ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF COMPUTER AND MATHEMATICAL SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Named Entity Recognition for Afan Oromo

By: **Mandefro Legesse Kejela**

A THESIS SUBMITTED TO
THE SCHOOL OF GRADUATE STUDIES OF THE ADDIS ABABA UNIVERSITY IN PARTIAL FULFILLMENT
FOR THE DEGREE OF MASTERS OF SCIENCE IN COMPUTER SCIENCE

October 2010

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF COMPUTER AND MATHEMATICAL SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Named Entity Recognition for Afan Oromo

BY: **Mandefro Legesse Kejela**

ADVISOR: **Dejene Ejigu (Ph.D)**

APPROVED BY

EXAMINING BOARD:

1. Dr. Dejene Ejigu, Advisor _____
2. _____ _____
3. _____ _____

Dedication

I dedicate this work to my father **Legesse Kejela Dengi** who passed away while suffering a lot to bring a bright future for his children.

Acknowledgements

God did what he promised me. I praise his name. My deepest gratitude goes to my advisor Dr. Dejene Ejigu for his continuous follow up starting from title selection to this end. His constructive comments were very essential and his friendly treatment was admirable. I would like to thank the authors of LingPipe tool Mr. Bob Carpenter and Mr. Breck Baldwin for helping me a lot in those difficult situations and for their quick reply to my questions. I tell you guys, the LingPipe tutorial you prepared is an interesting material that enabled me to quickly understand the tool.

I don‟t think I could quickly come to understand CRFs algorithm if I didn‟t discuss with my friend Moges Ahmed. Mogi, you are really a cooperative and an enjoying friend. Marhaba! I would also like to thank Sultan Mohammad & Walatamariam Seifu from „Bariisaa‟ and Adugna Hailu & Yewubnesh Kebede from „Kallacha Oromiyaa‟ news papers for their cooperation in providing me the softcopy of their respective news articles. Teferi Nugusie, Wandimalem Geneti, Daniel Lamessa and Ebissa Daba tagged Afan Oromo words with their POS tag which helped me to train the POST model. I would like to thank them a lot. Adugna Barkessa helped me much to understand the linguistic nature of Afan Oromo language.

My uncle Mekonnen Dilbo supported and encouraged me as a father. Moke, I will not forget what you did for me and may God bless you & your family. My mother Nuguse Nagassa directed me on the right way of life and sacrificed a lot to raise me. I am very grateful to her. Finally, I would like to extend my heartfelt appreciation to my two brothers Nurfeta Legesse and Baru Legesse for their moral support and encouragement.

Dhaamsa

Yeroo ammaa kana teknoolojiiwwan barreeffama afaan dhala namaa hubatan, afaan uumamaa dubbachuu kan dandaʻan, dhala namaa gorsuudhaan tajaajilanii fi bakka namaas buʻuun hojii adda addaa hojjetan arguun dandaʻameera. Kun hundi buʻaalee qorannoo gama ‚*natural language processing (NLP)*ʻn godhamanii dha. Qorannoon akkasii kun afaanota biyyoota gara dhihaa hedduu fi afaanota Eshiyaa tokko tokkoof baayʻee kan ittiin adeemame yoo taʻullee, Afaan Oromootiif kan jalqabame baayʻee dhiheenyatti. Qorannoon adeemsifamanis baayʻee xiqqoo dha. Biyya keenyatti qorannoon gama kanaan kan adeemsifaman Yunivarsitii Finfinneetti barattoota barnoota digirii lammaffaa isaanii mummeelee *saayinsii kompiyuteraa* (computer science) fi *saayinsii odeeffannoo* (information science) keessatti hordofaniini. Kanneen adeemsifamanis kan jajjabeeffaman yoo taʻanillee, hojjetamanii faffacaʻanii hafun ala qaamni isaan faana buʻuun akkaataa qorannooleen kun hojiitti jijjiiraman kan mijeessu hin jiru. Rakkooleen adda addaa kanneen akka baajata gahaa dhabuu sababa jiraniif qorannooleen tumsa barbaadan baayʻeen gadi fageenyaan utuu hin adeemsifamin hafu. Rakkoolee kana hubachuudhaan qaamonni dhimmi kun isaan ilaallatu fi Afaan Oromoo afaan teknoolojii gochuuf dharraa qaban, qorannoolee akkasii faana buʻuun sadarkaa isaa hubachuu, tumsi yoo barbaachises tumsa barbaachisu akka godhaniif dhaamsa koo dabarfadha. Keessumattuu biiroleen bulchiinsa mootummaa naannoo Oromiyaa hubannoo gama kanaa qaban kanneen akka *Ejensii ICT Oromiyaa* yunivarsitoota waliin odeeffannoo wal jijjiiruun, yoo dandaʻame ammoo dirree mijataa akka mijeessaniif adaraan jedha.

Qorannoon koo kun qaama NLP taʻee, gita ‚*Information Extraction*ʻ jalatti gosa ‚*Named Entity Recognition*ʻ jedhamu dha. Kaayyoon isaas sooftiweerii barreeffamoota Afaan Oromoo keessaa maqaalee kanneen akka *maqaa namaa*, *maqaa bakkaa*, *maqaa dhaabbilee*, wantoota hamma waan tokkoo ibsan kanneen akka *dhibbeentaa*, *qarshii* fi *yeroo* taʻiintokko itti taʻe hatattamaan hubatuu fi addaan baasuu hojjechuu dha. Qorannoon adeemsise kanaanis yeroo afaanota risoorsii hedduu qaban kanneen akka afaan Ingilizii waliin wal-bira qabamu buʻaa gaarii argadheera.

Maandafroo Laggasee

Table of Contents Page

ACKNOWLEDGEMENTS ..... **I**

DHAAMSA ..... **II**

LIST OF TABLES ..... **VI**

LIST OF FIGURES ..... **VII**

LIST OF ACRONYMS & ABBREVIATIONS ..... **VIII**

ABSTRACT ..... **IX**

CHAPTER ONE: INTRODUCTION **1**

1.1. General Overview 1
1.2. Statement of the Problem 4
1.3. Motivations 4
1.4. Objectives 5
1.4.1 General Objective 5
1.4.2 Specific Objectives 5
1.5. Methodology 5
1.5.1 Review of Literatures 5
1.5.2 Tool Selection 6
1.5.3 Corpus data collection and Development 6
1.5.4 Prototype Development 6
1.5.5 Conducting Experiments 6
1.6. Scope 7
1.7. Application of Results 7
1.8. Thesis Organization 8

CHAPTER TWO: LITERATURE REVIEW **9**

2.1 Natural Language Processing 9
2.2 Information Extraction 11
2.2.1 Architecture of Information Extraction System 13
2.2.2 Types of Information Extraction 13
2.3 Named Entity Recognition (NER) 14
2.3.1 Named Entity types 16
2.3.2 Architecture of NER System 17
2.3.3 Named Entity Recognition Approaches 18
2.3.4 Feature Space for NER 20
2.3.5 Evaluation Metrics for NER 24

2.4 CONDITIONAL RANDOM FIELDS (CRF) 25
2.4.1 The Model 26
2.4.2 Training 28
2.4.3 Inference 30
2.5 AFAN OROMO LANGUAGE 32
2.5.1 Background 32
2.5.2 Writing system of Afan Oromo language 32
2.5.3 Syntax of Afan Oromo sentences 33
2.5.4 Word Categories of Afan Oromo 34
2.5.5 Named Entities in Afan Oromo 36
2.6 SUMMARY 39
CHAPTER THREE: RELATED WORKS 41
3.1 NAMED ENTITY RECOGNITION FOR ENGLISH 41
3.2 NAMED ENTITY RECOGNITION FOR SPANISH 43
3.3 NAMED ENTITY RECOGNITION FOR INDIAN LANGUAGES 45
3.4 NAMED ENTITY RECOGNITION FOR JAPANESE 48
3.5 SUMMARY 51
CHAPTER FOUR: DESIGN OF AONER 53
4.1 THE APPROACH USED 53
4.2 ARCHITECTURE OF AONER 53
4.3 PROCESSES OF AONER 55
4.3.1 Learning Process 55
4.3.2 Prediction Process 56
4.4 PHASES OF AONER 56
4.4.1 Pre-processing Phase 56
4.4.2 Training Phase 58
4.4.3 Recognition Phase 60
4.5 AFAN OROMO POS TAGGER (AOPOST) 62
4.6 SUMMARY 63
CHAPTER FIVE: IMPLEMENTATION OF AONER 64
5.1 CORPUS DEVELOPMENT 64
5.1.1 AONER NE Categories 64
5.1.2 Afan Oromo NE Corpus 65
5.2 PRE-PROCESSING PHASE 67
5.2.1 Parsing 67
5.2.2 Tokenization 68

5.3 TRAINING PHASE 71
5.3.1 BIO Encoding 71
5.3.2 Feature Extraction 72
5.3.3 NE Chunking 76
5.3.4 Building the model 78
5.4 RECOGNITION PHASE 80
5.4.1 Recognition 81
5.5 SUMMARY 82
CHAPTER SIX: EXPERIMENTATION OF AONER 84
6.1 EXPERIMENTATION ENVIRONMENT 84
6.2 EVALUATION METRICS 85
6.3 NATURE AND STATISTICS OF THE TEST DATA 86
6.4 PERFORMANCE OF AOPOST 86
6.5 PERFORMANCE OF AONER 87
6.6 EVALUATION SCENARIOS OF AONER 89
6.6.1 Influence of increasing the training data 89
6.6.2 Influence of Features 90
6.7 DISCUSSION 95
CHAPTER SEVEN: CONCLUSION & FUTURE WORK 96
7.1 CONCLUSIONS 96
7.2 CONTRIBUTION OF THE WORK 98
7.3 FUTURE WORKS 98
REFERENCES 100
APPENDICES 105
APPENDIX A: FEATURE SPACES FOR AONER 105
APPENDIX B: SOME OF AFAN OROMO NUMERALS 107
APPENDIX C: NAME OF DAYS AND MONTHS IN AFAN OROMO 108
APPENDIX D: WORD SHAPE FEATURE VALUES 109
DECLARATION 110

List of Tables Page

TABLE 2.1: NAMED ENTITY TYPES AS DEFINED BY MUC....16
TABLE 2.2: NAMED ENTITY TYPES AS DEFINED BY CONLL....17
TABLE 2.3: WORD-LEVEL FEATURES....22
TABLE 2.4: LIST LOOKUP FEATURES....23
TABLE 2.5: DOCUMENT AND CORPUS FEATURES....23
TABLE 3.1: SUMMARY OF RELATED WORKS....52
TABLE 5.1: AONER"S NE CATEGORIES....65
TABLE 5.2: STATISTICS OF AFAN OROMO NE CORPUS....67
TABLE 5.3: BIO LEGAL TAG SEQUENCE....68
TABLE 5.4: PATTERNS OF TOKENS AS USED BY AONER"S TOKENIZER....70
TABLE 5.5: TOKEN/TAG SEQUENCE AS GENERATED BY BIO ENCODER....72
TABLE 6.1: INFLUENCE OF INCREASING THE TRAINING DATA....89
TABLE 6.2: INFLUENCE OF WORD-SHAPE FEATURE....90
TABLE 6.3: INFLUENCE OF SUFFIX FEATURE....91
TABLE 6.4: INFLUENCE OF PREFIX FEATURE....92
TABLE 6.5: INFLUENCE OF POS FEATURE....93

List of Figures Page

FIGURE 1.1: STANFORD‟S ENGLISH NE RECOGNIZER 3
FIGURE 2.1: INFORMATION FLOW IN NLP....9
FIGURE 2.2: INFORMATION RETRIEVAL VS INFORMATION EXTRACTION WORKS 12
FIGURE 2.3: ARCHITECTURE OF A TYPICAL INFORMATION EXTRACTION SYSTEM 13
FIGURE 2.4: ARCHITECTURE OF A TYPICAL NER SYSTEM 18
FIGURE 4.1: PROPOSED ARCHITECTURE FOR AONER 54
FIGURE 5.2: AONER‟S NE FEATURE EXTRACTION ALGORITHM....76
FIGURE 5.3: AONER‟S NE CHUNKING ALGORITHM....78
FIGURE 6.1: CONFUSION MATRIX OF AOPOST 88
FIGURE 6.2: INFLUENCE OF FEATURES ON RECALL VALUE 94
FIGURE 6.3: INFLUENCE OF FEATURES ON PRECISION VALUE 94
FIGURE 6.4: INFLUENCE OF FEATURES ON F1 VALUE....94

List of Acronyms & Abbreviations

AONER – Afan Oromo Named Entity Recognizer

AOPOST – Afan Oromo POST

BIO – Begin Inside Outside

BOS – Beginning Of Sentence

CoNLL – Conferences on Natural Language Learning

CRF – Conditional Random Fields

EOS – End Of Sentence

FNs – False Negatives

FPs – False Positives

HMM – Hidden Markov Model

IE – Information Extraction

IR – Information Retrieval

IREX - Information Retrieval and Extraction Exercise

MEMM – Maximum Entropy Markov Model

MUC – Message Understanding Conference

NER – Named Entity Recognition

NEs – Named Entities

NLP – Natural Language Processing

POS – Part Of Speech

POST – POS Tagger

SVM – Support Vector Machine

TPs – True Positives

Abstract

This thesis describes the development of Named Entity Recognition (NER) system for Afan Oromo language. NER is an information extraction task aimed at identifying and classifying words of a sentence, a paragraph or a document into predefined categories of NEs. A lot of researches in NER task have been conducted for European and Asian languages, while this work is the first of its kind for Afan Oromo, a language that has the largest native speakers in Ethiopia. A new Afan Oromo NER solution is proposed based on a hybrid approach which contains machine learning and rule based components. Afan Oromo NE corpus of size more than 23,000 words have been developed based on CoNLL‟s 2002, BIO tagging scheme. Four NE categories have been identified and used in the study: person, location, organization and miscellaneous. The miscellaneous category includes date/time, monetary value and percentage. Some of the components in the system include NE Chunker, Feature Extractor and Model Builder. The NE Chunker chunks a sequence of tokens belonging to the same NE category. The feature extractor extracts features from the training data. Position, word-shape, POS, normalization, prefix and suffix of a token were used as features. The model builder estimates the model‟s parameters. Stochastic gradient descent has been used to estimate the model‟s parameter. We have also developed Afan Oromo POST model that can generate POS feature. Evaluation results from our system were promising. We obtained an average performance of Recall 77.41%, Precision 75.80% and F1-measure 76.60% in two major experimentation scenarios: increasing the size of the training data and examining the influence of features. The result from our experiment shows that features play a vital role than increasing the training data size. Examining the influence of features justified that we used the best combination of feature for the development of the system. In general, the algorithms and techniques used in this study obtained good performance when compared to the other resource-rich languages like English.

Keywords: Named Entity Recognition, Named Entities, Conditional Random Fields, Afan Oromo.

Chapter One: Introduction

1.1. General Overview

Language is a medium of communication that enables human beings to exchange ideas and information. In its written form it serves as a means of recording information and knowledge on a long term-basis and transmitting what it records from one generation to the next, while in its spoken form it serves as a means of coordinating our day-to-day life with others (Allen 1996).

A set of symbols and conventions used by human beings for communication purposes is known as natural language (Amanuel 2006). Languages, particularly natural languages, are studied by a discipline known as Linguistics. Nowadays with the existence of computer technologies there is a huge effort going on toward processing natural languages using computers. The academic discipline that studies computer processing of natural language is known as Natural Language Processing (NLP) or Computational Linguistics (Liddy 2001). NLP is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications (Liddy 2001).

The most explanatory method for presenting what actually happens within NLP system is by means of the „levels of language" approach. There are different levels of NLP: phonological (deals with the interpretation of speech sounds within and across words), morphological (deals with the componential nature of words), lexical (deals with the interpretation of the meaning of individual words), syntactic (focuses on analyzing the words in a sentence so as to uncover the grammatical structure of the sentence), semantic (determines the possible meanings of a sentence by focusing on the interactions among word-level meanings in the sentence), discourse (works with units of text longer than a sentence) and pragmatic (uses additional information about the social environment in which a given document exists (Liddy 2001). This study relates to Lexical level of language by focusing on detecting and

classifying the so called Named Entities (NEs) in Afan Oromo[1] texts. NLP is an extremely complicated task. The case is worse for languages like Afan Oromo whose structure is not yet efficiently studied for electronic information exchange and dissemination. For computers to understand natural languages, they should be made to handle variants of the same basic word form together with the unique meanings and the specific interpretations that each form has. This further complicates the task for computers to understand natural languages.

Named Entity Recognition (NER) is the process of identifying and classifying words of a sentence, a paragraph or a document into predefined categories of NEs. In the taxonomy of computational linguistics tasks, it falls under the domain of "Information Extraction", which extracts specific kinds of information from documents as opposed to the more general task of "document management" which seeks to extract all of the information found in a document (Zhou & Su 2002). NER performs what is known as surface parsing, delimiting sequences of tokens that answer the questions like "who", "where" and "how much" in a sentence (Zhou & Su 2002). For instance, a simple news named-entity recognizer for English might find the person name *Angela Merkel* and the location name *Germany* in the text *Angela Merkel is the chancellor of Germany*.

The term “Named Entity”, now widely used in NLP, was coined for the Sixth Message Understanding Conference (MUC-6) (Grishman & Sundheim 1996). At that time, MUC was focusing on information extraction tasks where structured information of company activities and defence related activities is extracted from unstructured text, such as newspaper articles. In defining the task, people noticed that it is essential to recognize information units like names, including person, organization and location names, and numeric expressions including time, date, money and percent expressions (Zhang & Johnson 2003). NER has become more important nowadays due to the large amount of available electronic text, which makes it necessary to build systems that can automatically process and extract information from text (Zhou & Su 2002). Figure 1.1 shows a screen shot of Stanford University‟s English NER system.

[1] The language of Oromo is written in different literatures as Afaan Oromoo, Afan Oromo, Oromiffa or Oromo. For this study, however, we adopt *Afan Oromo* to refer to Oromo People‟s language.

