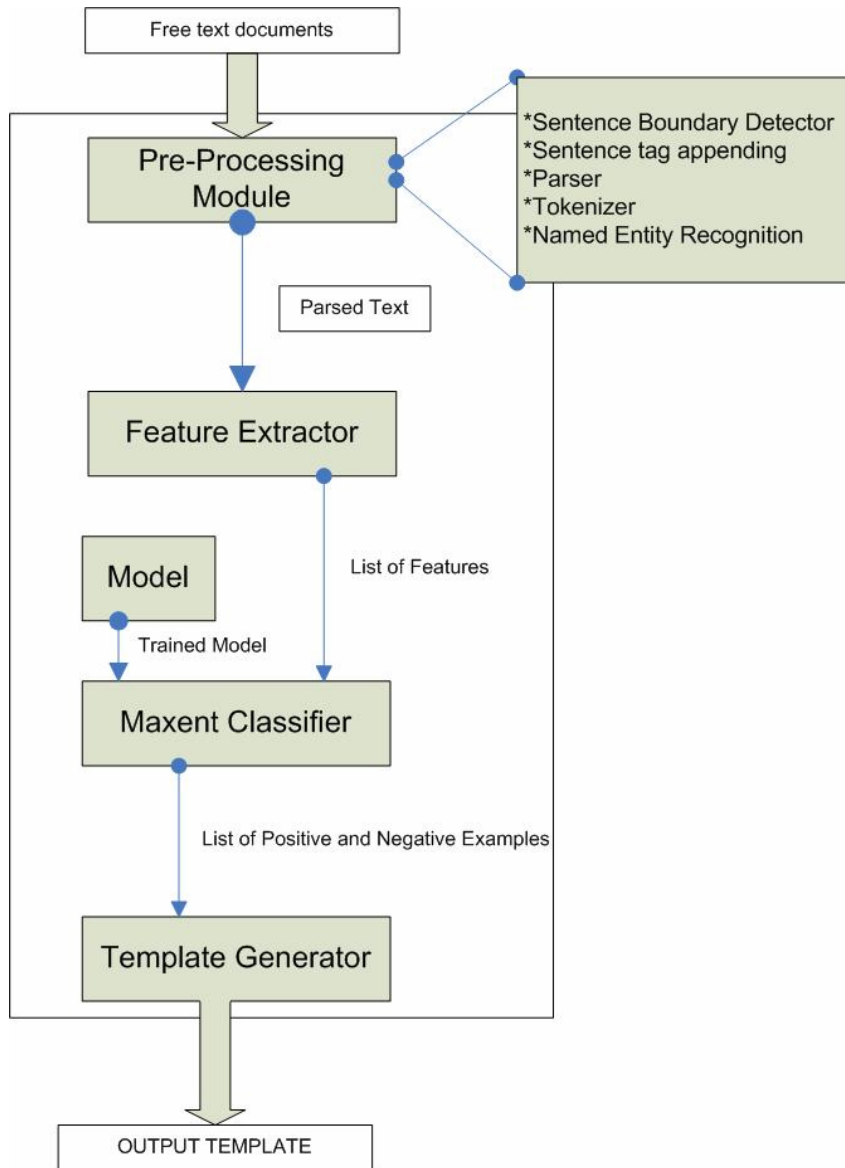
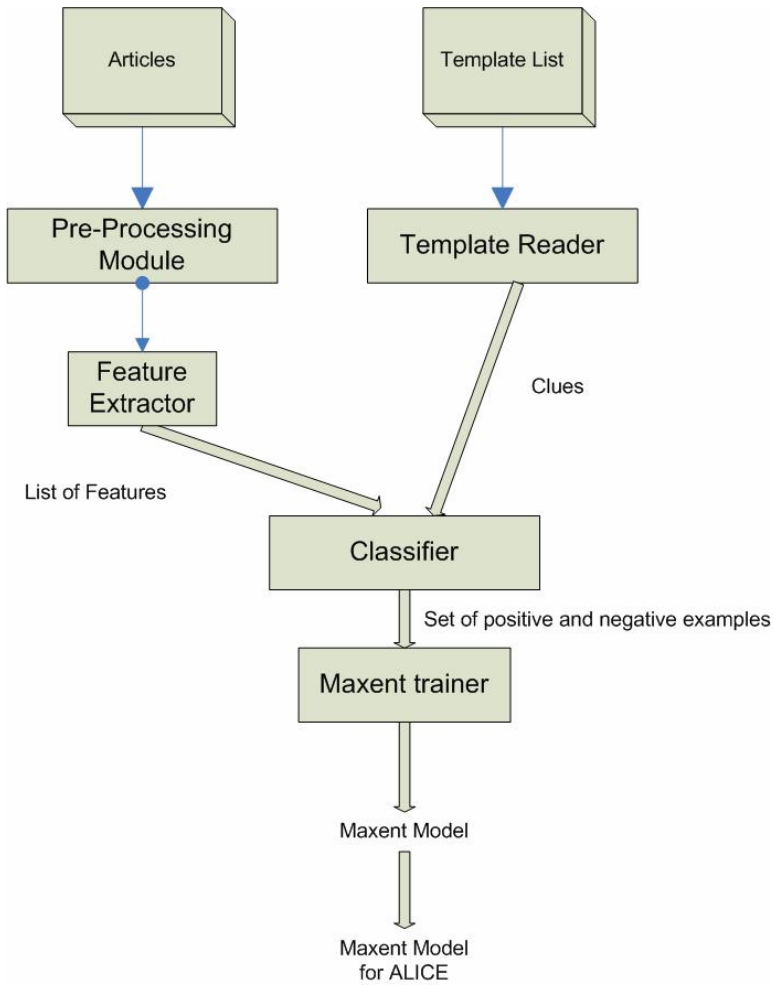


Figure 3: ALICE - the classifier model



Training Module

Figure 4: ALICE - Training Module

Weakly labeled data:

The system developed by Subramanian et. al. [12] has been used to retrieve weakly labeled documents. For a given manually labeled document, articles of the same date (as manually labeled document) or the next date and referring to the same event (as manually labeled document) form the source for its weakly labeled documents. They restricted their search for weakly labeled documents to articles describing a death event, or in other words a terrorist event which resulted in death of the victim. The key idea that they have used is

that it is highly probable that news articles of same date as the event or the next day's date that contain the victim's name would most likely describe the same event and hence can be used as a source of weakly labeled data.