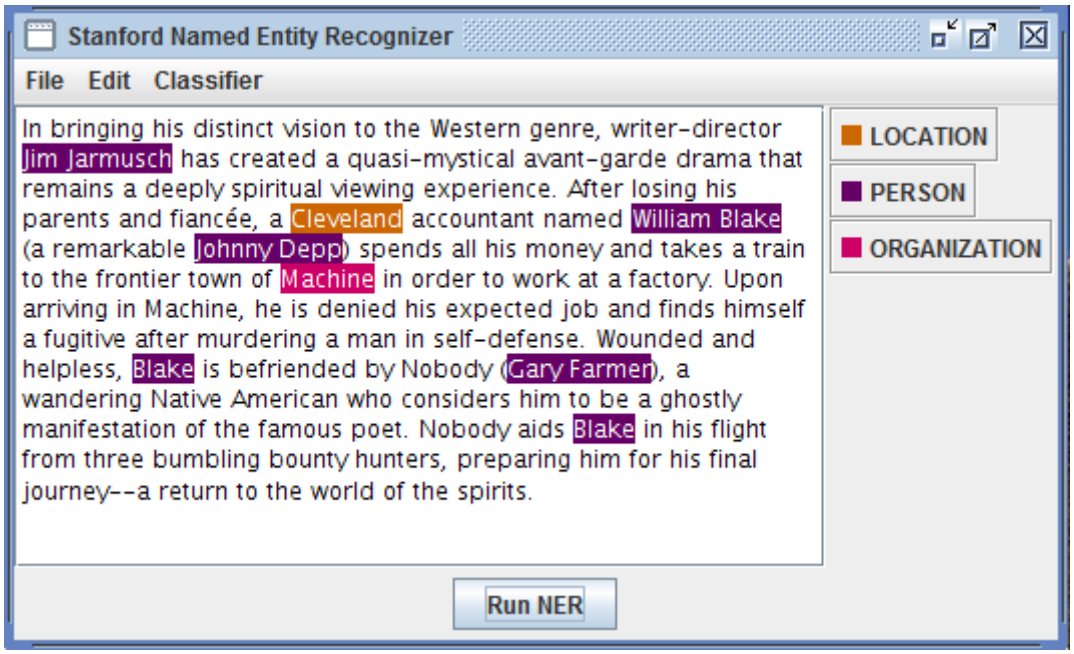

Figure 1.1: Stanford's English NE Recognizer

NER systems are created using two approaches: linguistic grammar-based (also called Rule-based) techniques and statistical (also called Machine Learning). Grammar-based systems focus on extracting names using lots of human-made rules sets. The systems consist of a set of patterns using grammatical (e.g. part of speech), syntactic (e.g. word precedence) and orthographic features (e.g. capitalization) in combination with dictionaries (Budi & Bressan 2003). Grammar-based systems typically obtain better results, but at the cost of months of work by experienced computational linguists though they lack the ability of portability and robustness, and are furthermore the high cost of the rule maintenance increases whenever the data is slightly changed. On the other hand, statistical NER systems typically require a large amount of manually annotated training data and are porTable, adapTable and easily maintainable.

Afan Oromo (when translated it means *Oromo Language*) is one of the major Languages that is widely spoken and used in Ethiopia (Abarra 1988). As (Amanuel 2006) puts it, it is grouped in Low Land East Cushitic within the Cushitic family of the Afro-Asiatic phylum. In this Cushitic branch of the Afro-asiatic language family Afan Oromo is considered as one of the most extensive languages among the forty or so Cushitic languages (Amanuel 2006). It

is a mother tongue for Oromo people, who are the largest ethnic group in Ethiopia comprising 33.5% (25.5 Million) of the population according to the 2008 census. With regard to the writing system, „Qubee" (Latin-based alphabet) has been adopted and become the official script of Afan Oromo since 1991. Currently, it is an official language of the regional state of Oromia (which is the largest regional state in Ethiopia). It is widely used as both written and spoken language in Ethiopia and some neighbouring countries, including Kenya and Somalia (Amanuel 2006).

1.2. Statement of the Problem

The identification and classification of NEs in plain text is of key importance in numerous NLP applications. In information extraction systems, NEs generally carry important information about the text itself, and thus are targets for extraction. Since entity names form the main content of a document, NER is a very important step toward more intelligent information extraction and management (Zhou & Su 2002). The task has also important significance in the Internet search engines and is an important task in many of the language engineering applications such as Machine Translation, Question-Answering systems, Indexing for Information Retrieval and Automatic Summarization (Srikanth & Murthy 2008). These facts show that NER system is the basic and relevant component for the development of most of NLP related applications. But Afan Oromo does not have NER system so far. Thus, the development of such a system for Afan Oromo is essential for easing the development of Afan Oromo related NLP applications and that is what this study is intended for. As far as the knowledge of the researcher is concerned no identical study was conducted so far for the language.

1.3. Motivations

The aforementioned facts and Figures show that Afan Oromo is a widely spoken language in the horn of Africa and it actually has a rich morphology. With the fact that the language is used in offices, schools and media, there is a huge electronic data available that encourages studies related to NLP tasks associated with the language. So, the development of NLP applications for this language is required to cope up with the current technologies of NLP. Besides these, this study is inspired by the contribution of NER tasks to other NLP studies.

1.4. Objectives

The objective of the study is described as general objective and specific objectives.

1.4.1 General Objective

The general objective of the study is to develop Afan Oromo NER system.

1.4.2 Specific Objectives

1. Study the structure of Afan Oromo language and the linguistic patterns that refer to NEs.
2. Study applications dealing with NER.
3. Study the architectures, approaches and features used for the development of NER systems.
4. Investigate and understand the nature of a CRF model.
5. Review related studies that are conducted so far both for Afan Oromo and other languages.
6. Propose and design Afan Oromo NER system.
7. Train the proposed NER system with Afan Oromo texts.
8. Develop a prototype of the proposed Afan Oromo NER system.
9. Test the performance of the developed prototype with different parameters.

1.5. Methodology

A number of methods (techniques) are employed for the successful completion of this study. Some of them are discussed below.

1.5.1 Review of Literatures

A number of related works and resources have been reviewed. This consists of conference and journal articles, white papers and NER systems developed for other languages. The large portions of reviewed materials are conference and journal articles. The nature, operational background and performance of NER systems like **Stanford-ner**, **MENE** and **LingPipe's neDemo** are also studied. In addition, a discussion was made with Afan Oromo Linguistic

experts regarding the linguistic nature of the language like the grammatical structure and the properties of Afan Oromo named entities.

1.5.2 Tool Selection

For the development of Afan Oromo NER prototype we used Java programming language as a backbone. For the NER part, LingPipe library is used. Since we used part-of-speech tag of words as a feature for NER, we also developed our own part-of-speech tagger (see section 4.5) on LingPipe that can easily be integrated into our system.

1.5.3 Corpus data collection and Development

Afan Oromo does not have publicly available annotated corpus text for any NLP task including NER so far. As a result, we collected an electronic text (a total of 4014 articles) from two state owned news papers: „*Bariisaa*“ and „*Kallacha Oromiyaa*“. The collected articles were further edited manually by removing those sentences which does not have NEs at all. The resulting data is then tokenized and tagged manually in accordance with the CoNNL 2002 standard. Finally, NE corpus of size more than 23,000 tokens out of which 3,575 are NEs was developed. Since we used part-of-speech tagger (POST) in our system, for feature extraction, we did additional work of developing a corpus for training the POST. Accordingly, we developed an additional corpus of size 4588 words with each word tagged with their part-of-speech.

1.5.4 Prototype Development

A prototype was developed in order to check whether our study works in accordance with the ideas and theories of NER.

1.5.5 Conducting Experiments

After the formal system was being developed, it was tested and evaluated with different parameters by performing an objective testing. The performances obtained throughout the conducted experiments were given in three evaluation metrics: Precision, Recall and F-measure.

1.6. Scope

There are different types of NEs that an NER system can be designed to recognize. Our study focused on the recognition of four NEs from news text that include:

1. *Person Name (PER)* – name of a person including his/her father and grandfather.
2. *Location name (LOC)* – name of geographical entities like cities, regions, country.
3. *Organization name (ORG)* – name of entities like companies, institutions and organizations.
4. *Miscellaneous (MISC)* – collects four numeric entities: Date, Time, Monetary values and Percentage.

1.7. Application of Results

NER is an important tool in almost all NLP application areas. Proper identification and classification of named entities are very crucial and pose a very big challenge to the NLP researchers (Ekbal et al. 2008). Named entity recognition software serves as an important preprocessing tool for tasks such as information extraction, information retrieval and other text processing applications. Based on these facts, Afan Oromo NER system plays a crucial role in information extraction based researches and applications associated to the language. It also opens the track for future Afan Oromo NLP studies. It also simplifies development of the following applications for Afan Oromo.

- Search Engines (Semantic based)
- Machine Translation
- Question-Answering Systems
- Indexing for Information Retrieval
- Automatic Summarization

1.8. Thesis Organization

The rest of this paper is organized as follows. Chapter 2 is a review of literatures. Before discussing NER, the chapter starts with an overview about NLP (Natural Language Processing) and IE (Information Extraction). The chapter is then devoted to the discussion of NER at large with specifics on approaches used in NER, architecture of NER systems, evaluation metrics and feature spaces for NER systems. CRF algorithm is also discussed in this chapter. In addition, an overview and language specific topics regarding Afan Oromo are also discussed in this chapter. Chapter 3 contains related works done so far for different languages in the arena of NER task. In this chapter a great focus is given to the approaches followed, the algorithm and features used, and the performance of those systems.

Chapter 4 discusses the architecture and design of our system, Afan Oromo NER. The architectural components of our system are briefly discussed in this chapter. Chapter 5 is concerned with the implementation issues of our system. The techniques, methods and algorithms we followed toward the development of our system are discussed clearly in this chapter. Chapter 7 contains topics concerned with the evaluation of the performance of our system with different parameters. Chapter 8 concludes our study by outlining the benefits gained from our research work. It also gives an insight to the expected future works.

Chapter Two: Literature Review

2.1 Natural Language Processing

In our time of information age, inspired by the application of computers, vast number of disciplines are inheriting the ideology of computers to help them carry-out detail studies in their domain. The use of computers even opened a track for the inauguration of new fields. One of them, resulted from the combination of computer science, linguistics and cognitive psychology disciplines, is Natural Language Processing (NLP). As a result of this, NLP is given different names like speech and language processing, human language technology, computational linguistics, and speech recognition and synthesis (Jurafsky & Martin 2000).

NLP is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications (Liddy 2001). In NLP, the contribution of a Computer Science discipline is to develop an algorithm that specifies the internal representation of data and define the way they can be efficiently processed in a computer. Figure 2.1 shows the flow of information in NLP (Bose 2004).

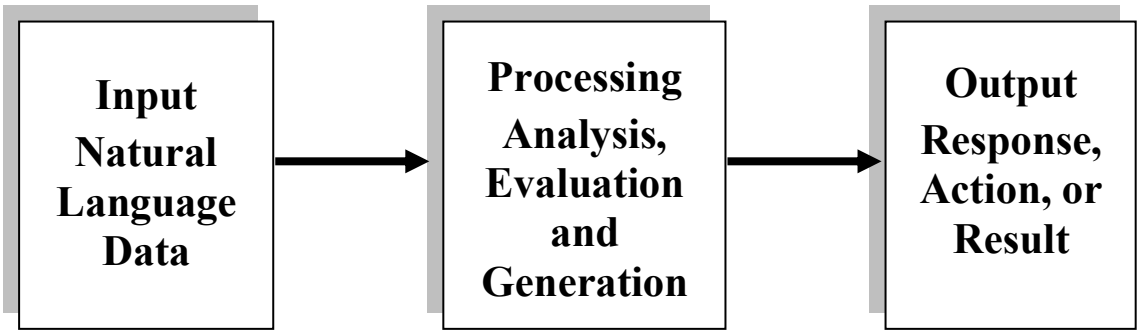

Figure 2.1: Information flow in NLP

Basically, engaging in complex language behaviour requires various kinds of knowledge of language (Jurafsky & Martin 2000, Bose 2004). This language knowledge is also called, as in (Liddy 2001), „levels of language approach". NLP must possess considerable knowledge

about the language under consideration. Naturally, languages contain a number of ambiguities at one or more of the above levels. One can say that input is ambiguous if there are alternative linguistic structures that can be built for it and ambiguity problem is resolved via disambiguation (Bose 2004).

The idea of giving computers the ability to process human language is as old as the idea of computers themselves. Research in natural language processing has been going on for several decades dating back to the late 1940s (Liddy 2001). During those times, it was a difficult task, even though not impossible, for it is based on human made rules. Preparing rules requires extensive involvement of talented linguistic experts and it is a time consuming work. In recent years researches in the area of NLP are increasing rapidly.

Among the main reasons behind the scene are (Liddy 2001):

- An increased availability of large amounts of electronic text
- Availability of computers with increased speed and
- The advent of the Internet

These attributes shifted NLP from rule-based approach to machine-learning approaches. The existence of machine-learning approaches resulted in an open and comforTable environment that encourages the development of different NLP based systems. So, NLP has become an applied, rather than a theoretical science.

With over sixty years, natural language systems are still very complicated to design and they are still not perfect because human language is complex, and it is difficult to capture the entire linguistic knowledge for hundred percent accuracy in processing (Bose 2004). However, NLP technologies are becoming extremely vital in order to ease access into systems, thereby making those systems more user-friendly. As (Bose 2004) puts it, increasing number of companies are investing in and deploying voice or speech recognition and processing technologies at an alarming rate to save money by replacing operators and to improve service to their customers. For example, Windows® Vista® has come up with facilities like Windows Speech Recognition enabling us to address our computer as we were

addressing another person. In the future, NLP is expected to further improve existing systems and hope that we will see technologically intelligent and more user-friendly systems.

NLP provides a good baseline for both theoretical and implementation of a range of applications. In fact, according to (Liddy 2001), any application associated with text is a candidate for NLP. This includes information extraction, text summarization, machine translation, speech recognition and etc. As our work is a related to Information Extraction we give an overview of IE in the next section.

2.2 Information Extraction

Information Extraction (IE) is the name given to any process which selectively structures and combines data which is found, explicitly stated or implied, in one or more texts (Cowie & Lehnert 1996). It involves picking out specified types of information from natural language text. IE has a vital role in evaluating and comparing different NLP technologies. It is an important task in NLP with many practical applications.

Researches in the area of IE are promoted and evaluated by a conference called Message Understanding Conference (MUC). The MUCs were initiated by NOSC to assess and foster research on the automated analysis of military messages containing textual information (Grishman & Sundheim 1996). Although called “conferences”, MUCs are characterized by the evaluations the participants submit in order to be eligible to attend the conferences. Broadly one can say that the IE field grew very rapidly when ARPA, the US defense agency, funded competing research groups to pursue IE, based initially on scenarios like the MUC-4 terrorism events (Cowie & Lehnert 1996). In a very real sense, DARPA created the field of IE, in part by focusing on a certain kind of task (Applet & Israel 1999). As opposed to in-depth natural language processing, IE is a more focused and goal oriented task (Rilof 1993). For example, the MUC-4 task was at extracting information about terrorist events, such as the names of perpetrators, victims and instruments.

The clear boundary between IE and Information Retrieval (IR) cannot be drawn in terms of the complexity of the language features. The best way to characterize the two, as in (Applet & Israel 1999), is in terms of functionality. The task of IR is to search and retrieve documents in response to queries for information. But, IE extracts specific kinds of information from documents while IR seeks to extract all of the information found in a document (Zhou & Su 2002). So, IE tends to be more specific while IR is more general like search engines of our time such as Google®. Figure 2.2, obtained from (GATE 2009), depict the difference between the two.

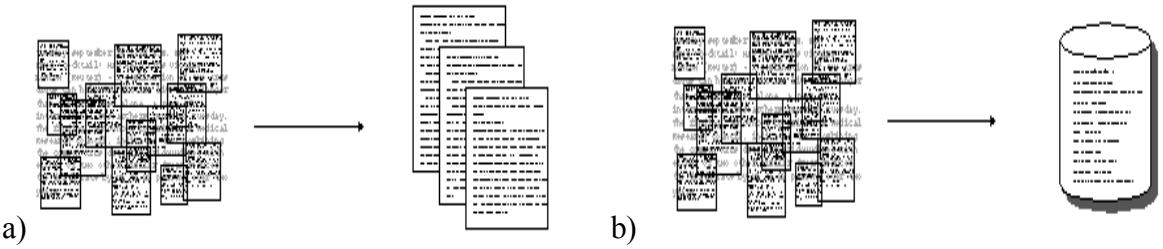

Figure 2.2: showing how a) information retrieval b) information extraction works

Figure 2.2 illustrates the fact that IR is associated with analysing the documents while IE is associated with analysing the facts. Naturally, any technology associated with IE has to be preceded by an IR phase and IE can be considered as a processed IR.

IE was a tedious task in early days of 1980s for it is based on hand-crafted rules. During those times, the development of IE systems requires the collaboration of linguistic experts in order to write rules that tell a system how to do a task. But nowadays, with the advent of World Wide Web and the existence of large electronic documents, there is an encouraging environment for the development of IE and other NLP systems using machine learning approach. This enabled researchers to use electronic texts and train machines, so that machines can perform near to human performance. This refers to *machine learning*. However, still IE is a difficult task. The main reason for this is the fact that most electronic texts are naturally unstructured lacking well-formedness.

2.2.1 Architecture of Information Extraction System

IE is a domain specific task. This means, an IE system developed and working effectively for one domain cannot perform with the same accuracy and efficiency on another domain or may not work at all. For example, IE developed for terrorist event cannot work for housing advertisement domain. However, there are core elements that are shared by nearly every extraction system, regardless of whether it is designed according to the Rule-based or machine-learning paradigm. Figure 2.3 illustrates the components shared by IE systems (Applet & Israel 1999).

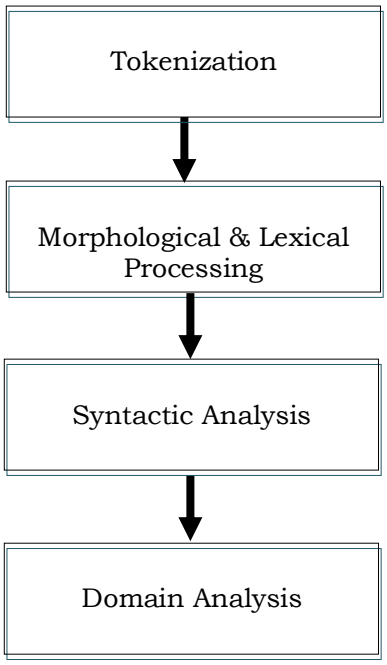

Figure 2.3: Architecture of a typical Information Extraction System

2.2.2 Types of Information Extraction

MUC was the first to provide the formal classification of types of IE task in 1998 (Cunningham 2005). It defined an IE task by splitting it into five tasks:

- **Named Entity Recognition (NER)**: Finds and classifies names, places, etc. It is all about finding entities.
- **Coreference resolution (CO)**: Identifies identity relations between entities. It is concerned with entities and references (such as pronouns) that refer to the same thing.
- **Template Element construction (TE)**: Adds descriptive information to NER results (using CO). i.e what attributes entities have.

- **Template Relation construction (TR)**: Finds relations between TE entities. It is all about what relationships between entities there are.
- **Scenario Template production (ST)**: Fits TE and TR results into specified event scenarios. Events that the entities participate in.

This study is concerned with how named entity recognition can be developed for Afan Oromo. The next sections are dedicated to the detail discussion of the concepts in named entity recognition.

2.3 Named Entity Recognition (NER)

In Information Extraction systems, accurate detection and classification of named entities is a very important task because they help us extract knowledge from texts such as where the event happened, who were involved and when it happened (Solorio 2004).

Named Entity (NE) Recognition (NER), the basic component of IE (Razvan 2007), is aimed at classifying every word in a document into some predefined categories. It was introduced during the sixth Message Understanding Conference in 1996 (Grishman & Sundheim 1996). The core goal of NER is to locate the entities (things) in natural language text and specifies their type. It involves finding Named Entities, such as names of organizations, persons, locations, time expressions, and numeric expressions such as money and percentage expressions from structured and unstructured documents (Sekine & Eriguchi 2000, Sang & Meulder 2003). Putting it in another way, (Z. Kozareva 2006) describes NER as an automatic information extraction concerning particular person, location, organization, title of movie or book. While doing so, it provides answers to some of the „**wh**" questions like "who", "where", "when" and "how much" in a sentence. Generally speaking, NER performs what is known as surface parsing, delimiting sequences of tokens that answer these important questions (Zhou & Su 2002).

The above sentences rotate around the fact that NER task is concerned with the extraction of the so called *Named Entities (NEs)*. According to (Sang & Meulder 2003), NEs are phrases that contain the names of persons, organizations and locations. In the expression *named*

entity, the word *named* restricts the task to those entities for which one or many rigid designators, as defined by Kripke, stands for the referent.

A NER task can be described as composition of two subtasks, *entity detection* and *entity classification* (Kozareva 2006**,** Kozareva et al. 2005). The entity detection subtask is concerned with identifying an entity among other words in a sentence. It determines the boundaries of the entity (e.g. the place from where it starts and the place it finishes). For detection reason mostly a scheme known as BIO is used (Razvan 2007, Solorio 2004). With this scheme, each word sequence is tagged with one of three tags, beginning of NE (B), inside the NE (I) or outside (O). Named entities are extracted by doing token classification and then assembling maximally contiguous sequences of **B** and **I** tokens. This is important for tracing entities composed of two or more words such as "Baankii Intarnaashinaalii Oromiyaa" (Oromia International Bank), "Letnaanti Koloneel Abdiisaa Aagaa" (Lieutenant Colonel Abdisa Aga). Once all entities in the text are detected, they are passed for classification in a predefined set of categories such as person, location, organization or others. This task is known as entity classification. The classification is done by classifiers based on some patterns and a set of features. The final NER performance is measured considering the entity detection and classification tasks together.

In NER task, proper identification and classification of named entities are very crucial and pose a very big challenge to the NLP researchers. The challenge comes from a number of reasons (Mikheev et al. 1999a). First, the definition of what is and is not a Named Entity (NE) can be very complex. A second difficulty is that it is important to tag exactly the right words. The entire string "Arthur Andersen Consulting" should be marked as an ORGANIZATION; one should not mark the substring "Arthur Andersen" as a PERSON. Again, this may appear ad hoc and the definition of how much should be marked up will be needed by the application. But for any application, consistency of NE markup is crucial. The third and biggest problem is that named entities are expressed with words which can refer to many other things. One might think that NER could be done by using lists of names of people, places and organizations, but that is not the case. To begin with, the lists would be huge: it is estimated that there are 1.5 million unique surnames just in the U.S. (Spiegel 1985). It is not feasible to list all possible surnames in the world in a NER system. These

levels of ambiguities in NER make it difficult to attain human performance (Ekbal et al. 2008).

2.3.1 Named Entity types

Named Entities (NEs) play a central role in conveying important domain specific information in text, and good named entity recognizers are often required in building practical information extraction systems. There are no general types of NE that are commonly used across all languages. As a result of this, the number and type of NEs an NER system can recognize differs from language to language or from domainto domain. This feature is quite variable due to the ambiguity in the use of the term *Named Entity* depending on the different forums or events. There are conferences or contests that are organized to define the types of NEs and evaluate the performance of a given NER system developed for a language. The conferences are, for example, Message Understanding Conference (MUC) for English (Sassano & Utsuro 2000), Conferences on Natural Language Learning (CoNLL), a language independent NER task (Erik 2002) and Information Retrieval and Extraction Exercise (IREX) for Japanese (Sekine & Eriguchi 2000). MUC played a crucial role to the emergence and fostering of researches in the area of NER. These contests had defined the NE types for their respective domain and the corresponding NE types for MUC and CoNLL are shown in Table 2.1 & 2.2 respectively.

Table 2.1: Named entity types as defined by MUC

Named Entity	Example
PERSON	Smith, Obama, Clinton
ORGANIZATION	IBM, LIFAN Motors
LOCATION	Texas, Cape Town
DATE	25/02/2010, January 15
TIME	8:30 AM
PERCENTAGE	10%
MONETARY AMOUNT	$120.00, €250

Table 2.2: Named entity types as defined by CoNLL

Named Entity	Example
PERSON	Del Bosque, Van Bastan
ORGANIZATION	UPC, Nuffic, Royal Netherlands
LOCATION	Twente, Barcelona
MISCELLANIOUS	46 Euro, 19:00 GMT

2.3.2 Architecture of NER System

A typical named entity recognizer has four core elements regardless of whether it is designed according to rule-based approach or automatic machine learning approach. The architecture of a typical NER system is shown Figure 2.4. The core elements are Tokenization, Morphological and Lexical processing, Identification and Classification (Applet & Israel 1999**,** Louis et al. 2006). Tokenization is the first step in interpreting text by splitting up a string of words/characters (comprising a document, paragraph or sentence) into minimal parts of structured text that are useful to be used as a unit, referred to as a token (Louis et al. 2006). With regard to NER, tokenization can consist of sentence splitting and word segmentation as a subtask. After tokenization process is completed, morphological and lexical processing proceeds. During this step, word tokens in a document are sequentially tagged as being **I**nside or **O**utside of a given named entity. It mainly employs part-of-speech tagger (Applet & Israel 1999) and each word in a sequence of words is labeled with an **I**nside or **O**utside tag (Razvan 2007). In addition, it employs components like NP chunking and feature extraction. The morphological and lexical processing mainly helps for the detection of NEs which is depicted in the following Figure as identification. The identification component detects NEs with the aid of stored models or rules, based on the approach used. The detected NEs are then ready to be classified into their respective classes. The classification step takes the detected NEs and categorizes them into their corresponding categories. The classification is done by a classifier.

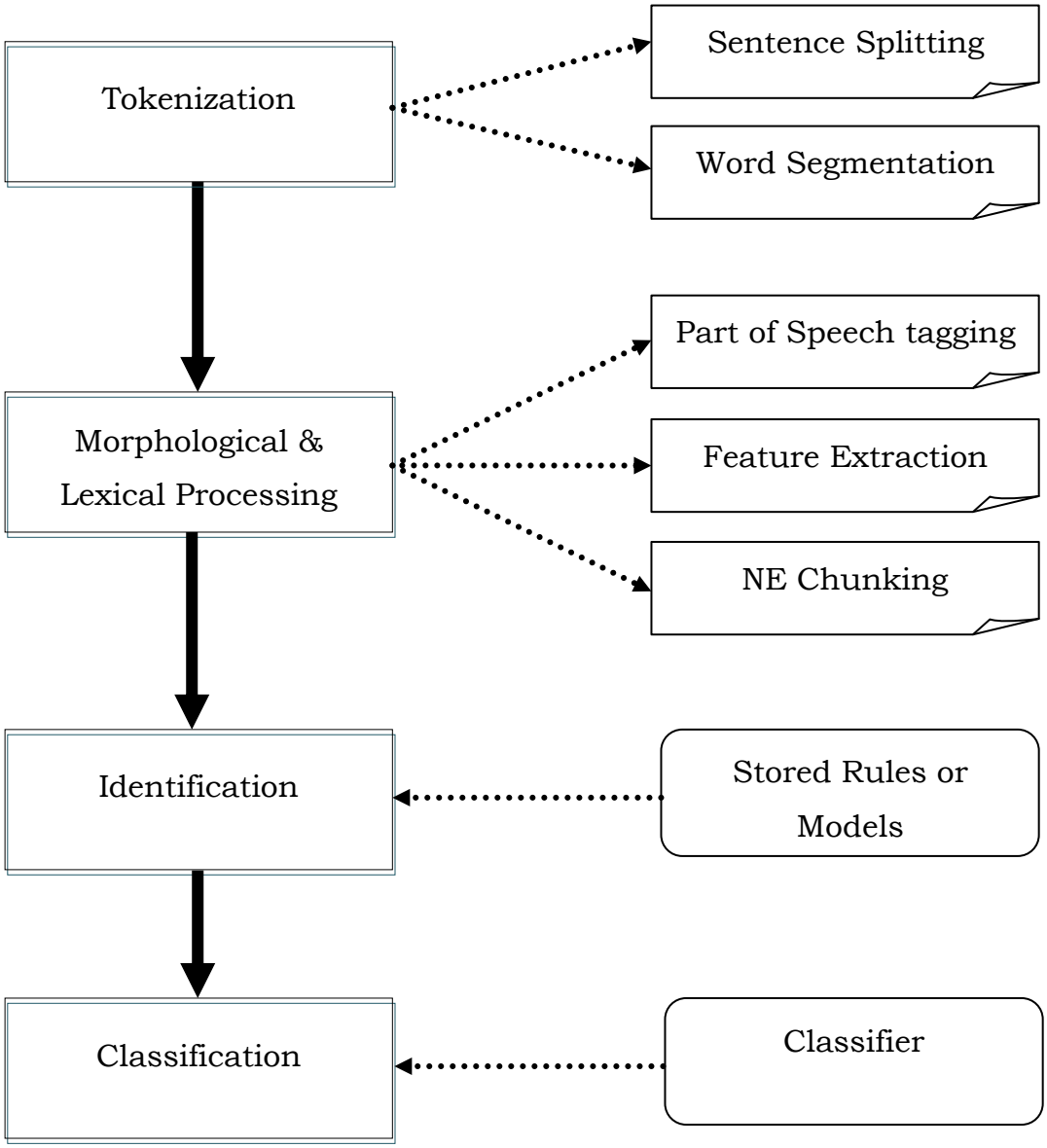

Figure 2.4: Architecture of a typical NER system

2.3.3 Named Entity Recognition Approaches

In recent years the development of an automatic named entity recognizer is gaining warm acceptance and is becoming a popular research area. NER systems developed so far, for different languages, are based on one of the three approaches (Wu et al. 2006, Mansouri et al. 2008): Rule-based, Machine Learning-based and Hybrid approaches.

Rule-based approach

Hand-made Rule-based NER systems extract names entities using lots of human made rule sets. Generally, the systems consist of a set of patterns using grammatical (e.g. part of speech), syntactic (e.g. word precedence) and orthographic features (e.g. capitalization) in combination with dictionaries (Mansouri et al. 2008). An example for this type of system is: "Atileeti Daraartuu Tulluun Boqqojjiitti dhalatte." (Athlete Derartu Tulu was born in Bokoji.) In this example a proper noun (Daraartuu Tulluu) follows a person's title (Atileeti), and then the proper noun that is started with capital letter and ends with suffix (-tti) (Boqqojjii) is a location's name. These approaches are relying on manually coded rules. These kinds of models have better results for restricted domains, are capable of detecting complex entities that learning models have difficulty with. However, the rule-based NE systems lack the ability of portability and robustness, and furthermore the high cost of the maintenance of the rule increases even when the data is slightly changed. These type of approaches are often domain and language specific and do not necessarily adapt well to new domains and languages.

Machine learning approach

The purpose of this NER approach is converting identification problem into a classification problem and employs a classification statistical model to solve it (Mansouri et al. 2008). It requires a set of annotated texts from which the system looks for patterns and relationships to make a model. The model helps it to identify and classify named entities into a particular class using machine learning algorithms. There are three types of machine learning model that are in use for NER (Nadeau & Sekine 2007): Supervised, Semi-supervised and Unsupervised machine learning model. Supervised learning involves using a program that can learn to classify a given set of labeled examples that are made up of the same number of features (Mansouri et al. 2008). The name supervised is given from the fact that the people who annotated the training example are supervising the program about the clear differences among the texts. Supervised techniques include (Nadeau & Sekine 2007) Hidden Markov Model (HMM), Maximum Entropy Models (MEM), Support Vector Machines (SVM) and Conditional Random Fields (CRF). Semi-supervised, which is relatively recent, involves a small degree of supervision, such as a set of seeds, for starting the learning process (Nadeau

2007). The main idea behind this technique is to give the system a hint or a clue and make it to recognize those hint words in different contexts of different sentences. By increasing the number of clues the scope of the system can be increased. In unsupervised learning, the goal of the program is to build representations from data that can then be used for data compression, classifying, decision making and other purposes (Mansouri et al. 2008). Unsupervised learning is not a very popular approach for NER and it basically relies on lexical resources (e.g.,WordNet), on lexical patterns and on statistics computed on a large unannotated corpus (Nadeau & Sekine 2007). In general, unlike rule-based ones, machine learning based systems are porTable, robust and easy to maintain while requiring large amount of annotated training data (Mansouri et al. 2008).

Hybrid approach

This approach, as the name implies, is the result of the combination of rule-based and machine-learning approaches. It uses new methods using strongest points from each method (Mansouri et al. 2008). Although this type of approach can get better result than some other approaches (Mansouri et al. 2008), the weakness of handcraft Rule-base NER remains the same when there is a need to change the domain of data.

2.3.4 Feature Space for NER

For a human being, identifying in a text which parts correspond to NEs can be trivial by considering the word itself. However, this cannot be the case for programs designed to detect and classify NEs. The challenge is that NE expressions are hard to analyze because they belong to the open class of expressions (Ekbal et al. 2008), i.e., there is an infinite variety and new expressions are constantly being invented. (Solorio 2004) elaborates this idea as:

> "*The common difficult problem to all of natural language processing tasks is: ambiguity. Another inconvenience is that documents are not uniform, their writing style, as well as the vocabulary can change dramatically from one document to another*."

For this reason, NER systems require evidence as to how correctly recognize and classify NEs. As defined in (McDonald 1996), there are two kinds of evidences that can be used in

NER to solve the ambiguity, robustness and portability problems described above. The first is the internal evidence found within the word and/or word string itself while the second is the external evidence gathered from its context. These evidences are collectively referred to as NE features.

NE features are describers or characteristic attributes of words designed for algorithmic consumption (Nadeau 2007). In NER, useful features include neighbouring words and word bigrams, prefixes and suffixes, capitalization, membership in domain-specific lexicons, and semantic information from sources such as WordNet **(**Sutton & McCallum 2006**)**.

(Nadeau & Sekine 2007) present the features most often used for the recognition and classification of named entities by describing them along three different axes: Word-level features, List lookup features and Document and corpus features.

2.3.4.1 Word-level features

Word-level features are related to the character makeup of words. They specifically describe word case, punctuation, numerical value and special characters. Table 2.3 lists subcategories of word-level features.

Table 2.3: Word-level features

Features	Examples
Case	- Starts with a capital letter - Word is all uppercased - The word is mixed case (e.g., ProSys, eBay)
Punctuation	- Ends with period, has internal period (e.g., St., I.B.M.) - Internal apostrophe, hyphen or ampersand (e.g., O"Connor)
Digit	- Digit pattern (e.g., date, percentage, interval) - Cardinal and Ordinal - Roman number - Word with digits (e.g., W3C, 3M)
Character	- Possessive mark, first person pronoun - Greek letters
Morphology	- Prefix, suffix, singular version, stem - Common ending (e.g., "soft" in *Microsoft*, *Cybersoft*)
Part-of-speech	- proper name, verb, noun, foreign word
Function	- Alpha, non-alpha, n-gram[2] - lowercase, uppercase version - pattern, summarized pattern[3] - token length, phrase length

2.3.4.2 List lookup features

Lists are the privileged features in NER. The terms "gazetteer" (Louis et al. 2006), "lexicon" (Asahara & Matsumoto 2003) and "dictionary" (Srikanth & Murthy 2008) are often used interchangeably with the term "list". List inclusion is a way to express the relation "is a" (e.g., *Nekemte is a town*). It may appear obvious that if a word (*Nekemte*) is an element of a list of towns, then the probability of this word to be town, in a given text, is high. Table 2.4 lists subcategories of List lookup features.