Thus, given a date and a human target name, the corpus is scanned for articles with the same date and the next day's date. The articles retrieved are scanned for the presence of the human target's name. If the name occurs in the article, it is retained; otherwise it is ignored. These articles are termed as weakly labeled data.

The architecture of weakly labeled data extraction system is shown in figure 5. Once the weakly labeled documents are extracted, a template for the weakly labeled document is generated from the template of corresponding manually labeled data.

Weakly labeled data is used in our system along with the manually labeled data as an additional input to the training module (figure 5). And the performance of the system with and without the weakly labeled data is analyzed.

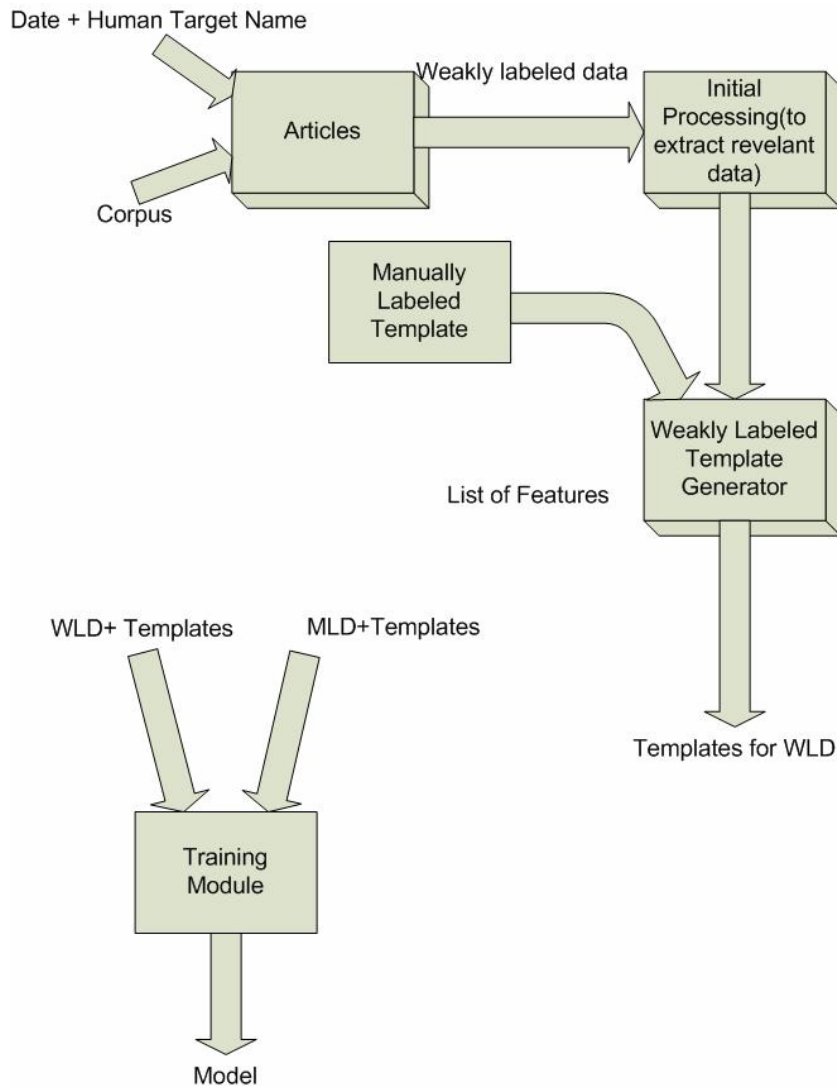


Figure 5: Weakly labeled data Model

3.3 Module Description

3.3.1 Pre-processing Modules

The preprocessing module consists of several small modules used on the free-text data before it can be used for feature extraction and other processing. The input to the preprocessing module is a free-text document and output is a document containing one parse tree per sentence. We also get as output a named entity tagged file which tags every word in a sentence. Two main components of the pre-processing module are:

1. **Parser module:** The parser module consists of following three steps:

- Sentence boundary detector: The sentence boundary detector detects the end of a sentence. The output of this sub-module is the input document with each new sentence starting at a new line. The reason we do this is because the parser used requires the input in a certain format.
- Adding sentence tags (<s> ... </s>): As per parser's requirements, the sentence tags are added at start and end of every sentence.
- Parser: The parser takes ASCII text with sentences delimited by <s> ...</s>, and outputs the parsed versions in the Penn tree-bank style. So if the input is

<s> (`He'll work at the factory.") </s>

The output will be:

```
(S1 (PRN (-LRB- -LRB-)
      (S (` `)
          (NP (PRP He))
          (VP (MD 'll)
              (VP (VB work) (PP (IN at) (NP (DT the) (NN factory))))))
          (. .)
          (" "))
      (-RRB- -RRB-)))
```

The words in bold above are the words from the sentence and other words like VP, NP, DT etc are the POS tags associated with each word or phrase. For example, NP is a tag assigned to a group of words that qualify as Noun Phrase (NP). An alphabetical list of POS tags is included in the Appendix A for reference.

2. **NER module:** The task of NER module is to identify the noun phrases that are names, and assigning a class to each name. It tags noun phrases with the following four classes: person (PER), organization (ORG), location (LOC), and miscellaneous (MISC). The NER module used in this project uses a maximum entropy approach. The NER system requires input in a tokenized format, so a tokenizer software is also used.

As explained in the overall architecture, there are few main components involved which are described below in detail

3.3.2 Feature Extractor

The input to the feature extraction system is the parsed document from the pre-processing phase and output of this sub-module is a list of features. Before going in to the design details, some terminology needs to be defined.

Base Noun Phrase (baseNP):

After the pre-processing module, what we get is one parse-tree per sentence in the article. From this parse tree, we need to extract the potential slot filling values. Since, the template slot value is always a noun phrase; we need to generate all possible noun phrases from a parse-tree and then classify these noun-phrases as positive or negative examples. Positive examples for a slot 'A' are those noun-phrases which fill this slot in the template.