[2] A feature can be created by isolating the non-alphabetic characters of a word (e.g., nonalpha(A.T.&T.) = ..&.)

[3] a pattern feature might map all uppercase letters to "A", all lowercase letters to "a", all digits to "0" and all punctuation to "-":x = "G.M.": GetPattern(x) = "A-A-"

Table 2.4: List lookup features

Features	Examples
General List	- General dictionary[4] - Stop words (function words) - Capitalized nouns (e.g., January, Monday) - Common abbreviations
List of entities	- Organization, government, airline, educational - First name, last name, celebrity - Astral body, continent, country, state, city
List of entity cues	- Typical words in organization. (e.g., "associates") - Person title, name prefix, post-nominal letters - Location typical word, cardinal point

2.3.4.3 Document and Corpus features

Document features are defined over both document content and document structure. Large collections of documents (corpora) are also excellent sources of features.

Table 2.5: Document and corpus features

Features	Examples
Multiple occurrences	- Other entities in the context - Uppercased and lowercased occurrences - Anaphora, coreference
Local syntax	- Enumeration, apposition - Position in sentence, in paragraph, and in document
Meta information	- Uri, Email header, XML section - Bulleted/numbered lists, Tables, Figures
Corpus frequency	- Word and phrase frequency - Co-occurrences - Multiword unit permanency

[4] Common nouns listed in a dictionary are useful, for instance, in the disambiguation of capitalized words in ambiguous positions (e.g., sentence beginning)

2.3.5 Evaluation Metrics for NER

The performance of a NER system is measured basically, in order to compare it with human performance. It employs a means of comparing the output of the system against annotated data by trained language analysts. (Mansouri et al. 2008) explain the evaluation task associated with NER as:

> "*The task requires that the system recognizes what a string represents, not just its superficial appearance. Sometimes, the right answer is superficially apparent and can be obtained by local pattern matching techniques. In other cases, the right answer is not superficially apparent, as when a single capitalized word could represent the name of a location, person, or organization, and the answer may have to be obtained using techniques that draw information from a larger context or from reference lists.*"

Even though, there are a different number of contests or conferences organized to evaluate the performance of NER systems, their core method of evaluation and metrics are the same. The performance is evaluated using 3 metrics: Precision (P), Recall (R) and F-measure (F) and they are represented in percentage form (Mansouri et al. 2008, Nadeau 2007 and Razvan 2007). The general formula for each metrics is as follows:

$$P = \frac{Number\ of\ Correct\ Responses}{Number\ of\ Responses}$$

$$R = \frac{Number\ of\ Correct\ Responses}{Number\ of\ Correct\ in\ Key}$$

$$F = \frac{PR}{\frac{1}{2}(P+R)}\ which\ is\ mathematically\ equivalent\ to\ \frac{2PR}{P+R}$$

Precision measures the percentage of correctly extracted relations (number of correct answers) out of the total number of relations extracted (number of actual system guesses). Recall measures the percentage of correctly extracted relations (number of correct answers) out of the total number of relations annotated in the corpus (number of possible entities in the solution). F-measure is the harmonic mean of precision and recall.

2.4 Conditional Random Fields (CRF)

Nowadays most of NLP studies are conducted using machine learning approach. Machine learning approach, the supervised one, makes use of common algorithms like HMM, MEMM and CRF. Since our study is conducted and the model is trained using CRF, we will give an overview of CRF in this section. In what follows, X is a random variable over data sequences to be labeled, and Y is a random variable over corresponding label sequences. In NER task X represents the sequence of words we are going to label and Y represents the NE label, that is, one of the entity types Person, Location, Organization, and Other.

Modelling a relational data can be done through either generative or discriminative modeling. Generative modelling models the joint distribution p(y,x) where the variable y represents the attributes of the entities that we wish to predict, and the input variable x represents our observed knowledge about the entities. According to (Sutton & McCallum 2006), modeling the joint distribution can lead to difficulties when using the rich local features that can occur in relational data, because it requires modeling the distribution p(x), which can include complex dependencies. HMM is an example of generative modeling. Discriminative modeling models a conditional distribution p(y|x) which does not include a model of p(x). For this reason, dependencies among the input variables x do not need to be explicitly represented. The conditional probability of the label sequence can depend on arbitrary, non independent features of the observation sequence without forcing the model to account for the distribution of those dependencies (Lafferty et al. 2001). CRF and MEMM are based on discriminative modeling.

The following point, taken from (Sutton & McCallum 2006), can best justify the suitability of modeling the conditional distribution over modeling the joint distribution for classification based tasks like NER:

Relational data has two characteristics: first, statistical dependencies exist between the entities we wish to model, and second, each entity often has a rich set of features that can aid classification. Modeling the joint distribution can lead to difficulties when using the rich local features that can occur in relational data, because it requires modeling the distribution p(x), which can include complex dependencies. Modeling these dependencies among inputs can lead to intracTable models, but ignoring them can lead to reduced performance. A solution to this problem is to directly model the conditional distribution p(y|x), which is sufficient for classification.

MEMM is a victim of the so called label bias problem in which the transitions leaving a given state compete only against each other, rather than against all other transitions in the model. In probabilistic terms, transition scores are the conditional probabilities of possible next states given the current state and the observation sequence (Lafferty et al. 2001). In general, label bias arises when the choice of an upstream (earlier) tag resets probabilities, allowing a choice of bad downstream tags based on the upstream tag. Because of these all facts, we used CRF as an algorithm for modeling.

2.4.1 The Model

A CRF, introduced by (Lafferty et al. 2001), is a framework for building probabilistic models to segment and label sequence data. They are probabilistic models for computing the probability $p(\vec{y}|\vec{x})$ of possible output $\vec{y} = (y_1, \ldots, y_n) \in Y$ given the input $\vec{x} = (x_1, \ldots, x_n) \in X$ which is also called the observation. There are variants of CRFs like linear-chain CRF (Klinger & Tomanek 2007) and Skip-chain CRF (Sutton & McCallum 2006). For we used linear chain CRF as a learning algorithm we will focus on linear-chain CRF. All the discussions that follow are based on (Klinger & Tomanek 2007).

A special form of a CRF, which is structured as a linear chain, models the output variables as a sequence. This special form of a CRF is known as linear chain CRF. Any CRF can be formulated as:

$$P(\vec{y}|\vec{x}) = \frac{1}{Z(\vec{x})} \prod_{j=1}^{n} \psi_j (\vec{x}, \vec{y}) \quad (1)$$

where $\vec{y}$ and $\vec{x}$ respectively represent the sequence of labels and observations that are going to be modelled. $\psi_j(\vec{x}, \vec{y})$ are the different factors corresponding to maximal cliques in the independency graph. The $Z(\vec{x})$ is a normalization factor. It normalizes the probability distribution to [0, 1] and it is computed as:

$$Z(\vec{x}) = \sum_{\vec{y}'} \prod_{j=1}^{n} \boldsymbol{\psi}_j (\vec{x}, \vec{y}') \quad (2)$$

Dealing with conditional probability distributions is difficult since they are complex in nature. The best way to deal with them is through conditional independence. Conditional independence is an important concept used to decompose complex probability distributions into a product of factors, each consisting of the subset of corresponding random variables. This concept makes complex computations (which are for example necessary for training or inference) much more efficient. The decomposition is represented by factors of the form $\psi_j(\vec{x}, \vec{y})$:

$$\psi_j(\vec{x}, \vec{y}) = exp\left(\sum_{i=1}^{m} \lambda_i f_i(y_{j-1}, y_j, \vec{x}, j)\right) \quad (3)$$

where λ_i represents the parameter estimated for each feature from the training data and *m* is the total number of features extracted for a single word. Equation 3 is an exponential function (base *e*). The notation *exp* is used since the expression in the bracket is complex to be written as an exponent. $f_i(y_{j-1}, y_j, \vec{x}, j)$ is a feature function. A feature function is computed by looking at two adjacent labels y_{j-1}, y_j, the whole observation sequence $\vec{x}$ and the current position in the input sequence j. Feature function produces a real value. For example, we can define a simple feature function which produces binary values: it is 1 if the current word is ETC and the current label y_j is ORGANIZATION.

$$f_1(y_{j-1}, y_j, \vec{x}, j) = \begin{cases} 1 \ if \ y_j = ORGANIZATION \ and \ \vec{x} = ETC \\ 0 \ otherwise \end{cases} \quad (4)$$

Based on the above discussions, derived formulas and assuming that n + 1 is the length of the observation sequence, a linear chain CRF model can be written as:

$$p_{\vec{\lambda}}(\vec{y}|\vec{x}) = \frac{1}{Z_{\vec{\lambda}}(\vec{x})} \cdot exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m} \lambda_i f_i(y_{j-1}, y_j, \vec{x}, j)\right) \quad (5)$$

where *n* stands for the total number of words in the sentence and *j* is the position of the current word with the sentence. This is the model that is going to be trained as part of a machine learning component. In our NER work it will be trained on Afan Oromo NE corpus. The trained model will predict the possible NE words from the plain text through a process called inference. In the linear-chain CRF model given above (equation 5), the weights λ_i are not dependent on the position *j*. This technique, known as parameter tying, is applied to ensure a specified set of variables to have the same value. The normalization to [0, 1] is given by:

$$Z_{\vec{\lambda}}(\vec{x}) = \sum_{\vec{y}\in\gamma} exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m} \lambda_i f_i(y_{j-1}, y_j, \vec{x}, j)\right) \quad (6)$$

Summation over γ, the set of all possible label sequences, is performed to get a feasible probability. The next two successive sections will respectively discuss how a linear chain CRF model is trained and how inference is done. These two points will help us later in the implementation of our system to show how the model is trained on Afan Oromo NE corpus and how it recognizes possible NEs from a plain input text.

2.4.2 Training

Training a CRF model is the procedure of estimating the parameter λ in the model. In order to find λ, we need fully labelled training data $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}, \ldots (x^{(N)}, y^{(N)})\}$ where $x^{(1)}\ldots x^{(N)}$ is an observation sequence and $y^{(1)}\ldots y^{(N)}$ is a label sequence. Since CRFs define the conditional probability *p*(y|x), the appropriate way for parameter estimation is to maximize the conditional likelihood of the training data which can be done by applying the maximum-likelihood method. This means that training a linear-chain CRF model is done by maximizing the log-likelihood L on the training data τ.

$$\bar{L}(\tau) = \sum_{(\vec{x},\vec{y})\in\tau} \log p(\vec{y}|\vec{x})$$

$$= \sum_{(\vec{x},\vec{y})\in\tau} \left[\log \left(\frac{exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m}\lambda_i f_i(y_{j-1},y_j,\vec{x},j)\right)}{\sum_{\vec{y}'\in Y} exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m}\lambda_i f_i\left(y'_{j-1},y'_j,\vec{x},j\right)\right)} \right) \right] \quad (7)$$

The above formula is prone to an overfitting problem. A model which has been overfit will generally have poor predictive performance, as it can exaggerate minor fluctuations in the input data. For this reason, to regularize the training (make it smooth) a Gaussian prior is used. The Gaussian prior is represented as $-\sum_{i=1}^{n}\frac{\lambda^2}{2\delta^2}$. The parameter δ^2 models the trade-off between fitting exactly the observed feature frequencies and the squared norm of the weight vector. The smaller the values are, the smaller the weights are forced to be, so that the chance that few high weights dominate is reduced. For the purpose of avoiding overfitting, the likelihood function L(τ) can be reorganized as:

$$L(\tau) = \sum_{(\vec{x},\vec{y})\in\tau} \left[\log \left(\frac{exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m}\lambda_i f_i(y_{j-1},y_j,\vec{x},j)\right)}{\sum_{\vec{y}'\in Y} exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m}\lambda_i f_i\left(y'_{j-1},y'_j,\vec{x},j\right)\right)} \right) \right] - \sum_{i=1}^{n}\frac{\lambda^2}{2\delta^2} \quad (8)$$

$$= \underbrace{\sum_{(\vec{x},\vec{y})\in\tau}\sum_{j=1}^{n}\sum_{i=1}^{m}\lambda_i f_i\left(y_{j-1},y_j,\vec{x},j\right)}_{A} -$$

$$\underbrace{-\sum_{(x,y)\in\tau}\log\underbrace{\left(\sum_{\vec{y}'\in Y} exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m}\lambda_i f_i\left(y'_{j-1},y'_j,\vec{x},j\right)\right)\right)}_{Z_{\vec{\lambda}}(\vec{x})}}_{B} - \underbrace{\sum_{i=1}^{m}\frac{\lambda^2}{2\delta^2}}_{C}$$

The partial derivations of L(τ) by the weights λ_k are computed separately for the parts *A*, *B*, and *C*. The derivation for part *A* is given by:

$$\frac{\partial(A)}{\partial\lambda_k} = \sum_{(\vec{x},\vec{y})\in\tau}\sum_{j=1}^{n} f_k(y_{j-1},y_j,\vec{x},j) = \tilde{E}(f_i) \quad (9)$$

The derivation of part *A* gives the expected value under the empirical distribution of a feature f_i, $\tilde{E}(f_i)$. The derivation for part *B*, which corresponds to the normalization, is given as follows and it specifies the expectation under the model distribution $E(f_i)$.

$$\frac{\partial(B)}{\partial\lambda_k} = \sum_{(\vec{x},\vec{y})\in\tau}\sum_{\vec{y}\in\gamma} P_{\vec{\lambda}}(\vec{y}|\vec{x})\sum_{j=1}^{n} f_k\left(y'_{j-1}, y'_j, \vec{x}, j\right) = E(f_i) \quad (10)$$

Part *C*, the derivation of the penalty term, is given by:

$$\frac{\partial(C)}{\partial\lambda_k} = -\frac{2\lambda_k}{2\delta^2} = -\frac{\lambda_k}{\delta^2} \quad (11)$$

Based on equations 9, 10 and 11, the partial derivations of L(τ) is equivalent to:

$$\frac{\partial L(\tau)}{\partial\lambda_k} = \tilde{E}(f_k) - E(f_k) - \frac{\lambda_k}{\delta^2} \quad (12)$$

and the maximum likelihood method finds the parameter λ_k by equating the first derivation to 0 as:

$$\tilde{E}(f_k) - E(f_k) - \frac{\lambda_k}{\delta^2} = 0 \quad (13)$$

Computing $\tilde{E}(f_i)$ is easily done by counting how often each feature occurs in the training data. Computing $E(f_i)$ directly is impractical because of the high number of possible label sequences. Thus, a dynamic programming approach known as the Forward-Backward Algorithm is applied which is complex to explain in this study.

2.4.3 Inference

The problem of inference is to find the most likely sequence $\vec{y}$ for given observations $\vec{x}$. When adapted to an NER task, this means, inference is associated with finding the most likely sequence of NE categories (like PERSON, ORGANIZATION, LOCATION and

OTHER) that can best represent an input plain text. Note that this is not about to choose a sequence of labels, which are individually most likely. That would be the maximization of the number of correct states in the sequence. To solve an inference problem the Viterbi Algorithm is applied.

To start with, the quantity $\delta_j(s|\vec{x})$ which is the highest score along a path, at position j, which ends in state s, is defined as:

$$\delta_j(s|\vec{x}) = \max_{y_1,y_2,\ldots,y_{j-1}} p(y_1, y_2, \ldots, y_j = s|\vec{x}) \quad (14)$$

The induction step is:

$$\delta_{j+1}(s|\vec{x}) = \max_{s' \in S} \delta_j(s') \cdot \psi_{j+1}(\vec{x}, s, s') \quad (15)$$

The array $\psi_j(s)$ keeps track of the *j* and *s* values. Once this is available the inference is carried out, using viterbi algorithm, in four steps as follows:

1. **Initialization**: The values for all steps from the start state $\perp$ to all possible first states *s* are set to the corresponding factor value.

$$\forall_s \in S: \delta_1(s) = \psi(\vec{x}, \perp, s) \quad (16)$$

$$\psi'(s) = \perp$$

2. **Recursion**: The values for the next steps are computed from the current value and the maximum values regarding all possible succeeding states s'.

$$\forall_s \in S: 1 \leq j \leq n: \delta_j(s) = \max_{s' \in S} \delta_{j-1}(s')\ \psi(\vec{x}, s', s) \quad (17)$$

$$\psi_j(s) = \operatorname{argmax}_{s' \in S} \delta_{j-1}(s')\ \psi(\vec{x}, s', s)$$

3. **Termination**:

$$p^* = \max_{s' \in S} \delta_n(s') \quad (18)$$

$$\vec{y}_n^* = \operatorname{argmax}_{s' \in S} \delta_n(s') \quad (19)$$

4. **Path backtracking**: Recompute the optimal path from the lattice using the track keeping values ψ_t.

$$\vec{y}_t^* = \psi_{t+1}(\vec{y}_{t+1}^*) \qquad t = n-1, n-2, \ldots, 1 \tag{20}$$

This completes the viterbi algorithm. The path backtracking step used to follow the path with larger probability and the labels that are found along the path will be assigned to the observation sequences.

2.5 Afan Oromo Language

2.5.1 Background

Afan Oromo is a mother tongue for Oromo. The Oromo people are dominantly inhabited in Oromia region. Neighbouring regions and countries of Oromia region are: Amhara region in the north, Beninshangul region in the northwest, Sudan in the west, Gambella region in the southwest, SNNP region and Kenya in the south, Somali region in the east and Afar region in the northeast. As an indication of strong ties of the Afan Oromo language with other regions in Ethiopia, Oromia is the region that neighbours all the regions in Ethiopia except the Tigray region. Oromo people are also inhabited in neighbouring countries like Kenya and Somalia.

Currently Afan Oromo is the official language of the regional state of Oromia (the largest regional state in Ethiopia) being used as a working language in offices, educational language for all non-language subjects in junior-secondary schools (1-8 grades). At the country level, in Ethiopia, out of the total 22 public universities 8 are offering degree programs majoring in Afan Oromo and Addis Ababa University is offering Afan Oromo at Masters degree level. Afan Oromo is also widely used as both written and spoken language in some neighbouring countries, including Kenya and Somalia (Amanuel 2006).

2.5.2 Writing system of Afan Oromo language

With regard to the writing system, „*Qubee*“ (Latin-based alphabet) has been adopted and became the official script of Afan Oromo since 1991 (Gamta 1992). This writing system was adapted largely from the fact that its characters do explicitly represent the vowels and the consonants of a language (Gamta 1992). The „*Qubee*“ writing system has a total of 33 letters that consists of all the 26 English letters with an addition of 7 combined consonant letters. All

the vowels in English are also vowels in „*Qubee*". Vowels have two natures in the language and they can result in different meaning. The natures are short and long vowels. A vowel is said to be short if it is one. If it is two, which is the maximum, then it is called long vowel. Consider these words: lafa (ground), laafaa (soft). The rest of „*Qubee*" are consonants. The combined consonant letters are known as „*qubee dachaa*" and they are *ch*, *dh*, *sh*, *ny*, *ts*, *ph* and *zy*.

In addition to the 33 symbols, it is recommended to know the following principles associated with „*Qubee*" (Gamta 1992):

1. Two vowels in succession indicate that the vowel is long, e.g. *bitaa* (left);
2. Gemination (a doubling of a consonant) is phonemic in Afan Oromo, e.g. *damee* (branch), *dammee* (sweety);
3. h is not geminated at all;
4. The same word can have two or more forms depending on its context, e.g. nama kadhu (ask person), namaa kadhu (ask for person);
5. When it occurs at the end of a word, the single "a" is pronounced schwa (inverted e) whereas it is pronounced (delta) elsewhere;
6. Understandably, instead of diacritic signs, the combined letters are used so as to align them with typewriter characters.

2.5.3 Syntax of Afan Oromo sentences

There are a lot of rules towards word ordering during sentence construction in Afan Oromo. But for this study we want to focus on two syntaxes: Main clause word order and Noun phrase word order. The points discussed in this section are based on (Catherine 2001).

Main clause word order

In general, the normal order of words during sentence construction in main clause follows the SOV format: Subject Object Verb. However, the resulting sentence structure may vary as shown in the following cases.

I. Subject + verb

Uffati qoore. (The cloth dried).

Baga gammadde. (Congratulations).

II. Subject + complement (object or adverbial) + verb

Ittaanaan dafee dhufa. (Itana will come soon).

Burraaqaan gara mana sagadaa deeme. (Buraka left for church).

III. Complement + verb

Harkaa fi fuula isaa dhiqate. (He washed his hands and his face, too).

Tokkummaa qabaadhaa. (Have a unity).

IV. The sequence of different complements does not follow a special order. It seems that they are arranged according to semantic reasons.

Harka saamunaadhaan dhiqadhu!

Hand soap with wash (Wash your hand with soap!)

Noun phrase word order

The noun usually precedes the qualifiers. The qualifiers are arranged in the sequence: noun – adjective – possessive/demonstrative pronoun. The subject marker is added to the noun itself and to the qualifiers of the noun as the base form is repeated in all qualifiers of the noun.

Namicha dheeraa sana argitee? (Did you see that tall man?)

Intalli diimtuun sun baay"ee bareeddi. (That red girl is very beautiful).

2.5.4 Word Categories of Afan Oromo

In this section we will see some of the word categories that are in use in Afan Oromo language and have contributions to our study. The points discussed here are based on the papers (Diriba 2002) and (Getachew 2009).

Noun Category

Nouns are words that are used to name or identify any of categories of things, people, places or ideas or a particular of one of these entities. Based on its contextual position, nouns are commonly found at the beginning of a sentence in Afan Oromo. For instance, words that are italic in the following sentences are nouns.

Hoolaan marga dheeda. (The sheep grazes grass).

Diimaan filannoo kooti. (Red is my preference).

Noun can further be classified as common or proper noun. A proper noun is a noun that will name a specific, usually a one-of-a-kind item. In Afan Oromo, it begins with a capital letter no matter where it occurs in a sentence. Example:

Nagaasaan barsiisaa dha. (Nagasa is a teacher).

Najjoon godina *Wallaga Lixaa* keessatti argamti. (Nedjo is found in West Wollega zone).

Verb Category

The verbs are words or compound of words that expresses action, a state of being and/or relationship between two things. In their most powerful and normal position, they are found at the end of the sentence as shown below.

Caalaan farda *bite*. (Chala bought a horse)

Leensaan *dhufte*. (Lensa has come)

Adverb Category

Afan Oromo adverbs are words which are used to modify verbs. Adverbs usually precede the verbs they modify or describe. Example:

Diinkolaas *bor* deema. (Dinkolas will go tomorrow).

Akaakayyuun koo *kaleessa* dhufe. (My grandfather came yesterday).

Adjective Category

Adjectives in a sentence modify nouns to denote quality of a thing; that is, it specifies to what extent a thing is as distinct from something else.

For example,

Faqqadaan *gabaabaa* dha. (Fekede is short).

Haati manaa isaa *furdoo* dha. (His wife is fat).

Pronoun category

In Afan Oromo, like in other languages, pronoun is a word that is used instead of a noun or noun phrase. They are characterized based on number and gender. For instance, *ishee 'her', isa 'him', isaan 'they'* are some.

For example:

Haati manaan Margoo *furdoo* dha. (Margo'ʼs wife is fat).

Haati manaan *isaa* furdoo dha. (His wife is fata).

Adpositions in Afan Oromo

The term adpositions refer to words, which will have meaning only when they are attached or used together with other words such as nouns, verbs, pronouns and adjectives. For example:

STVO-*dhaaf*	(for STVO)	Darartuu-*n*	(Darartu is)
Baddannoo-*tti*	(in Baddano)	Bariisaa-*rraa*	(from Barissa)

2.5.5 Named Entities in Afan Oromo

It is understood that NEs are words that are used to name a person, organization, location and other things. In this section, we will discuss the nature of NEs in Afan Oromo. Our study focused on the recognition of four categories of NEs: PERSON, LOCATION, ORGANIZATION and MISCELLANEOUS. The miscellaneous category included DATE, TIME, MONETARY VALUE and PERCENTAGE. The discussions made here are based on the pattern of NEs noticed during corpus development and from the general background of the researcher. The researcher learnt all subjects[5] of his elementary schools (1-8 grades) in Afan Oromo. He also took Afan Oromo as a subject from 9-12 grades.

[5] Except for Amharic and English subjects.

The nature and properties of NEs in Afan Oromo mostly resembles that of English. They are categorized under the word category of proper noun. In Afan Oromo, NEs like name of a person, organization and location share common principal characteristics that, they are capitalized when written. Additional properties such as clue words and suffixes are also used to identify NEs in Afan Oromo. Let us see the nature of NEs from these properties point of view one by one.

Capitalization is the main marker of NEs which other properties rely on. It is one of those properties used in our study. Unless they are of type numbers (like date, time, monetary values), NEs start with a capital letter regardless of whether they are located at the beginning, middle or end of the sentence as shown below.

Obbo *Gammadaa Amantii* gara *Walisoo* deeme.

(Mr. *Gameda Amenti* went to *Waliso*).

Warshaan Simmintoo Mogor omisha isaa dachaa sadiin dabale.

(*Mogor Cement Factory* tripled its production).

Clue words are other useful properties in identifying NEs in Afan Oromo. Clue words are common words that mostly precede NE words. Clue words basically work in conjunction with capitalization. Some of the common clue words used in Afan Oromo are „*ɔbbo*", „*ɠaadde*" for PERSON, „*ɱagaalaa*", „*ɠanaa*" for LOCATION and „*warshaa*","*baankii*" for ORGANIZATION, „*ƀara*", „*ɖaqiiqaa*" for DATE/TIME, „*ɋarshii*", „*ɖoolaara*" for MONETARY VALUES, „*ɖhibbeentaa*" for PERCENTAGE. Unlike that of English, in Afan Oromo, clue words for location, organization and miscellaneous NE types, come before NEs they represent. Consider the following.

Yunivarsitii Addis Ababaa (Addis Ababa *University*)

Magaalaa Dirree Dhawaa (Dire Dawa *town*)

Qarshii 100 (100 *birr*)

In addition to the previous commonly listed clue words, we also gathered and organized a list of clue words to be used for our study. These words frequently come after NEs in news texts. Our system used both clue words as features.

Elmituu *W.A* aksiyoona gurguraa jira. (Elmitu *S.C* is selling shares).

Shifarraa Jaarsoo *gabaaseera*. (Shifara Jarso reports).

Suffixes can also be used as properties to detect NEs in Afan Oromo. They are not used by themselves rather they work in conjunction with capitalization. They provide information being attached to the NE word itself. When attached to the NE itself the current word should start with capital letter as shown in the following sentence.

Tolashiin Mattuu-tii hojjetti. (Tolashi works in Mattu).

The above discussed properties are collectively called features in NER task. They are essential information for any NER system to recognize NEs as discussed in section 2.3.2. Based on this fact, we used features to conduct our study. Some of the possible features that can be used to identify NEs in Afan Oromo are found in appendix A.

Numerals in Afan Oromo

Numerals are words representing numbers. They can be cardinal or ordinal numbers. In Oromo, the ordinal numbers are formed from the cardinal numbers by suffixing the suffix –*affaa*. For example:

Cardinal	**in English**	**Ordinal**	**in English**
tokko	one	tokko-*ffaa*	first
afur	four	afura-*ffaa*	fourth

Some of the Numerals used in Afan Oromo are listed under appendix B.

Date/Time in Afan Oromo

This merges both date and time. In Afan Oromo, both can be written either in number or numeral. The writing format of date mostly follows the pattern dd/mm/yyyy or dd/mm/yy. I.e. *02/11/2002 A.L.H*. (02/11/2002 E.C) or *21/07/10 A.L.A* (21/07/10 G.C). For this study, the date category included day, month, year or the combination of them. Example:

Gurraandhala 25 (*February 25*)

Gaafa *kibxataa* (*Tuesday*)

Bara *1522tti* (in the year 1522)

The complete list of days and months used in Afan Oromo is found in appendix C.

Time informs the clock value at which an event occurs. Time expressions in Afan Oromo are similar to that of English. Example:

Amma sa‟atii *4* ta‟eera. (Now it‟s 4 O‟clock).

Daqiiqaa *kudhan* booda. (After ten minutes).

Monetary Values in Afan Oromo

Monetary values are value or worth that a product or service would bring to someone if sold. In Afan Oromo, they can be written either in number or numeral. For example,

Qarshii 25. (25 birr[6]).

Paawundii digdamii shan. (Twenty five pound).

Percentages in Afan Oromo

Percentages are used to express how large/small one quantity is, relative to another quantity. It is a way of expressing a number as a fraction of 100. The symbol to represent a percentage, %, is also used in Afan Oromo. But here it comes before the number, unlike that of English. The word „percent‟ has also an equivalent in Afan Oromo as „dhibbantaa‟ or „dhibbeentaa‟. Example:

Dhibbeentaa 14n dabale. (14 percent increase).

%14n dabale. (14% increased).

2.6 Summary

NLP is a computer processing of natural/human language. It has different group of tasks such as IR and IE. IR seeks to extract all of the information found in a document while IE extracts specific kinds of information from document. NER is a special task of IE that focuses on the detection and classification of the so called NEs into predefined categories. NEs are words

[6] Ethiopia's currency. In Afan Oromo it is called '*Qarshii*'.

like name of persons, organizations and locations. Since NEs carry important information about a text they are the target of extraction. NER systems are developed using three approaches: Rule-based, Machine Learning and Hybrid. To recognize NEs out of a text, an NER system requires a set of features that clearly identify them from other words of a text. Commonly, the performance of an NER system is evaluated with three evaluation metrics. These are Precision (P), Recall (R) and F-measure. A CRF is the current state of the art algorithm for NER systems that estimates the conditional probability $p(\vec{y}|\vec{x})$ of a label sequence Y and an observation sequence X, which is preferable over the joint probability p(y, x) for relational data. Afan Oromo is one of the widely spoken languages in Ethiopia and has a writing system called „*Qubee*".

Chapter Three: Related Works

In this chapter, we will present some works that are done so far in the area of NER. We will explore a little of NER tasks studied for four languages: English, Spanish, Japanese and Hindi.

3.1 Named Entity Recognition for English

The first work we are going to review, for English, is the work of (Finkel et al. 2005). In most of the current statistical models of NLP, it is common to use the Viterbi algorithm, a dynamic programming algorithm, which focuses only on local structures. However, using only local structures has the key limitation as natural language contains a great deal of non-local structure. The authors of this paper aimed at overcoming this limitation by using an algorithm that permits the use of non-local structure, Gibbs sampling, and demonstrate the significant improvement over Viterbi‟s algorithm on two IE tasks among which there is found a CRF-based statistical NER system known as *stanford-ner*. The software provides a general (arbitrary order) implementation of linear chain CRF sequence models coupled with well-engineered feature extractors for NER. Included are a good 3 class (PERSON, ORGANIZATION, and LOCATION) for English and another pair of models trained on the CoNLL 2003 English training data.

The models of non-local structure, employed in this system, are themselves just sequence models, defining a probability distribution over all possible state sequences. The model also makes it possible to flexibly model various forms of constraints in a way that is sensitive to the linguistic structure of the data. The distributional similarity features used here improved the performance but the models require considerably more memory. The dataset used by the system was created for the shared task of the CoNLL. The English data is a collection of Reuters newswire articles annotated with four entity types: *person* (PER), *location* (LOC), *organization* (ORG), and *miscellaneous* (MISC). The data is separated into a training set, a development set (testa), and a test set (testb). The training set contains 945 documents, and

approximately 203,000 tokens. The development set has 216 documents and approximately 51,000 tokens, and the test set has 231 documents and approximately 46,000 tokens. The models were all trained on the union of the CoNLL, MUC-6, MUC-7 and ACE named entity corpora, and as a result the models are fairly robust across domains.

In their experiments they compared the impact of adding the non-local models with Gibbs sampling to their baseline CRF implementation. In the CoNLL named entity recognition task, the non-local models increase the F1 accuracy by about 1.3% obtaining a total of 86.86%. This gain appears to be modest when compared to the winner of the CoNLL English task that reported an F1 score of 88.76%. However, the biggest drawback of this model is the computational cost. For example, taking 100 samples dramatically increases test time. The system does not also work for the miscellaneous (MISC) entity mention.

On the other hand, (Borthwick et al. 1998) demonstrates a new system based on maximum entropy, called MENE (Maximum Entropy Named Entity) which was NYU's (New York University) entrant in the MUC-7 NER task. Being developed within the frame-work of maximum entropy theory and utilizing a flexible object-based architecture, the system is able to make use of an extraordinarily diverse range of knowledge sources in making its tagging decisions. These knowledge sources include such features as Binary features ("the token begins with a capitalized letter" or "the token is a four-digit number"), Lexical features (the tokens that surround the entity are compared with the vocabulary), Section features (current section of an article like "Date", "Preamble", and "Text") and External systems features (the output of hand-coded "Proteus"[7] named-entity tagger). It also makes use of a broad array of dictionaries of useful single or multi-word terms such as first names, company names, and corporate suffixes. These dictionaries required no manual editing and were either downloaded from the web or were simply "obvious" lists entered by hand. The system was ranked 4th out of the 14 entries in the N.E. evaluation.

The system, built from on-the-shelf knowledge sources, contained no hand-generated patterns and achieved a result on dry run data which is comparable with that of the best statistical systems. Given appropriate training data, it is believed that this system is highly porTable to

[7] It is a NE tagger developed at NYU to enter the MUC-6 competition.

other domains and languages and has already achieved state-of-the-art results on upper-case English.

MENE's maximum entropy training algorithm gives it reasonable performance with moderate-sized training corpora or few information sources, while allowing it to really shine when more training data and information sources are added. The system was trained on 350 aviation disaster articles (this training corpus consisted of about 270,000 words, which the system turned into 321,000 tokens). The training data focused on airline disasters while the test data was on missile and rocket launches. Having utilizing a lot of features and a bunch of training data, the performance of the system was not found to be as expected. It obtained an F-measure of 88.80%. It is believed that the deterioration in performance was mostly due to the shift in domains caused by training the system on airline disaster articles and testing it on rocket and missile launch articles.

3.2 Named Entity Recognition for Spanish

The work of (Kozareva, 2006) presented the development of a Spanish NER system based on machine learning approach that employs two algorithms: instance-based and decision trees. No morphologic or syntactic information was used. However, they proposed and incorporated a very simple method for automatic gazetteer construction which can be easily adapted to other languages. An already available NE annotated corpus was used in order to conduct an exhaustive and comparative NER study when labeled and unlabeled data is present. To explore the effect of labeled and unlabeled training data to their NER, two types of experiments were conducted. For the supervised approach, the labels in the training data were previously known. For the semi-supervised approach, the labels in the training data were hidden.

They used bootstrapping which refers to a problem setting in which one is given a small set of labeled data and a large set of unlabeled data, and the task is to induce a classifier. Two classifiers were used to decide as to how a labeled example is added into the training data. Accordingly, a newly labeled example is added into the training data L, if the two classifiers C1 and C2 agreed on the class of that example.

They identified a set of feature vectors which comprises the contextual, lexical and gazetteer information that helps to differentiate NEs from other normal words. NEs extracted using the identified features are then further filtered using validation patterns, Bigram and Trigram, in order to improve the performance and generate automatic gazetteer lists (for person and location mentions). 173,468,453 words which are unlabeled are used as a corpus.

Finally, the performance of the system is evaluated when labeled and unlabeled training data was available. Results show that by the help of bootstrapping, the supervised classifier obtained F1 measure of 91.88% and the semi-supervised classifier obtained 81.62%.