A noun-phrase can be composed of one or more noun-phrases, but what we want is a base noun-phrase (baseNP) i.e. the minimal noun-phrase because a minimal noun-phrase would actually fill the slot. Thus, from a parser output, we generate a list of baseNPs.

For example, if we have a phrase “*Mayor of Achi*”, the parsed output for this sentence would be “(NP (NP (NN MAYOR)) (PP (IN OF) (NP (NNP ACHI))))” where ‘NP’ tag means a noun phrase. As is clear, parsed sentence contains a parent noun phrase which has two other noun phrases (NPs) Mayor and Achi. Thus, the phrase “Mayor of Achi” in itself is not a base noun phrase but “Mayor” and “Achi” separately form two base noun phrases.

Feature Extraction: Once we have a list of baseNPs, we generate the features which define a noun-phrase. Features used in the system developed here are as described below:

1. Head Word: Head word is the most important part of a baseNP. For e.g. For the phrase “About 50 Peasants”, head word is: PEASANTS.
2. Verb of the Noun Phrase: Verb in the sentence of the noun phrase is used as a verb. For example, in *A killed B*, verb *kill* is used as a feature.
3. Preposition associated with the verb: Prepositions associated with the verb of Noun Phrase (NP). For example, in *A was shot to death*, *Shoot_To_Death* is an important feature.
4. Prepositional Phase, Noun Phrase is part of: Sometimes, the baseNP is not directly related to a verb but it is a part of the Prepositional Phrase associated with the verb. For e.g. *A was killed by B*, *kill_by* is used as a feature for baseNP
5. Noun-Preposition pair: Sometimes a noun and preposition phrase pair is used to convey information. For example, “MURDER OF THE PRIESTS”. This feature

aims at capturing information in such phrases. If the baseNP is PRIESTS, this feature will be MURDER-OF.

6. Other noun phrases associated: Usually a verb in a sentence has its subject and object noun phrases. If the base noun phrase we are considering is a subject in the sentence, then the other noun phrase in the object of the sentence is used with the verb as a feature. For example, in A killed B, if our noun phrase is A, then kill_B is used a verb.
7. Named Entity of the base noun phrase: Named entity tagger identifies a noun as noun for a person, location or organization. A human target will be a person, place of event will be a location and perpetrator organization will obviously correspond to an organization named entity. Thus, named entity can give us information about whether a given noun phrase could be a possible slot filler or not and hence named entity is used as one of the features for the bNP. The named entity of a given NP can be taken as the NE of its Head Word.

Above, we gave a brief description of the features used in this system. The details of how these features are extracted given the parse tree are discussed in the next chapter.

Morphological Stemming:

The verbs features such as agent verb or patient verb could be in any form. For example, verb kill can be in forms like kills, killed, had killed, is killing, was killing etc. But what we are interested in from feature point of view is the pure verb form kill because no matter in what form it is used, it will always imply that something or somebody was killed. So, we need to extract the pure forms of verb features. This is important because otherwise if

the system was trained with sentence “A killed B”, it would never be able to identify C as victim in “E saw D killing C”.

3.3.3 Template Reader

The template reader is used during training while building a slot classifier for a given slot. It extracts the slot value from the template and this slot value is used by classifier to classify baseNP as positive or negative training examples.

3.3.4 Maxent Classifier

Maxent classifier is used while training the system. The main objective while building the Maxent classifier is to classify the set of baseNPs into positive or negative examples. In the model that we are trying to build, we have one classifier per slot or for a group of two slots. Given a list of noun phrases and a template associated with them, those noun phrases which contain the fill value for the slot we are building a classifier for, are used as positive examples and other remaining base noun phrases are used as negative examples.

Coreference resolution: Coreference resolution is the process of determining whether two expressions in natural language refer to the same entity in the world. This is useful in building a classifier by helping us in identifying the positive examples more accurately. If a noun phrase A is a positive example then a noun phrase B that corefers A should also be taken as a positive example as they essentially refer to the same entity.

In this project, a very simple coreference system has been implemented based on the approach used by Soon et. al. [11]. Three criteria defined below are used to identify if two noun phrases are coreferences of each other.

- **String Match:** To compare two strings for a possible match, we first remove articles (*a, an, the*) and demonstrative pronouns (*this, these, that, those*) from the

strings before performing the string comparison. Therefore, *the license* matches *this license*, *that computer* matches *computer*.

- **Acronyms:** For organization names, we check for acronym match such as *IBM* and *International Business Machines Corp.* In this case, the longer string is chosen to be the one that is converted into the acronym form. The first step is to remove all post-modifiers such as *Corp.* and *Ltd.* Then, the acronym function considers each word in turn, and uses its first letter to form the acronym. Two variations of the acronyms are produced: one with a period after each letter, and one without.
- **Appositive Feature (APPOSITIVE):** In sentence, “*Bill Gates, the chairman of Microsoft Corp*” the phrase “*the chairman of Microsoft Corp.*” is in apposition to “*Bill Gates*”. Our system determines whether *j* is a possible appositive construct by first checking for the existence of verbs and proper punctuation. Like the above example, most appositives do not have any verb; and an appositive is separated by a comma from the most immediate antecedent, *i*, to which it refers. Further, at least one of *i* and *j* must be a proper name. And to check if a noun phrase is a proper noun or not, we use the Named Entity tag which is I-PER means it’s a person’s name.

3.3.5 Template Generator

The classifier gives as output a list of noun phrases and probabilities associated with them, probability of how likely it is that this noun phrase would fill the slot in the template. From this output we need to build the templates. The template generator considers all noun phrases and takes those noun phrases with probability > 0.5 as slot fillers.

3.3.6 Maximum Entropy training and prediction model

In this project, a maximum entropy approach is used to train a model for information extraction system. A maximum entropy system is generally used to solve statistical classification problems where our objective is to find the probability of a class and context occurring together [3]. The class and the context are defined based on the NLP task. For building the slot classifier, context is the set of features for a give phrase in the parse tree and class is whether given the features set for the phrase, it qualifies for the slot or not.

First, the maximum entropy system is used to train and develop a classifier model using the list of features generated from the manually labeled data. Once the system has learnt the association between features it has seen and the slots, it is used to predict the class for the new set of features from testing documents. In other words, given a list of noun phrases and their feature values, the Maximum Entropy predictor helps in making a decision as to whether a given noun phrase should fill the slot or not. In the next section, we describe the implementation details of the design modules defined in this section.

CHAPTER 4 IMPLEMENTATION DETAILS

4.1 External software used

1. **MXTerminator [8]:** MXTERMINATOR is a JAVA (JDK 1.1) implementation of the sentence boundary detector. This is used as a first step in pre-processing module. Output of this system is same as input document with one sentence per line. This is done to convert the data to the format in which the parser requires its input to be in.
2. **Charniak Parser [4] :** In this project, we have used the parser developed by Eugene Charniak based on the paper "A maximum-entropy-inspired parser" []. The system produces parses at an average rate of about 2 seconds per sentence on my 450 MH Sun (after about 45 seconds to load all of the data files). We have used version 4.0 which has a precision/recall of about 89.8% on the standard Penn treebank test set.
3. **NER:** Named Entity Recognition system developed by another student Jin Kait is used in this project. This NER system is based on work done by Chieu et. al. [6]. The NER model the system provided was trained on mixed case data. But the MUC4 documents used in this project are all upper case data. Since, the NER system uses the case of the letters in the word as one of the feature to assign the name entity, the trained model provided could not be used on the fly. The system was trained again by converting the training data to upper case and then using the model.
4. **Tokenizer:** Sed tokenizer script has been used here to convert the data into the format NER requires its input to be. Input document for NER tagger consists of one token per line and a header at the start of the document.
5. **WordNet 2.0. [15]:** WordNet is used in this project for morphological stemming. WordNet is an online lexical reference system. The source code of WordNet was

integrated with our system and the morphological function provided by WordNet is used in this project. Given a verb in any form, the morphstr() function of WordNet gives you the pure morphological form / base form (lemma) of the word.

6. **Wnjn 1.6. [14]:** WordNet's implementation is in C. This package allows you to use WordNet from Java using a JNI interface to the C code. This was really helpful as the project has been done in Java as well.
7. **The Maximum Entropy Framework (Maxent 2.3.0) [13]:** The opennlp.maxent package is a Java package for training and using maximum entropy models. The data for a classification problem is described as a set of constraints. Given the constraints, the maximum entropy model is computed, the model with the maximum entropy of all the models that satisfy the constraints. Choosing the maximum entropy model is motivated by the desire to preserve as much uncertainty as possible.

To create a model, one needs the training data, and then implementations of two interfaces in the opennlp.maxent package, EventStream and ContextGenerator. The example implementations provided were extended for integration with our code by modifications on how context data or features are read from the file and how outcome results are presented.

4.2 Implementation of String Slots

In this project, the aim was to develop a classifier for string slots and experiment with using weakly labeled data to improve system performance. In the implementation of this system, the important steps and their implementation details are discussed below.

Implementation steps for training phase:

1. Use parse trees from preprocessing module to retrieve base noun phrases.

2. Extract features of the base noun phrases
3. Classify the base noun phrases as positive or negative examples
4. Use Maxent system to train the model

Implementation details for testing phase:

1. Use parse trees from preprocessing module to retrieve base noun phrases.
2. Extract features of the base noun phrases
3. Use the model to Predict
4. Generate Templates from predicted probability for slot fillers

Step 1 and 2 are common to both the training and testing phase. These are discussed in detail below.

1. Finding base Noun Phrase (bNP)

Task: Given a parse tree, find the base noun phrases associated with it.

Algorithm:

```

Traverse down the tree from root
if reached end of tree
    return
else if node = = NP
    check if any of this node's children is another NP
    if yes
        ignore this NP and traverse further down
    else
        base noun phrase found, add it to the list
else if node other than NP
    traverse further down

```

2. Finding features:

- Head Word: Given a base noun phrase, a head word is the rightmost leaf node of the baseNP sub-tree. For e.g. For the phrase “About 50 Peasants”, parse tree is

(NP (QP (IN ABOUT) (CD 50)) (NNS PEASANTS))

Head word here is: PEASANTS.

- Named Entity (NE): The named entity information is associated with the leaf nodes. Named entity of a base noun phrase is taken as the NE feature of its head word.

For other features, we need to first figure out whether the baseNP is an *agent* or *patient* of the sentence. This decision is based on two things, whether the bNP is the subject or object in the sentence and whether the verb in the sentence is used in active or passive voice.

((subject && active) || (object && passive)) => Agent

For example,

- (subject && active) => A killed B, A is the agent
- (object && passive) => B was killed by A, A is the agent

((subject && passive) || (object && active)) => Patient

For example,

- (object && active) => A killed B, B is the patient
- (subject && passive) => B was killed by A, B is the patient

Algorithm to determine if a given bNP is subject or object of a sentence:

```

 Traverse up the tree from the base noun phrase node
   if bNP is part of a verb phrase,
     then it is an object
   else if going up the tree we reach the S node i.e. the
 Sentence node
     then it is the subject.

```

Active or passive voice:

We traverse up the tree from the bNP node to find the VP (verb phrase) node of the sentence. The POS tags in the sub-tree of VP give us information about the nature of the verb. The table below shows information about these tags:

VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present

- The POS tags like VBG, VBZ and VBP are always associated with the verb used in active form. These three classes together cover sentences like *was eating /is eating /will be eating /has been eating /he eats /they eat*
- However, for POS tags like VBN and VBD, auxiliary verb such as *was, is* etc also play a part in deciding whether it is an active verb form or passive verb form.
- VBD when used without an auxiliary verb, it is an active verb. For example, *He ate*. Actually, VBD should always be used without an auxiliary verb but if incase it is used with an auxiliary verb then we should tackle it similar to VBN as explained below.
- For VBN, if it is used with auxiliary verbs such as *has, have* it is an active verb form but with other auxiliary verbs such as *was, were*, it is a passive verb form. For example, *has eaten, has reported* are active verb forms, where as *was killed* is a passive verb form. However, this rule can be applied if there is only one auxiliary verb. If we have say two auxiliary verb like *has been killed, would have been killed, was being tortured* etc, then in this case we can't classify the verb into active or passive form by just considering verbs like *has* and *have*, we need to consider other auxiliary verbs as well.