On the other hand, (Kozareva et al., 2005) demonstrated the building of a complete NER system using small set of labeled and large amount of unlabeled data with the help of self-training and co-training algorithms. Both algorithms allow a classifier to start with few labeled examples to produce an initial weak classifier and later to use only the unlabeled data for improving the performance.

The system works in two phases: Named Entity Detection (NED) & Named Entity Classification (NEC). During the NED phase NEs are detected with the help of predefined set of features. The features used are lexical (position of the word, word form, word makeup), orthographic (all letters in capitals, initiate in capitals), trigger words and gazetteer. Considering the time-performance disadvantage of self-training and co-training techniques, and the complexity of each one of the tasks they have to resolve, they decided to use self-training for NED. A single classifier is used for self-training and it is based on K-nn algorithm which is known by its property of taking into consideration every single training example when making the classification decision and significantly useful when training data is insufficient. The classifier labels the unlabeled data for several iterations and converts the most confidently predicted examples of each class into a labeled training example. The instances detected by the self-training algorithm are classified with the voted co-training method.

The co-training algorithm used here is named as Voted Co-training, an extension of a co-training strategy proposed by (Goldman and Zhou, 2000). The idea behind the strategy of (Goldman and Zhou, 2000) is that the two algorithms make use of different representations

for their hypotheses and thus they can learn two diverse models that can complement each other by labeling some unlabeled data and enlarge the training set of the other. In order to decide which unlabeled examples a classifier should label, they derive confidence intervals. The strategy of voted co-training used in this work is as follows.

The voted co-training, which is used for NEC, starts with a small set of hand-labeled examples and three classifiers that learn the same pool of unlabeled examples. The unlabeled data set is turned into labeled, for each iteration, and instances with some growing size predefined by the user are added into the training set. In order to guarantee the labeling confidence of the unlabeled examples a voting strategy will be applied for each example. Two initial classifiers compare the predicted class for each instance. When they agree, this instance is added directly into a temporal set T, when they disagree, the prediction of an external classifier is considered and voting among them is applied. Instances added to the already existing temporal set T are those on which the external classifier agrees with at least one of the initial classifiers.

The system has been developed and tested for Spanish language. The test file contained around 21300 tokens of which 2000 have been annotated by human as NEs. The obtained results are encouraging, 90.37% f-score for detecting 2000 entities using 8020 unlabeled examples and 67.22% f-score for entity classification with 792 unlabeled examples.

Finally, they outlined that for instance-based learning, it is not necessary to store all training instances. Because it can happen that an instance correctly classified by the self-training algorithm may not contribute with new information thereby not improving the performance.

3.3 Named Entity Recognition for Indian Languages

The NER system for the second widely spoken Indian language, Telugu, was demonstrated in (Srikanth & Murthy 2008). The system is developed in such a manner that, first they started building a CRF (Conditional Random Fields) based Noun Tagger. They then developed a rule

based NER system for Telugu. Their focus was mainly on identifying person, place and organization names.

The system functions in two phases. In the first phase, called noun identification phase, each word in a sentence is treated as to which category it comes under (noun or not noun). This is done with the help of feature vector consisting of morphological features[8], length, stop words, affixes, part-of-speech, position, orthographic information and suffixes.

In the next phase, called noun classification phase, nouns which have already been identified in the noun identification phase are checked for named entities.

The classification is done using two different systems, one using heuristic based system and the second using machine learning Conditional Random Field (CRF) based system. It is observed that the former system faces difficulty in differentiating between ambiguities. For the later system, it is observed that there is not much improvement in the performance of the system by including more of the neighboring words as features. The experiments conducted reveals that the performance of both systems is comparable to each other but CRF based system outperforms the heuristic based system when gazetteer and noun tagger features are considered.

The corpus they used adds up to nearly 27.3 million words consisting of each word annotated as noun or not-noun. Trained on a manually tagged data of 13,425 words and tested on a test data set of 6,223 words, the Noun Tagger has given an F-Measure of about 92%. A manually checked NE tagged corpus of 72,157 words has been developed using the rule based tagger through bootstrapping. Good performance has been obtained using the majority tag concept. They have obtained overall F-measures between 80% and 97% in various experiments. The system does not handle multi-word expressions - only individual words are recognized and partial matches are also considered as correct in their analyses.

[8] Morphological analyzer developed at University of Hyderabad has been used to obtain the root word and the POS category for the given word.

A successful Hindi NER system was presented in (Li and McCallum, 2003), which was developed using CRF with feature induction. The system was required to find all appearances of three types of entities: Person, Location and Organization.

The CRF‟s tremendous freedom to include arbitrary features helped them to automatically discover relevant features by providing a large array of lexical tests though they have little knowledge of the language. While CRFs generally can use real-valued feature functions, in their experiments, all features are found to be binary.

On the other hand, the ability of feature induction to automatically construct the most useful feature combinations that increase conditional likelihood let the training procedure automatically perform the feature engineering by selecting feature conjunctions that will significantly improve performance. At the beginning they started with no features at all and choose new features iteratively. In each iteration, some set of candidates are evaluated (also using the Gaussian prior), and the best ones are added to the model. In an effort to reduce overfitting, they used a combination of a Gaussian prior and early stopping. Even though, the Gaussian prior has also been thought to significantly reduce overfitting, they observed that its effectiveness for this purpose is less than widely believed. Adapting the BIO scheme, they used two kinds of labels for each entity type: B-type for the start of an entity and I-type for the inner part. For non-entities, they used the label O.

The training set for the system consisted of 340K words. Feature induction constructed 9,697 features from an original set of 152,189 atomic features; many are position-shifted but only about 1% are useful.

To train the CRF, they experimented with various options, such as first-order versus second-order models, using feature induction or not and using lexicons or not. In an effort to reduce overfitting, they also tried different Gaussian priors and early-stopping. The evaluation shows that, the first-order model performs slightly better than the second-order model on the validation set, and the testing performance is significantly better when using feature induction. Using lexicons or not does not make much difference, and tight Gaussian priors do not improve the performance.

Their model does not perform as well on the test set. They hypothesize that this phenomenon may be due to the significant mismatch between the training/validation data and the test data. Highest test set accuracy of their system is the F-value of 71.50%.

3.4 Named Entity Recognition for Japanese

An interesting Japanese NER task that combined simple rule generation and decision tree (RG+DT) algorithm was presented in (Isozaki 2001). When compared to other statistical methods, the advantage of RG+DT is its readability and independence of generated rules. According to the experiment, this method‟s performance is comparable to that of the maximum entropy system, and it can be trained more efficiently.

Extracting NEs in Japanese is a difficult task. This is due to the reason that:

- The language does not have a useful hint for capitalization, a feature that simplifies the detection of NEs in such languages as English.
- Proper noun and common nouns also look very similar.
- The rare use of inter-word spacing makes the tokenization process difficult exacerbating the task of NER. For the current system, however, they used an off-the-shelf morphological analyzer for tokenization.

The RG+DT system generates a *recognition rule* from each NE in the training data. Then, the rule is refined by decision tree learning. By applying the refined recognition rules to a new document, it gets NE candidates. Then, non-overlapping candidates are selected by a kind of longest match method.

The methodology is consisting of the following procedures.

Generation of recognition rules: Each tokenized NE is converted to a recognition rule that is essentially a sequence of part-of-speech (POS) tags in the NE. Last word of a NE is used as a basic feature for classification. It is registered into a *suffix dictionary* for each non-numerical NE class (i.e., ORGANIZATION, PERSON, LOCATION, and ARTIFACT) in order to accept only reliable candidates. When a large amount of training data is given, thousands of recognition rules are generated. For efficiency these recognition rules are

compiled by using a hash Table that converts a hash key into a list of relevant rules that have to be examined.

Refinement of recognition rules: Since some recognition rules are not reliable, NE candidates for a rule can be obtained by applying each recognition rule to the untagged training data. By comparing the candidates with the given answer for the training data, it is possible to classify them into positive examples and negative examples for the recognition rule. Consequently, decision tree learning applied to classify these examples correctly.

Arbitration of candidates: Once the refined rules are generated, they can be applied to a new document. Since overlapping tags are not allowed, a kind of *left-to-right longest match method* is used. In this case, the starting points are compared with the earliest ones. If two or more candidates start at the same point, their ending points are compared and the longest candidate is selected. Therefore, the candidates overlapping the selected candidate are removed from the candidate set. This procedure is repeated until the candidate set becomes empty.

The system used the standard IREX training data (CRL NE 1.4 MB and NERT 30 KB). It works well and obtained F-measures of 84.03% and 87.43% on two different training data. The main reason behind its good performance is, at the first place, the percentage of long NEs is negligible. 91% of the NEs in the training data have at most three words. Second, the POS tags frequently used in NEs are limited. However, according to the observation, conventional decision tree systems have not shown good performance.

On the other hand, (Asahara & Matsumoto 2003) presented the extraction of Japanese NEs using redundant morphological analysis. Generally, Japanese NE extraction is done in the following ways:

- First, a text is segmented into words and is annotated with POS tags by a morphological analyzer.

- Then, a chucker brings together the words into NE chunks based on contextual information.

However this way of extracting NEs cannot extract NEs whose segmentation boundary contradicts that of morphological analysis outputs. To cope up with the problem, a more straightforward method is proposed in their paper in which the chunking process is performed on character basis. In this method, each character receives annotations with character type and redundant part-of-speeches (POS) information of the words found by a morphological analyzer. The redundant outputs of the morphological analysis are used as the base features for the chunker to introduce more rich information. Then a support vector machine (SVM) based chunker is implemented for the chunking process. This method achieved better score than all the systems reported for IREX[9] NE extraction task prior to this study.

The method is based on the following three steps:

1. A statistical morphological/POS analyzer is applied to the input sentence and produces POS tags of the *n*-best answers. In practice, log likelihood is used as cost. Naturally, maximizing probabilities minimize the costs. So, in this method, redundant analysis output means the top *n*-best answers within a certain cost width.
2. Each character in the sentences is annotated with the character type and multiple POS tag information according to the *n*-best answers. From the output of redundant analysis, each character receives a number of features. POS tag information is subcategorized so as to encode relative positions of characters within a word. Then, a character is tagged with a pair of POS tag and the position tag within a word as one feature.
3. Using annotated features, NEs are extracted by an SVM-based chunker. They used the chanker, called *yamcha,* which is based on support vector machines. Binary classifiers are extended to n-class classifiers to facilitate the chunking process. It is done deterministically either from the beginning or the end of sentence.

[9] An evaluation-based project for Information Retrieval and Information Extraction in Japanese.

In addition, the study introduced n-best answers as features for chunking to capture the behavior of the morphological analysis. The behavior is that, the ambiguity of word segmentation occurs in compound words. When both longer and shorter unit words are included in the lexicon, the longer unit words are more likely to be output by the morphological analyzer. As a result, the shorter units tend to be hidden behind the longer unit words. However, introducing the shorter unit words is more necessary to named entity extraction to generalize the model, because the shorter units are shared by many compound words. Furthermore, n-best answers also solved unknown word problem.

NE features used, for chunking, in this model are *POS tags*, *characters*, *character types* and *NE tags*. For the evaluation of their method, they used CRL NE data which includes 1,174 newspaper articles and 19,262 NEs. This system attained the best result of any system reported on IREX workshop obtaining an F-measure of 87.21%. The redundant outputs of morphological analysis slightly improved the accuracy of NE extraction except for numeral expressions like monetary values.

3.5 Summary

In this chapter we presented the revision of a number of NER works by focusing on four languages: English, Indian, Spanish and Japanese. Our study is related to these works by focusing on Afan Oromo language. While reviewing, we focused on the algorithm, feature sets, corpus size used and performance of those works. The evaluations of the works have shown that systems based on hybrid approach obtained better performance than those systems that are done based on machine learning algorithm alone. NER is a new concept for Afan Oromo and the language is resource scarce. Taking this problem into account and considering what we obtained from the revision of the related works, we propose a new solution to Afan Oromo NER problem. The proposed solution will be based on a hybrid approach which consists of a machine learning model and stored rules. The overall summary of each work is presented in Table 3.1.

Table 3.1: Summary of related works

Paper	Algorithm	Features	Corpus	Overall Performance
Finkel et al. 2005	CRF	Local & non-local structure	300,000 tokens	F1 – 86.86%
Borthwick et al. 1998	MEM	- Capitalization - Lexical - Dictionaries - External system	321,000 tokens	F1 – 88.80%
Kozareva, 2006	Decision tree + bootstrapping	- Contextual - Lexical - Gazetteer	173,468,453 tokens (unlabelled)	F1 – 91.88%[10] & F1 – 81.62%[11]
Kozareva et al., 2005	Bootstrapping (K-nn algorithm)	- Lexical - Orthographic - Trigger words - Gazetteer	21,300 (2000 annotated by hand)	F1 – 90.37%[12] & F1 – 67.22%[13]
Srikanth & Murthy 2008	CRF + Rules	- Morphological - Lexical - Orthographic	13, 425 (manual) + 72, 157 (boot.)	F1 – 80% - 97%
Li and McCallum, 2003	CRF + feature induction	- Capitalization - Lexicons - Conjunctions	340,000	F1 – 71.50%
Isozaki 2001	Rule gene. + Decision trees	- POS tags - Suffix - Last word	CLR NE 1.4MB + NERT 30KB	F1[14] – 84.03% & 87.43%
Asahara & Matsumoto 2003	Redundant Morphological Analysis + SVM	- POS tags - Character types - NE tags	1, 174 newspaper (19,262 NEs)	F1- 87.21%

[10] Supervised
[11] Semi-Supervised
[12] Detection
[13] Classification
[14] On 2 different training data

Chapter Four: Design of AONER

In this chapter we will discuss the overall design of our system, Afan Oromo Named Entity Recognizer (AONER). First we will discuss the approach we followed in the study. Then the general overview of the proposed system architecture from the perspective of the system‟s flow of operations in the form of processes will be discussed. Finally, we will present the detail explanation of the phases along with the subcomponents included in each phases.

4.1 The Approach Used

From the related works that have been reviewed (Srikanth & Murthy 2008, Isozaki 2001) and from the general comments given on different literatures (like Mansouri et al. 2008), we examined that the performance of systems based on hybrid approach is better than other approaches. Accordingly, AONER is designed to follow a hybrid approach. Our approach consists of a supervised machine learning component that collaborates with a set of stored rules. The machine learning component learns from annotated training data. A set of stored rules will then assist the machine learning component during the recognition process. The rules will help the machine learning component to avoid confusion during the course of ambiguities and also helps to encompass those parts of the language that are not covered by the machine learning component.

The machine learning component will be implemented using a Conditional Random Fields (CRF) algorithm. A CRF algorithm has the efficiency of dealing with non-independent, diverse and overlapping features. It has also the capability of overcoming the label bias problem. These two factors make CRF algorithm better suited for NER tasks as the task abundantly relies on features to recognize NEs.

4.2 Architecture of AONER

As discussed in section 2.3.2, there is no common and consistent architecture that represent an NER system. This could be due to one or more of the next three reasons: the writing style and character code used in languages, and the domain dependability of the NER task. The approach followed by a system can also enforce an NER system to have different

architectures. Considering the behaviour of Afan Oromo language, our proposed architecture for AONER is shown in Figure 4.1.

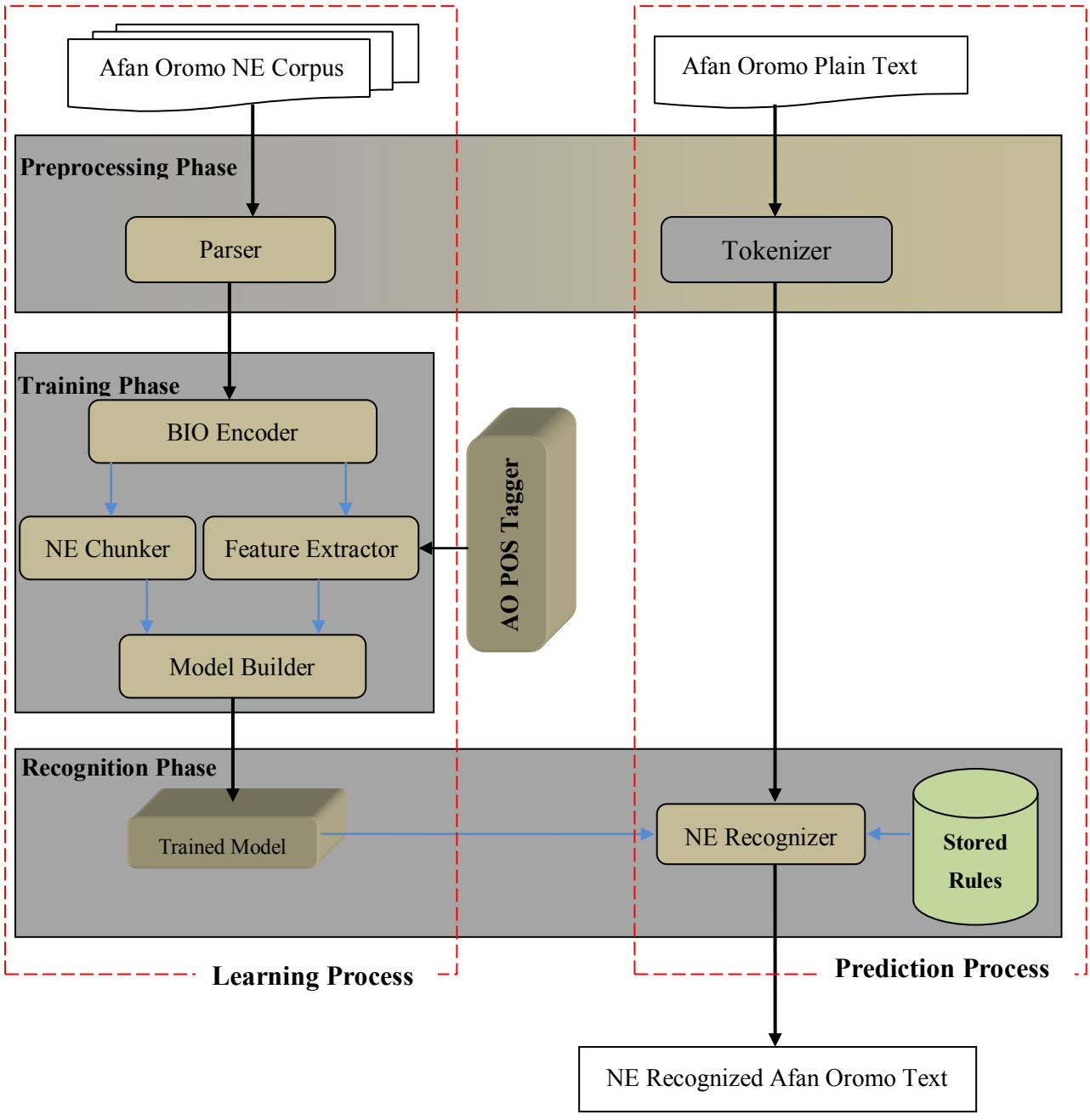

Figure 4.1: Proposed Architecture for AONER

In the AONER architecture, components that are functionally related are put in a shaded bigger rectangle, referred to as phases. Accordingly, our system has three main phases. These are *the pre-processing phase*, *the training phase* and *the recognition phase*. The red dotted

bigger boxes indicate processes specifying the evolution of the system. Our system has two processes. The left box is *the learning process* and the right box is *the prediction process*.