Once we have classified the noun phrase as subject or object of the sentence and verb as active and passive voice, we know whether the noun phrase is an agent or patient of the sentence. Then, the features to be extracted are described below. These features are same as what we described earlier. Here we discuss some implementation details of these.

- **If baseNP is the agent**, e.g. A killed B, A is the agent
 1. **Verb of the agent NP, VAg:** If the baseNP in the sentence is the agent then all verbs associated with it are used as the features. For e.g. in sentence above, verb *kill*.
 2. **Agent's verb and its patient, VAg_Pa_NP:** The agent's verb might have a patient associated with it, like in the case above. Then, *kill_B* is used as a feature where B is the patient of the verb kill. To find the patient noun phrase, traverse up the tree from the agent base NP until you reach root of the sentence 'S' and then explore sub-tree of the other child of root node to find a base noun phrase.
 3. **Prepositional phrase of the agent's verb, VAg_PP_NP:** The preposition phrase associated with the agent's verb can sometimes provide useful information such as in e.g. A shot B to death, *Shoot_To_Death* is an important feature. This information we can get by traversing the sub-tree of VP of the base NP.

- **If baseNP is the patient**, e.g. A was shot to death, A is the patient

1. **Verb of the patient NP, VPa:** If the baseNP in the sentence is the patient then all verbs associated with it are used as the features. For e.g. in sentence above, verb *shoot*.
2. **Patient's verb and its agent, VPa_Ag_NP:** The patient's verb might have an agent associated with it. e.g. in, B killed A, *kill_B* is used as a feature for the patient A. To find the agent noun phrase, traverse up the tree from the patient base NP until you reach root of the sentence 'S' and then explore sub-tree of the other child of root node to find a base noun phrase.
3. **Prepositional phrase of the patient's verb, VPa_PP_NP:** The preposition phrase associated with the patient's verb can sometimes provide useful information such as in e.g. above, A was shot to death, *Shoot_To_Death* is an important feature.

This information we can get by traversing the sub-tree of VP of the base NP.

- **If baseNP is a part of the Prepositional Phrase (PP):**
 1. **Preposition of the verb, VP_Prep:** Sometimes, the baseNP is not directly related to a verb but it is a part of the Prepositional Phrase associated with the verb. For e.g. A was killed by B, *kill_by* is used as a feature for baseNP B. To find this feature, traverse up the tree from base NP first to find if it has a PP as its parent node. And then traverse up further to check if PP has a VP as its parent. Then the preposition and verb together would be used as the feature.
 2. **Noun-Preposition, NP_Prep:** This feature aims at capturing information in phrases such as "MURDER OF THE PRIESTS". If the baseNP is PRIESTS, this feature will be MURDER-OF. Like in previous feature, we traverse up

the tree from base NP first to find if it has a PP as its parent node. And then traverse up further to check if PP has a NP as its parent. Then the preposition and noun together would be used as the feature.

This list of features is prepared for all slot classifiers and is used for training a maximum entropy model.

The two steps described above namely finding the base noun phrases and feature extraction, are common to both training and testing phase. The remaining steps of the two phases are discussed below.

Training phase continued:

3. Classify the base noun phrases as positive or negative examples

If we are building a classifier for HUM-TGT slot, then strings in the HUM-TGT slot in the template are used to differentiate between positive and negative examples. Positive examples are those in which the base noun phrase contains one of the strings that fills the HUM_TGT slot for the article. The coreference module described in the design section, is used here to find the coreferenced base noun phrases of positive examples and include them in the positive example set.

4. Maxent Classifier:

With the features extracted and classified as positive or negative example, a slot classifier is built for the slots in the template using the Maximum-entropy framework. The `opennlp` package used for this purpose provides source code which was modified a little for using it with our system.

Testing phase continued:

3. Use the model to Predict:

The model trained by our system is now used to predict the class for context of test articles. By context, we again mean a set of features. The model assigns each of the base noun phrases a probability that this noun phrase would fill the slot.

4. Generate Templates from predicted probability for slot fillers

The outcome of the predictor is written to a file. The output template generator parses information in this file and fills the template slots. For few slots such as HUM TGT name and HUM TGT description, one classifier is used as they essentially classify similar information. Thus, there will be one outcome for both the slots together. But while generating the template, we need to differentiate the outcome between these two slots. To do this, we use the Named Entity information. The base noun phrase with named entity as I-PER implies that the noun phrase is a proper noun and hence a suitable candidate for the HUM-TGT Name slot. Other noun phrases are classified as HUM-TGT description. Similarly, to differentiate between Perpetrator individual id and organization id slots, named entities I-PER and I- ORG is used.

4.3 Using weakly labeled data

The system developed by Subramanian et. al. provides a set of weakly labeled documents. These articles from the CORPORA are present in the SGML format, only the text portion is extracted from them. Templates for the weakly labeled data are prepared using the templates for the corresponding manually labeled data. Few changes are made to the template such as Message ID and INCIDENT date is modified according to the weakly labeled document. Also, strings or slot values in the template which are not present in the weakly labeled document are removed.

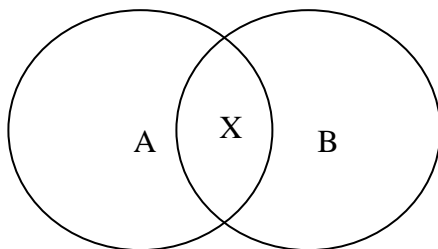
The system by Subramanian et. al. gives us a set of weakly labeled documents with their corresponding template set. This set of documents and templates can be used directly with out system by just appending the templates for weakly labeled documents in the template list and adding weakly labeled articles to the directory where manually labeled data resides. The weakly labeled data also goes through the same set of processing modules as manually labeled data.

CHAPTER 5 RESULTS AND DISCUSSION

5.1 Scorer used

The scoring system provided by muc4 has been used in this project. This scorer is designed to test and compare the complete templates. However, this project concentrates on building classifier for String Slots only and hence the output template would only contain values for the string slots. To still be able to use the muc4 scorer for this project, the non-string slots were filled with values from the key template (one with the correct answer). Thus, the score we get would give us an idea about the system performance in classifying string slots. Also, in our system we only generate one template per article, so even the key template is modified so that multiple templates per articles are not considered.