As we are using a machine learning statistical approach, we developed news based Afan Oromo NE Corpus that consists of more than 23,000 tokens. This is the training data that helps the intermediate components to learn and estimate necessary parameters to identify NEs. The training data is mainly collected from two Afan Oromo news papers: „*Bariisaa*" and „*Kallacha Oromiyaa*".

4.3 Processes of AONER

In AONER architecture, processes represent the evolution of the system. AONER is designed in such a manner that, it first learns properties and parameters associated with NEs from the training data. It then receives input plain text and predicts the possible NEs out of the plain text. The architecture has two processes: *learning process* and *prediction process*.

4.3.1 Learning Process

Training is handled by components in the *learning process*. Pre-processing is initially performed on the training corpus. The corpus is tagged based on the *Begin-Inside-Outside (BIO) scheme*. Parsing, which is done by a *parser*, is the only pre-processing operation applied on the training corpus. The *Parser* checks whether the training data contains all the necessary information required from a NE corpus and the well-formedness of the tagged data against CoNLL"s 2002 standard. If the tagged training data is found consistent, it is passed to the *BIO Encoder* which identifies a token and its corresponding NE tag through encoding. If there are inconsistencies with the tagged training data, the parser will report error. The token/tag sequence generated by the *BIO Encoder* is handed to both the *NE Chunker* and the *Feature Extractor*. *NE Chunker* collects related tokens with similar tag to create a phrase called chunk. *Feature Extractor* extracts necessary features to identify NEs based on the generated token/tag sequence. The chunks and the extracted features will then be used as an input to the *Model Builder*. Once all the necessary inputs are fulfilled the builder commences the model building procedure to generate a *trained model*. In general, the learning process takes the training corpus and it will finally provide a trained model that will be used in the prediction process.

4.3.2 Prediction Process

The prediction process works on the plain text entered by a user. First, the plain text is pre-processed. Tokenization, which is done by *tokenizer*, is the only pre-processing task performed on the input plain text. The tokenized data is then passed to the *NE Recognizer*. The *trained model* in collaboration with the *stored rules* will assist the NE Recognizer to perform such tasks as invoking stored features, NE tagging and chunking on the input data. The NE Recognizer recognizes NEs out of input text with the aid of both the trained model and the stored rules. The result, NE recognized Afan Oromo text, is then supplied as an output.

4.4 Phases of AONER

In AONER architecture, a phase is just a module that holds related components together. Components are related to each other based on the task they perform, their respective input and their general contribution in a process.

4.4.1 Pre-processing Phase

The pre-processing phase is the phase where necessary pre-processing tasks are performed on the input data, the training corpus and the plain text, so that it can effectively be utilized by the rest of the phases. Parser and Tokenizer are the components placed in the pre-processing phase.

4.4.1.1 Parser

During the development of the corpus, we first went through all the obtained news articles and selected those sentences that have at least one NE. The obtained sentences are then tokenized and each token was tagged with its NE tag in accordance with the CoNLL"s 2002 standard. NE tagging during this time was done manually. This makes the task prone to errors. Some of the errors that might arise are errors like failing to be consistent with the rules of CoNLL and mistyping. If the data is victim of these errors, it will mislead the other tasks that depend on the data or even it will halt the progress into the next component. Parser is the essential component that checks the NE tagged training corpus and report if there are errors.

It mainly checks the NE tagged data against the CoNLL‟s 2002 rules and standards. We used CoNLL‟s BIO scheme for tagging the training data. CoNLL has (Erik 2002) developed a legal sequence of tags which every NE corpus should abide by. The Parser will check the data whether it is legally tagged or not. For example, for the following two tags created for the sentence "*Yunivarsitiin Amboo magaalaa Ambootti argama*.", the tagging on the left is legal while the tagging on the right violates the rule of CoNLL 2002 standard. Because, it has to be preceded by B-LOC tag or it has to be B-LOC.

Yunivarsitiin B-ORG	Yunivarsitiin B-ORG
Amboo I-ORG	Amboo I-ORG
Magaalaa O	magaalaa O
Ambootti B-LOC	Ambootti I-LOC
Argama O	argama O
. O	. O

Once the training data is checked by the parser and everything is found to be correct, the data will be made ready for generating the tokens and the tags, token/tag sequence, which is done by the BIO Encoder.

4.4.1.2 Tokenizer

Tokenization can basically be defined as the process of breaking up a text into its constituent elements, *tokens*. Tokenization can occur at a number of different levels: a text could be broken up into paragraphs, sentences or words. For any given level of tokenization, there are many different algorithms for breaking up the text depending on the intention of the system and the language under consideration. For example, at the word level, it is not immediately clear how to treat such strings as "can't," "$22.50," "New York," and "so-called". Having these confusing facts beside, mostly in languages like English, tokenization at the word level is done based on white space.

In Afan Oromo, tokenization is a trivial problem as its writing fashion is identical to English since words are separated by a white space. In fact, there are circumstances in which two

words are treated as a single word when they are joined to each other by a hyphen (-) and they are commonly called „*jecha tishoo*". For example „*gara-jabeessa*" and „*ɓu'a-qabeessa*" are treated as a single word. As shown in the architecture of the system, tokenization is done on the input plain text during the prediction process. Tokenization on the training corpus is done during corpus development. Tokenization for the input text is performed by a component called tokenizer. The tokenizer takes the input text supplied from a user and tokenizes it into a sequence of tokens that can make it easy to recognize NEs. Token is explained in detail in the implementation part of our system (section 5.2.2). The tokenizer will treat double words that are connected to each other by a hyphen (-) as separate words and the hyphen mark is treated as a token. Though a hyphen does not have a direct effect on NEs, treating it as a token will enable us group all punctuations under a single category. For example: the word „*gara-jabeessa'* has three tokens. The tokenizer treats words which contain apostrophe („) as a single word as in „*Ya'aaballoo*".

4.4.2 Training Phase

Training phase comprises necessary components that are used in the process of training the machine learning component. The components within this phase operate on the data from the training corpus and they are the *BIO Encoder*, *NE Chunker*, *Feature Extractor* and the *Model Builder*.

4.4.2.1 BIO Encoder

The training data will be tagged in such a manner that each line consists of a token and its NE tag. Though it is easy and straightforward to understand this as a user, both *NE Chunker* and *Feature Extractor* must know, from the training data, which one is a token and which one is a tag. The task of BIO Encoder is to analyze and then identify a token and its tag. After doing so, the encoder generates a token/tag sequence that will be supplied to both feature extractor and NE chunker. The process of generating token/tag sequence is called *Encoding*. Basically, an encoder needs a tokenizer to detect the boundary of a token and its tag. The tokenizer has to be consistent and compatible with the tokenizer used during corpus development. Otherwise it may wrongly detect the tokens and their tags. For this reason, the *tokenizer* that

was used during the NE corpus development is reused here by the encoder. As a result, we found it infeasible to place the tokenizer as a component on the architecture.

4.4.2.2 Feature Extractor

Features are properties of a text that are used to provide necessary information associated to a given NE and increase the confidence level of predicting a token as a NE. AONER‟s feature extractor is responsible for identifying and extracting all the necessary features from the training data. It is one of the essential components that supply necessary information (features) during the building of the model. For example in a sentence: "*Obbo Barkeessaan bor dhufu*.", The word „*Obbo*‟ and the case of the word „*Barkeessaan*‟ and its suffix „*n*‟ can be used as features to identify „*Barkeessaan*‟ as a person name. The feature extractor is designed to extract features from the training data and store (cache) them for future use. The features are extracted from an input sequence of tokens those supplied by the encoder‟s tokenizer and the tags those supplied by the encoder.

Since we used CRF algorithm, the feature extractor perform repeated extractions from the same context for different positions in the input. All the extracted features are supplied to the model builder which will estimate the parameters of the model. The features used in our study to identify Afan Oromo NEs include normalized tokens, part-of-speech tags, word-shape features, position features, and token prefixes and suffixes, all in a window of one token around the current token. The part-of-speech of the tokens is generated by a Part-Of-Speech Tagger (POST) which is discussed in one of the subsequent sections. Some of the possible feature spaces that can be used to identify NEs in Afan Oromo are listed under appendix A.

4.4.2.3 NE Chunker

The token/tag sequence supplied by the BIO encoder has to be chunked before building the model. This means, those sequences of tokens that follow each other and belong to similar category has to be considered as a phrase and assigned a single tag specifying their category. NE Chunker is designed to perform this task by detecting phrase boundaries. It creates a chunk set from a sequence of tagged tokens. This process is called *Chunking*. Chunking

combines two or more continuous NEs that have identical tag (with the first B and the successive Is) and tag them with a single NE tag. Chunking is helpful for those NEs that comprise two or more words. For example, let us consider a person name „*Abbaaduulaa Gammadaa*" which is composed of two words. Following the BIO tagging scheme „*Abbaaduulaa*" will be tagged as B-PER and „*Gammadaa*" is tagged as I-PER. When this is provided to the chunker, it understands that both words follow each other and also belong to the same NE type, PERSON. Based on this fact, it combines them together and creates a single chunk: „*Abbaaduulaa Gammadaa'* with a tag PERSON. The sequence of chunks will be handed to the model builder in parallel with extracted features.

4.4.2.4 Model Builder

Our main concern with the machine learning component is to build a trained model. All the previously discussed components in the training phase perform a task of generating required information that is used for the actual training procedure. The procedure of training the model is done by the model builder. Model builder is the component designed to estimate the model coefficients λ_k (section 2.4.2) and then build a trained model. This means it trains a CRF model. Training in this case consists of estimating the model coefficient based on the input from the training corpus that is supplied in the form chunking by the *NE Chunker*, and the extracted features that are supplied by the *Feature Extractor*. It might have other additionally required parameters. Since this is an implementation issue, we will show how the builder is implemented in Chapter 5. In general the model builder is designed to perform all the calculations that are discussed in section 2.4.2 and generates a trained model. The generated trained model will then be used in the recognition phase.

4.4.3 Recognition Phase

The recognition phase is the phase where the actual work of recognizing NEs from the tokenized plain text is done. Components included in this phase are the *Trained Model*, *NE Recognizer* and *Stored Rules*.

4.4.3.1 Trained Model

The trained model is what we look for as part of the machine learning component and it is the final output of the *learning process*. Since we used CRF algorithm in the training process and the model is trained with Afan Oromo data, we can call the trained model as *Afan Oromo CRF Model*. The model contains a set of values that are obtained from the calculations performed during the estimation. It corresponds to equation 5 of section 2.4.1. The trained model, here, is a set of parameters (known as weights) which corresponds to the importance of the features used in the task. It is an estimation of all the parameters that are obtained through training. The trained model is the very essential component that assists the *NE Recognizer* to recognize NEs from the tokenized plain texts. While doing so, it performs the task of *Inference*. Inference is the procedure of finding the most likely sequence of tags for the tokenized input texts as described in section 2.4.3. The model is the main component that plays a crucial role in supplying necessary information during NE recognition.

4.4.3.2 Stored Rules

Naturally, machine learning components require a huge amount of training data to perform well. Still with huge amount of training data one cannot be sure whether the data contains all the structure and organization associated to NEs. As a result, it can be possible to develop a reasonable amount of training data and then develop a set of rules that can work in collaboration with the trained model. The proposed architecture is designed based on this assumption. The stored rules are designed to focus on the general structure and organization of Afan Oromo NEs, so that they can cover those parts that are not covered by the model. The rules will be developed based on the idea of pattern matching. We have identified some of the words that are used to develop the pattern of NEs in Afan Oromo. They are found in appendix A. The rules are expected to assist the model in such ambiguities as when it gets equal probability for a token to classify it into different NE categories. By doing so, the rules will perform the task of disambiguation. The rules will act in the detection procedure of the recognition process.

4.4.3.3 NE Recognizer

NE Recognizer is the core component that takes a pre-processed input plain text from a user, recognizes NEs out of it and supplies NE recognized Afan Oromo text as an output. As

depicted on the architecture, it is assisted by both the trained model and the stored rules. AONER‟s NE recognizer performs two main tasks: *detection* and *classification*.

Detection is the first task which finds candidate tokens that can be NEs from the tokenized plain text. Based on the combined knowledge supplied from both the model and the rules, detection is performed as follows. The model, invoked by the recognizer, does a number of tasks turn by turn. The features that are extracted and stored during the training phase are supplied to the recognizer to identify NEs from the text. The model then selects candidate tokens based on the calculated probability. To check if there are wrongly detected tokens and there are found ambiguities by the model, the stored rules are used. This concludes the detection process.

The classification portion is also done by a joint work from the model and the rules. The recognizer starts the classification process by tagging the detected candidate tokens. Each token will be tagged with their possible NE tags using the BIO scheme. This is done based on the knowledge obtained from the model. The stored rules will react to check if the tokens are tagged correctly. If there are wrongly tagged tokens, it corrects. The recognizer then chunks those sequence of tokens with identical tag extension and make them ready for final output. This completes the classification part and the prediction process. Finally, the recognizer will generate NE recognized Afan Oromo text and send it as an output.

4.5 Afan Oromo POS Tagger (AOPOST)

Part-of-speech (POS) tagging is the process of choosing the correct grammatical tag for a word based on the context or morphological properties. The POS tags contain a significant amount of grammatical information such as quantity, person, and gender about the words and their neighbours (Jurafsky and Martin 2000). The task of assigning POS tag to words is accomplished by part-of-speech tagger (POST). The Input data to POST is the input text and the output of it is words accompanied by their POS tags. In NER, POST is helpful in such a way that it facilitates and simplifies NE tagging which focuses only on words with „noun‟ tag.

AONER uses POST that assigns words with their POS tags. POS is one of the features used by AONER. The POST used in our system is an external trained model. There are already two research works done on POS tagging for Afan Oromo; the works of (Getachew 2009 & Mohammed 2010). However, there is no fully implemented *Afan Oromo POST* so far. The first work was conducted using HMM. We tried to use and integrate it into our system. However, we found the codes to be unmanageable and burdensome. The second one is completely incompatible with our system due to the approach followed and the tool used. For this reason, we did additional work of developing our own POST. We developed our own Afan Oromo POST on LingPipe that can easily be integrated into our system. The model (POST) is trained with the corpus developed for the work (Getachew 2009). Since the size of the obtained training data is very small, we don‟t think it is reliable. We developed an additional Afan Oromo POS tagged corpus of size 4,588 words. The *POST* is called by *the feature extractor* to assign tokens with their possible POS tags.

4.6 Summary

The overall design of our system is discussed starting with the approach followed. Our study followed a hybrid approach with the machine learning component based on a CRF algorithm. We also presented the proposed architecture of our system. The system‟s architecture has *two* main processes: the learning and prediction processes. The *learning process* works on the training data and is used to generate the trained model. The *prediction process* is a process which works on the input text supplied by a user and is aimed at recognizing NEs from the text. The system also has *three* main phases. The *pre-processing phase* is where both the training data and the plain text are pre-processed for the next task. *Parsing* and *Tokenization* is the main *pre-processing* tasks with the former applying to the corpus and the later to the plain input text. The *training phase* comprises essential components that are used to generate token/tag sequence, extract features, chunk the tokens and estimate the model with the training data. The *recognition phase* is where the pre-processed input text is an input and NE recognized text is an output and it consists of components like the trained model, stored rules and NE Recognizer.

Chapter Five: Implementation of AONER

In this chapter we will present the algorithms by which the components in the architecture are implemented. Though the proposed architecture has two components, machine learning and stored rule, *we implemented the machine learning part*. The rule-based part needs collection and organization of rules that can be plugged into the machine learning component and its implementation is left as future work due to limitation of time. The sections are organized based on the flow of tasks within phases as described in the architecture. First we will give a brief explanation regarding the NE corpus used. Then we will discuss how the components in each phases are implemented starting with the pre-processing phase through the training phase to the recognition phase.

5.1 Corpus Development

In this section we will explain how the Afan Oromo NE corpus is developed by first giving the detail about the NE categories that are being identified and used in this research work. We will also discuss the nature and size of the corpus being developed.

5.1.1 AONER NE Categories

The first work before implementing our system was deciding the categories of NEs that can be recognized by the system. Adopted from CoNLL, four categories of NEs are identified: *Person*, *Location*, *Organization* and *Miscellaneous*. Though there is no specific rule by CoNLL what could be included in the miscellaneous category, we decided to include the *date/time*, *monetary values* and *percentages* under our miscellaneous NE category. These three entity types are defined by MUC and it makes sense if we include them under the miscellaneous category. The description of each NE category is given in Table 5.1.