Scores are reported in terms of three popular scoring measures for NLP systems: Recall, Precision and F-Measure. Recall and precision can be best described by the figure below:



If circle A represents the desired results and B the observed values and X the commonality between them, then

$$\text{Recall} = X/A$$

$$\text{Precision} = X/B$$

F-measure is defined as $[(2 * R * P)/(R + P)]$

5.2 Tests conducted

The effect of varying the training data size on performance of the system was analyzed for two cases: Manually Labeled Data (MLD) alone and Manually Labeled Data (MLD) + Weakly Labeled Data (WLD) (MLD+WLD).

In our experimental setup, we ran about 10 random trials. Each trial consisted of training on randomly selected Manually Labeled Document-set (MLD) of sizes 5, 10, 20 and 30 and subsequently training the same document-set augmented with Weakly Labeled Documents (MLD+WLD) and finally comparing the performance in both scenarios. The documents set of sizes 5, 10, 20 and 30 documents are such that set 5 is a subset of set 10 and set 10 is a subset of set 20 and so on. The score reported here is the F-measure value for string-slots averaged over results of 10 trials. The graphs in figure 6 and 7 shows variation of average F-measure value with the size of document set for both MLD and (MLD+WLD). Tables 1 & 2 show the percentage change in F-measure between MLD and (MLD+WLD) sets averaged over 10 trials. As can be seen, the score improves with addition of weakly labeled data in case of small training set but as the number of manually labeled documents increases in the training set, supplementing it with the weakly labeled documents does not boost the performance of the IE system. The score deteriorates for large sized document-sets.

MLD	Average Percent Change in F-measure
5	23.82357473
10	7.549981807
20	-3.700719638
30	-18.76440415

Table 1: Average Percentage change in F-measure from MLD to (MLD+WLD) for TST3

MLD	Average Percent Change in F-measure
5	13.09095
10	-17.0832
20	-25.8262
30	-39.4338

Table 2: Average Percentage change in F-measure from MLD to (MLD+WLD) for TST 4

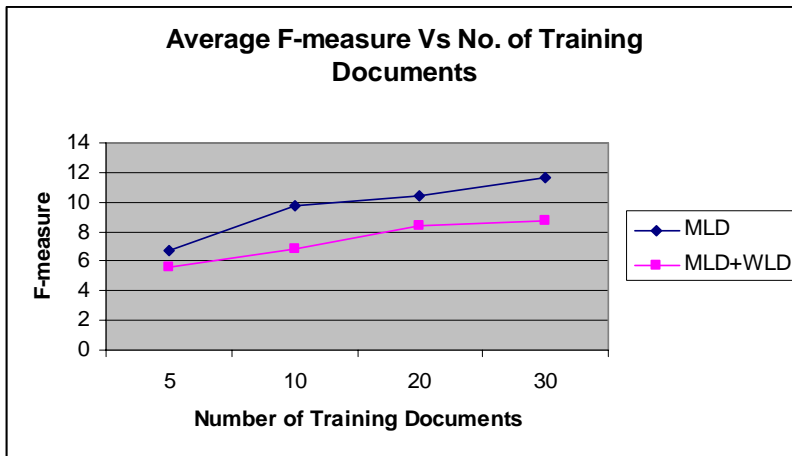


Figure 6: Average F-measure Vs No. of Training documents TST3

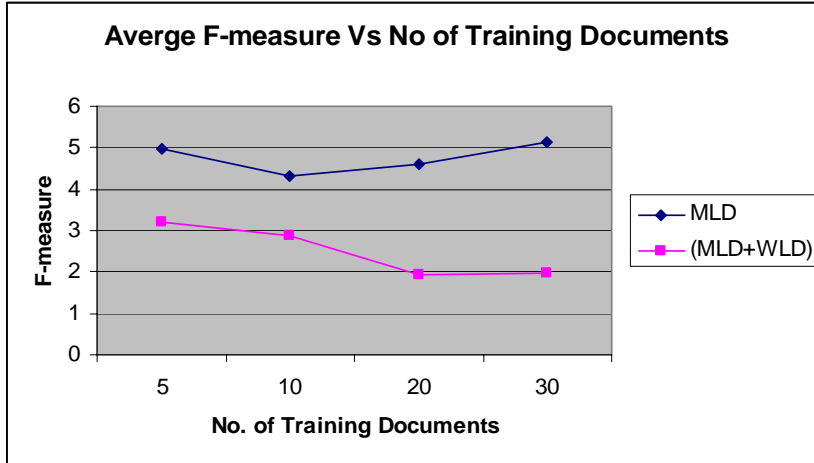


Figure 7: Average F-measure Vs No. of Training documents for TST4

Another experiment was done with larger document sets. In this case, the training document set size was varied as 100, 200, 500, 800 and 1300. We got around 212 weakly labeled documents from Subramanian’s system. So, test sets for weakly labeled data is $(x+212)$ where 212 is fixed set of weakly labeled data we have. The results for this test case are as shown below. The performance doesn’t improve on addition of weakly labeled data to weakly labeled data. The results obtained are as shown in Table 3 & 4 for test documents TST3. The graph is shown in figure 8. Corresponding results for TST4 test documents are shown in table 5& 6 and figure 9.

Results for TST3 test documents

Without weakly labeled data (MLD –Manually Labeled Data)

MLD	Recall	Precision	F-measure
100	12	39	18.35294
200	11	38	17.06122
500	15	31	20.21739
800	21	42	28
1300	20	37	25.96491

Table 3: Performance for MLD on TST3 files

With weakly labeled data (MLD +WLD –Manually Labeled Data + Weakly Labeled Data)

MLD	Recall	Precision	F-measure
100	7	38	11.82222
200	7	38	11.82222
500	5	21	8.076923
800	13	43	19.96429
1300	15	43	22.24138

Table 4: Performance for MLD+WLD for TST3 files



Figure 8: Performance for TST3 files with MLD & WLD

Results for TST4 test documents

Without weakly labeled data (MLD –Manually Labeled Data)

MLD	Recall	Precision	F-measure
100	13	46	20.27119
200	15	41	21.96429
500	23	45	30.44118
800	22	45	29.55224

Table 5: Performance for MLD on TST4 files

Without weakly labeled data (MLD +WLD –Manually Labeled Data + Weakly Labeled Data)

MLD	Recall	Precision	F-measure
100	6	48	10.66667
200	12	45	18.94737
500	14	43	21.12281
800	18	50	26.47059

Table 6: Performance for MLD + WLD on TST4 files



Figure 9: Performance for TST4 files with MLD & WLD

Performance results reported by Subramanian et al.:

The experiments conducted by Subramanian et. al., like ours, comprised of training on document-sets of sizes 5, 10, 20, and 30, and retraining with weakly labeled documents. And observations in this project tally with the observations made by them. To summarize the common observations, supplementing the training data-set consisting of manually labeled documents with weakly-labeled documents does enhance performance of an IE system but such improvement is only observed when documents-sets are small. As the number of manually labeled documents increases in the training set, supplementing with

the weakly labeled documents does not boost the performance of the IE system. The score deteriorates for large sized document-sets.

Discussion about results:

A part of this project was to verify results obtained by Subramanian et. al. for weakly labeled data because it is quite intuitive that addition of more training data should improve the performance. As can be seen from the graph and the tabulated results, results of this project tally with Subramanian et. al's results. The performance of the system reduces with addition of weakly labeled data when the document size is large.

This could be attributed to the increased noise when large training document sets are augmented with the weakly labeled documents, resulting in a declined performance. Possible reasons that could be explored in future work to justify this observation verified twice (i.e. in this project and Subramanian et. al's work) are discussed below:

1. **False Positives:** False positives are those noun phrases which are wrongly classified by the system as positive examples. As mentioned above, the classifier used in our system classifies all noun phrases which contain the slot fill values as positive examples. However, there is a very high chance that if an article describes the event of death of a person, it would mention its name only once in context of death and other occurrences of the name would provide other information about the person like where is lived, what he did for living etc. Thus, it is not appropriate to consider all occurrences of the slot fill values as positive examples.

However, the error due to false positive is very much dependent on the type of data. If the data is such that, in most of the articles there is always one occurrence of the slot fill value then there are lesser chances of getting false positives. But if the data has many references of the noun phrase then this could lead to many false positives and hence inaccurate results.

False positives could also become a serious problem if the slot fill value is a common noun for example 'people'. This would mean that many noun phrases in the articles could match this common noun and hence we would get a long list of positive examples, most of which could be false positives.

Thus, it is possible that weakly labeled data that we are using is more prone to false positives which would result in degraded performance. In order to find out whether we have this problem in weakly labeled data, we need to manually go through a set of weakly labeled data and look for such occurrences.

2. Nature of Test data: could be too similar to the training data

Another reason for lower performance with weakly labeled data could be that the test data is very similar to the manually labeled training data i.e. verbs and other features used in writing are common between the two but the test data could be very distant from the weakly labeled data in the sense that verbs used in the weakly labeled data are hardly used in the test data. In such a case, system would perform very well with the manually labeled training data due to its similarity with the test data and addition of the weakly labeled data would just add noise to the data due to its lack of similarity with the test data.

Moreover, every module or external software used in our system would have certain accuracy it provides and there will always be a margin of error. As many such modules are used in this project, the error is also propagated into our output. And this error would be more for a larger set of data. Thus, adding more data hence would also add some error.

So, we would have weakly data which is very distant from our test data but it has added some noise to our system, there by resulting in lowered performance.

Again, to analyze whether this is the case or not, we need to inspect the training, the test and the weakly labeled data.

3. **Inappropriate template association:**

As mentioned earlier, in Subramanian et. al. work that we have used, templates for weakly labeled data are generated using templates of corresponding manually labeled articles. Some information from the template is modified or removed so that it can be linked with the weakly labeled data. Some mismatch or inappropriate association between the article and the template could also be a reason for low performance. To verify this, we need to cross check the template and article pair and verify whether the information contained in the article matches the information in the template.

A good approach to solve this problem would be to check the above mentioned possible reasons and possibly some more reasons as well. Verifying above three reasons would

require a lot of manual inspection and a good sampling strategy to randomly select articles for testing.

CHAPTER 6 CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this project a maximum entropy based Information Extraction system has been developed. The system is able to extract the string slot values such as Hum-tgt name, hum-tgt description, Instrument used in incident, perpetrator etc. As part of the project, experiments were done to explore the possibility of using weakly labeled data to improve system performance. The system for extracting weakly labeled data was developed by Subramanian et. al. Although, intuitively it feels that performance should improve with use of weakly labeled data as we are essentially using more training data. But it was observed and also verified Subramanian et. al.'s observation that weakly labeled data fails to improve system performance. The possible intuitive reasons for these are discussed in the previous section.

We need to analyze the data involved in the project i.e. the training data, weakly labeled data and the test data to justify the observations made. In the previous section, we have proposed few approaches that could be taken to reason out these observations.

6.2 Possible Improvements

Few of the possible improvements that can be incorporated in the system and would result in better efficiency are discussed below:

Improvement of NER system: The NER system used in this project is trained using a mixed case text and it uses the case information as one of the features to distinguish between the proper nouns and common nouns. However, the MUC4 data used to build our system is all in upper case. Thus, there is some compromise in performance because information dependent on case is lost. This could be improved possibly by converting MUC4 data to mixed case using some NLP techniques or to modify the NER system such that it performs better with upper case text by possibly using some other characteristics of the data.

Improvement on group classifier. For some pair of slots, one classifier is built and during prediction, the outcome i.e. the set of result values is divided between the two slots depending on the named entity of the base noun phrase. For a given base noun phrase, named entity associated with the base noun phrase is taken as the named entity tag of the head word of the noun phrase. However, this assumption can lead to improper answer sometimes. For example, if the base noun phrase is “Former President Bill Clinton”, in this name entity of the whole noun phrase, under our assumption, would be taken as named entity of head word Clinton which is I-PER. Now, based on this named entity, this noun phrase would be used fill the HUM-TGT Name slot and HUM-TGT description slot will be empty. However, the correct answer would be HUM-TGT Name should get value Bill Clinton and HUM-TGT Description should get Former President as the slot value. To do this we need to consider the name entity of each word in the base noun phrase and possibly split the noun phrase into two parts and divide the answer between the two slots.

Improvement on False Positives:

Currently in the system, all base noun phrases which contain the slot fill value are taken as positive examples for training the classifier. However, it is very likely that only one occurrence of the victim's name in the article refers to his/her death and other occurrences talk about the person in general say about his education or living. Thus, in such a case it would be very wrong to take all occurrences as positive examples. These positive examples which are wrongly qualified as positive are termed as false positive. The false positives add noise to our data. And this noise increases as the number of training documents increase and hence performance may not necessarily improve with increasing training data.

To fix the problem of false positives, we could use more information from the template to classify a base noun phrase as positive example or not. Usually a sentence which contains the real positive occurrence would also contain information from other slots of that template as well. For example, a sentence saying B died, would usually contain information about how, where or when.

CHAPTER 7 REFERENCES

- [1] Regina Barzilay and Lillian Lee, “Learning to Paraphrase: An Unsupervised Approach Using Multiple-Sequence Alignment”, Proceedings of HLT-NAACL 2003 Main Papers , pp. 16-23 Edmonton, May-June 2003
- [2] Regina Barzilay and Kathleen R. McKeown. 2001. “Extracting Paraphrases from a Parallel Corpus.” Proceedings of the ACL 2001 Toulouse, France.
- [3] Adam L. Berger, Stephen A. Della Pietra and Vincent J. Della Pietra. “A maximum entropy approach to natural language processing.” Computational Linguistics, 22(1):39-71, 1996.
- [4] Eugene Charniak, “A Maximum-Entropy-Inspired Parser”. Proceedings of NAACL-2000
- [5] Chieu, Hai Leong, Ng, Hwee Tou, & Lee, Yoong Keok (2003). “Closing the Gap: Learning-Based Information Extraction Rivaling Knowledge-Engineering Methods”. Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03). (pp. 216-223). Sapporo, Japan.
- [6] Chieu, Hai Leong, & Ng, Hwee Tou (2003). “Named Entity Recognition with a Maximum Entropy Approach”. Proceedings of the Seventh Conference on Natural

Language Learning (CoNLL-2003). (Shared Task Paper). (pp. 160-163). Edmonton, Alberta, Canada.

[7] Adwait Ratnaparkhi. "A Simple Introduction to Maximum Entropy Models for Natural Language Processing." Technical Report 97-08, Institute for Research in Cognitive Science, University of Pennsylvania.

[8] Jeffrey C. Reynar and Adwait Ratnaparkhi. A Maximum Entropy Approach to Identifying Sentence Boundaries. In Proceedings of the Fifth Conference on Applied Natural Language Processing, March 31-April 3, 1997. Washington, D.C.

[9] Fabio Rinaldi, James Dowdall, Kaarel Kaljurand, Michael Hess Diego Molla. "Exploiting Paraphrases in a Question Answering System." The Second International Workshop on Paraphrasing: Paraphrase Acquisition and Applications (IWP2003)

[10] Yusuke Shinyama & Satoshi Sekine, "Paraphrase Acquisition for Information Extraction", The Second International Workshop on Paraphrasing: Paraphrase Acquisition and Applications (IWP2003) post-conference workshop (WS5) in conjunction with ACL2003, Sapporo, Japan, July 11, 2003.

[11] W. M. Soon, H. T. Ng, and D. C.Y. Lim, "A Machine Learning Approach to Coreference Resolution of Noun Phrases," Computational Linguistics, 27(4), pp.521-544, 2001.

[12] G. Subramanian, “Exploiting Weakly Labeled Documents for Information Extraction: A case study”, NUS Honours Year Project Report

[13] Maxent Classifier by OpenNLP, <http://maxent.sourceforge.net/>

[14] WordNet JNI Java Native Support, <http://wnjn.sourceforge.net/>

[15] WordNet, <http://wordnet.princeton.edu/>

Appendix A

Alphabetical list of part-of-speech tags used in the Penn Treebank Project

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	<i>and, but, or</i>	SYM	Symbol	<i>+, %, &</i>
CD	Cardinal number	<i>one, two, three</i>	TO	"to"	<i>to</i>
DT	Determiner	<i>a, the</i>	UH	Interjection	<i>ah, oops</i>
EX	Existential 'there'	<i>there</i>	VB	Verb, base form	<i>eat</i>
FW	Foreign word	<i>mea culpa</i>	VBD	Verb, past tense	<i>ate</i>
IN	Preposition/sub-conj	<i>of, in, by</i>	VBG	Verb, gerund	<i>eating</i>
JJ	Adjective	<i>yellow</i>	VBN	Verb, past participle	<i>eaten</i>
JJR	Adj., comparative	<i>bigger</i>	VBP	Verb, non-3sg pres	<i>eat</i>
JJS	Adj., superlative	<i>wildest</i>	VBZ	Verb, 3sg pres	<i>eats</i>
LS	List item marker	<i>1, 2, One</i>	WDT	Wh-determiner	<i>which, that</i>
MD	Modal	<i>can, should</i>	WP	Wh-pronoun	<i>what, who</i>
NN	Noun, sing. or mass	<i>llama</i>	WP\$	Possessive wh-	<i>whose</i>
NNS	Noun, plural	<i>llamas</i>	WRB	Wh-adverb	<i>how, where</i>
NNP	Proper noun, singular	<i>IBM</i>	\$	Dollar sign	<i>\$</i>
NNPS	Proper noun, plural	<i>Carolinas</i>	#	Pound sign	<i>#</i>
PDT	Predeterminer	<i>all, both</i>	"	Left quote	<i>(' or ")</i>
POS	Possessive ending	<i>'s</i>	"	Right quote	<i>(' or ")</i>
PRP	Personal pronoun	<i>I, you, he</i>	(Left parenthesis	<i>([, (, { , <</i>
PRP\$	Possessive pronoun	<i>your, one's</i>)	Right parenthesis	<i>(] ,) , } , ></i>
RB	Adverb	<i>quickly, never</i>	,	Comma	<i>,</i>
RBR	Adverb, comparative	<i>faster</i>	.	Sentence-final punc	<i>(. ! ?)</i>
RBS	Adverb, superlative	<i>fastest</i>	:	Mid-sentence punc	<i>(: ; ... - -)</i>
RP	Particle	<i>up, off</i>			