Regarding the organization entity, we decided to include the clue word specifying the type of the organization as part of the entity itself. The decision came from the fact that most organization names are given after person names or based on where they are situated. This

creates confusion with both person names and location names. Including the clue words makes it more informative and avoids confusion. For example the phrase „*Yunivarsitii Addis Ababaa*" is more informative as an organization name than „*Addis Ababaa*".

Table 5.1: AONER's NE Categories

Category	**Remark**	**Example**
Person	Name of a person without his/her title	Gammachuu, Ittiyaannee, Baaraak, Faqqadee
Location	Places that are used to indicate where an entity is found or an event occur	Oromiyaa, Najjoo, Malkaa Sadii, Itoophiyaa
Organization	A social arrangement which pursues collective goals, controls its own performance, and has a boundary separating it from its environment.	Yuunivarsitii Addis Ababaa, Baankii Hojii Gamtaa Oromiyaa, Warshaa Simmintoo Mogor
Miscellaneous	Includes three entities: date/time, Percentage and Monetary values	1922, Miliyoona 1.3, 98, %72, 3:30, $120.

5.1.2 Afan Oromo NE Corpus

Generally speaking, there is no authorized and publicly available tagged Afan Oromo text for any work of NLP. This holds true for NER also. As a result, we developed Afan Oromo NE corpus. Our first task before developing the corpus was deciding the domain of our study. We decided to work on the news domain. Accordingly, Afan Oromo news texts were gathered from two state owned Afan Oromo news papers: „*Bariisaa*" and „*Kallacha Oromiyaa*". A total of *4014 articles* were collected from both.

Our target is on sentences having populated NEs. However, most of the sentences from the collected articles do not have NEs at all. Directly using those sentences for corpus development is of no use rather than adding a burden work of tagging. As a result, we went through all the sentences in each article and those sentences having at least one NE were chosen for corpus development.

At the end, a text of more than 23,000 words was chosen for corpus development. The data was first tokenized in accordance with the consistent tokenization policy. In order to develop the corpus, we followed *CoNLL's 2002* format (Erik 2002). According to this format all data files contain one word per line with empty lines representing sentence boundaries. Each line also contains a tag which states whether the word is inside a named entity or not. The tag also encodes the type of named entity. Each token is separated from its tag with only a single white space. Here is a sample taken from the developed Afan Oromo NE corpus.

```
Ispeen B-LOC
magaalaa O
Siiviillaatti B-LOC
Moosees B-PER
Kipikiroofi I-PER
Taarikuu B-PER
Baqqalaa I-PER
tokkoffaafi O
lamaffaa O
ta'nii O
mo'ataniiru O
. O
```

This sentence has four NEs: *Ispeen* and *Siiviillaatti* are locations; *Moosees Kipikiroofi* and *Taarikuu Baqqalaa* are persons. Words tagged with O are not NE. The B-X tag is used for the first word in a named entity of type X and I-X is used for all other words in NEs of type X. Following this standard and considering the NE categories used in this study, a total of *9 NE tag sets* were identified and used for tagging the corpus. These are: *B-PER*, *I-PER*, *B-LOC*, *I-LOC*, *B-ORG*, *I-ORG*, *B-MISC*, *I-MISC* and *O*. Each word is tagged with one of these tag sets and the NE corpus was developed.

The total corpus file is divided into four files out of which three files are to be used as per the rules and regulations of the CoNLL 2002 shared task (Erik 2002). The four files are named as *the training file*, *the development file*, *the test file* and *the experimentation file*. The learning methods are trained with the training data. The data in the development file is used for tuning the parameters of the learning methods. When the best parameters are found, the method can be trained on the training data and tested on the test data. Here, the split between development and test data has been chosen to avoid that systems are tuned to the test data. The experimentation file is used later in the experimentation. The statistics of the developed Afan Oromo NE corpus is given in Table 5.2.

Table 5.2: Statistics of Afan Oromo NE corpus

	Total Size	**Total NE**	**PER**	**LOC**	**ORG**	**MISC**
Training File	8000 tokens	1206	411	266	296	233
Development File	5000 tokens	745	167	248	183	147
Test File	4000 tokens	704	234	235	91	144
Exp. File	6100 tokens	920	252	346	221	101

5.2 Pre-processing phase

Both the corpus and the plain text require pre-processing operations. For the corpus, pre-processing operations are required for the corpus was developed manually. Using it directly without processing will generate errors. The corpus has to be parsed before being utilized by the other tasks relying on it. The plain text supplied by a user has to be tokenized. In this section, we will explain how both the pre-processing operations are implemented in AONER.

5.2.1 Parsing

Manually developing the corpus makes it vulnerable to errors. One of the errors that can occur is the illegal tagging of the words. BIO scheme"s legal tag sequence is described on Table 5.3 where the first column lists the representation of the tags and the second column lists the tags that may follow them, with the variables ranging over all NE types.

Table 5.3: BIO legal tag sequence

Tag	Legal Following Tags
O	O, B-X
B-X	O, I-X, B-Y
I-X	O, I-X, B-Y

Table 5.3 shows that during the legal tagging of the BIO scheme *begin(B)* and *in(I)* tags have the same legal followers. B-Y is to refer to the first word of another NE type. In addition to illegal tagging, other errors might occur like:

- The distance between a token and its tag might be larger than a single white space
- There might not be a space between a token and its tag
- A token might be left untagged

The corpus has to be parsed and checked against all of these. Failing to do so makes the BIO Encoder to wrongly treat the tokens and their tags during the process of encoding. This in turn results in faulty feature extraction, wrong parameter estimation, and generates a faulty model.

Parsing is the crucial algorithm for checking the occurrence of the aforementioned errors and report if found. The corpus is parsed by the parser. The implementation algorithm of parser as implemented in AONER is given in Figure 5.1.

5.2.2 Tokenization

Tokenization is found at different phases of the AONER system. As discussed in section (5.1.2) the row data selected for corpus development were first tokenized and then tagged. During BIO encoding, the applicable tokenization policy is required to identify the tokens and their tags. The input plain text that is supplied by a user has to be tokenized before it is handed to the recognizer. On the other hand, the NE chunker needs to know the tokenization policy in order to detect the boundary of chunks since each chunk starts and ends on a token

start or end position. Since the phases are dependent on each other, the overall tokenization policy used in each case has to be the same and compatible.

```
For all lines in the file
    Read a line
        If a word is not tagged
            Report error
        If the space b/n a word & its tag is more than a single space
            Report error
        If the previous tag is O
            Check if the current tag is one of O & B-X
        Otherwise report error
        If the previous tag is B-X
            Check if the current tag is one of O, I-X, B-Y
        Otherwise report error
        If the previous tag is I-X
            Check if the current tag is one of O, I-X, B-Y
        Otherwise report error
End For
```

Figure 5.1: AONER's parsing algorithm

In this work, tokenization is implemented to mainly occur at word levels. But there is a special treatment for punctuation marks. Generally, what we mean by a token is just a sequence of characters satisfying the pattern in the Table 5.4 which is adopted from (Carpenter & Breck 2010). This pattern is used as a baseline for the tokenization policy used in this study.

Table 5.4: Patterns of tokens as used by AONER's Tokenizer

Pattern	Description
AlphaNumeric	Any sequence of upper or lowercase letters or digits.
Numerical	Any sequence of numbers, commas, and periods.
Hyphen Sequence	Any number of hyphens (-)
Equals Sequence	Any number of equals signs (=)
Double Quotes	Double forward quotes (``) or double backward quotes('')

As Afan Oromo writing style[15] is similar to that of English, mainly it is the white space that demarcates the boundary of a token. With regard to NER, it is important to treat punctuation marks like periods, commas, hyphens, equal sign and double quotes as a token. This stems from the fact that they significantly affect NE tagging. Consider the following sentence:

Pirezedaantiin Ameerikaa duraanii Ji'oorgi W. Buush jedhamu.

(*The former American president is called George W. Bush*).

In the above Afan Oromo sentence, which has 9 tokens, the period (.) in front of the letter *W* is part of NE (with tag I-PER). But the period at the end of the sentence is not part of NE (with tag O). All punctuation marks (except the apostrophe character) have been treated as tokens in our tokenization policy. The apostrophe (‚) character (called ‚*hudhaa*') is frequently seen in Afan Oromo words to specify missed consonant. If this character is treated as a token, then it will divide a single word into three different tokens with each token having no meaning. As a result, our tokenization algorithm ignores apostrophe mark and considers the word as a single token. For instance, words like *Ya'aaballoo*, *Saamu'el*, *Raadi'eel* are each treated as a single token.

[15] It is not the writing system. It is rather to say the writing pattern.

5.3 Training phase

Once the corpus is parsed and everything is successful, it will be passed to the training phase. The main tasks carried out in the training phase are:

- *BIO encoding* - generates a token/tag sequence from the parsed corpus that will be used in the feature extraction and NE chunking.
- *Feature extraction* - extracts features for each token in the training data.
- *NE Chunking* - creates a chunk from a sequence of tokens with similar NE type extensions
- *Model building/generation* - estimates the model parameter values and generate a trained model.

5.3.1 BIO Encoding

We have previously presented that the parsed corpus is a correctly tagged data with each line consisting of a token and its corresponding tag. The parser does not have the knowledge about which one is a token and which one is a tag. Identifying a token and its tag is essential for feature extraction and NE chunking. BIO encoding is the process of detecting the boundary and identity of a token and its tag. It also constitutes generating the token/tag sequence. The BIO encoding process is handled by BIO encoder component from the architecture.

The encoding procedure has of two steps: *detection* and *generation*. The first thing in the encoding process is to inform the encoder the format and identity of the available tag sets. We have discussed in *section 5.1.2* that we have identified and defined a total of 9 NE tag sets for this study. The detection step begins by detecting the boundary of a token. To do so, the encoding procedure needs to know the tokenization policy and this policy must be consistent with the one used during the corpus development. We have developed an internal tokenizer for the BIO encoder to enable it detect the tokens. The encoder detects the tags by looking at the stored tag sets. Once the tokens are detected the generation step commences. During the generation step the encoder generates the token/tag sequence. The generation of

token/tag sequence will ease the extraction of features and chunking. For easy understanding, the generated token/tag sequence by AONER's BIO Encoder looks like as in the Table 5.5.

Table 5.5: Token/Tag sequence as generated by BIO Encoder

Token	Tag
Godina	O
Iluu	B-LOC
Abbaa	I-LOC
Booraatti	I-LOC
magaalaa	O
Baddallee	B-LOC
bu′uuraaleen	O
misoomaa	O
adda	O
addaa	O
qarshii	O
miliyona	B-MISC
17n	I-MISC
hojjatamaa	O
jiru	O
.	O

5.3.2 Feature Extraction

We used a special type of CRFs algorithm known as *chain CRFs* to implement AONER. Feature extraction for chain CRFs is complex because of the need for repeated extractions from the same context for different positions in the input and different previous tags

(Carpenter & Breck 2010). Feature extraction for AONER is implemented based on the idea that features are extracted and will be cached for later use. Feature caching is required because of the lattice required for forward-backward decoding. Feature extraction is handled by feature extractor component.

Feature extractor was implemented as follows. First it takes an input sequence of tokens and set of possible tags to produce a set of features. The tokens are those supplied by the encoder"s tokenizer and the tags are supplied by the encoder"s coding of taggings. The extracted features in turn provide feature maps for nodes and edges. *Node features* are features based on the *input tokens* and *position* while the edge features are features based on the *input tokens*, *position*, and a *previous tag*. Naturally, CRFs require N node feature vectors and K*(N-1) edge feature vectors, where K is the number of possible tags and N is the number of tokens (Carpenter & Breck 2010). The feature map is constructed once for a given input and then used to populate the lattice required for decoding with feature vectors. This allows operations performed on the input sequence to be cached and reused. For instance, part-of-speech tags would be computed once at features object construction time and then reused in calls to get node or edge feature maps. LingPipe provides special interface for chain CRF feature extractors, `crf.ChainCrfFeatureExtractor`. The feature extractor then produces an instance of `crf.ChainCrfFeatures`.

Feature extraction for AONER is done in a range of a window of one token around the target token. This means, while extracting features for a token only each token residing to the left and right of that token are considered. The features used in our study include *word-shape features*, *position features*, *part-of-speech tags*, *normalized tokens*, and *prefixes* and *suffixes* of a token. The features have a binary magnitude of either 1 (one) or 0 (zero). A magnitude of 1 is assigned to a feature if that particular feature is active and 0 is assigned if the feature is not active for the token under consideration. For example: a position feature called BOS will have 1 if the current token is at the beginning of a sentence or 0 otherwise. The description of the feature sets used in AONER is given as follows.

Word-shape features: These groups of features include those features associated with a token‟s identity based on character "shape". They include such features as the Case, punctuation and digit of a token. In this study we adopted all the character shapes that were defined by LingPipe. The complete list of word-shape feature values that can be extracted for a token as used in AONER is listed in appendix D.

Position features: These are features associated with the position of a token in a sentence. They include features like beginning of a sentence, inside a sentence and end of a sentence. All other features are dependent on the value of position features. For example, Word shape feature is not extracted for a previous token if the current token is at the beginning of a sentence.

Part-of-speech tag features: These are features associated with the grammatical tag of a token based on the context or morphological properties. These features are extracted based on the tag that is assigned by an external model, AOPOST. We used the tag sets developed by (Getachew 2009). Based on the description given on the paper NE tokens will have one of NN, NP, NC and AX tags.

Normalized token features: Most of the tokens under the miscellaneous category follow similar pattern. For example date is mostly written as 26/8/10. Normalizing such tokens will ease categorization. Normalized token features are features designed to perform some light normalization of tokens to compress all numbers to the same kind of value. For instance, 26/8/10 can be converted to *DD*/*D*/*DD*.

Prefix and Suffix features: These are features associated with the extraction of prefix and suffix of a token. These features are extracted based on the specified constant length of prefixes and suffixes. First the maximum possible length of both prefixes and suffixes are set. From the training data we found that the maximum length of suffixes in Afan Oromo is 6. Prefixes are rarely observed in Afan Oromo and by default we set their maximum length to 3. The length of a token is first checked for both of these features to be extracted. These features are extracted for those tokens having larger length than the above specified length of prefixes and suffixes.

All the above features will be extracted for each token in the training data. The obtained values for each token will be appended and stored along with the token. When a feature

extraction is over, the obtained result will be sent to the model builder for parameter estimation. Assume we have a method called *fExtractor* which extract features. The feature extraction algorithm works as shown in Figure 5.2. The algorithm for extracting prefix features is the same as suffix features except changing the naming.

For *all tokens in the training data*

Activate *position feature*

If *the current token is BOS*

fExtractor.append(position feature as BOS and set as 1).

If *the current token is EOS*

fExtractor.append(position feature as EOS and set as 1).

Activate *word shape feature*

fExtractor.extract(word shape feature for the current token and set as 1).

If *the current token is not BOS*

fExtractor.append(the word shape feature of the previous token and set as 1).

If *the current token is not EOS*

fExtractor.extract(the word shape feature of the next token and set as 1).

Activate *normalized token feature*

fExtractor.extract(normalized token feature for the current token and set as 1).

If *the current token is not BOS*

fExtractor.append(normalized token feature of the previous token and set as 1).

If *the current token is not EOS*

fExtractor.extract(normalized token feature of the next token and set as 1).

Activate *POS tag feature*

fExtractor.extract(POS tag feature for the current token and set as 1).

If *the current token is not BOS*

fExtractor.append(the POS tag of the previous token and set as 1).

If *the current token is not EOS*

fExtractor.extract(the POS tag of the next token and set as 1).

Activate *Suffix feature*

Based on the specified maximum length of suffixes

If *the length of the current token is less*

Ignore suffix feature for the current token.

Otherwise *fExtractor.extract(suffix feature for the current token and set as 1).*

If *the current token is not BOS*

fExtractor.append(suffixes of the previous token and set as 1).

If *the current token is not EOS*

fExtractor.extract(suffixes for the next token and set as 1).

End For

Return fExtractor( ).

Figure 5.2: AONER's NE feature extraction algorithm

5.3.3 NE Chunking

Parameter estimation during model generation requires a corpus of chunking. A special type of chunking used in our work is NE chunking which creates phrases known as chunks from a sequence of tokens. It is done by NE chunker. NE Chunking in AONER is implemented in such a way that first the token/tag sequence generated by the BIO encoder will be examined. After being examined, the tokens are converted into a character sequence with each character in the sequence given a position number. When converting into a character sequence, the final position number will be the position of the end character plus 1. For example a sentence can be converted to a character sequence as follows.

```
Obbo Abbaa Jifaar bulchaa Jimmaa turan.
0123456789012345678901234567890123456789
```

Changing the tokens into character sequence helps to compute the range covered by a phrase. This in turn is helpful for representing the chunk sets.

Once the input tokens are represented in character sequence, those tokens with BI tag sequence will be combined to create a chunk. This means from the sequence of tokens, those tokens following each other with tag sequence of the form *B-X I-X I-X* ... will be combined together.

With regard to our work, chunks can be defined as phrases belonging to one of the four NE categories. The obtained chunks will be assigned a tag that was taken from the extension of the constituent BI sequences. Chunks are represented by *chunk sets* which store the position of the chunk in the character sequence. Chunk sets can also represent the length of a chunk by specifying the start and end of the chunk in the character sequence. Chunk sets also store the entity type of the chunk. If a token sequence is not of the form BI sequence, i.e. it has the tag sequence of the form B-X B-Y, then a single token will treated as a chunk. The chunk will be assigned a tag similar with the extension of the B tag. For instance, if the sentence in the previous example was in the training data, it will obviously be tagged as:

```
Obbo(O) Abbaa(B-PER) Jifaar(I-PER) bulchaa(O) Jimmaa(B-LOC)
turan(O) .(O)
```

NE chunking the above sentence based on the character sequence information provided by the previous example results in two chunk sets of the form *(5,17):PER* and *(26,32):LOC*. The numbers in the bracket specify the start and end [plus 1] character offset of the chunk set and the tags after the colon specify the NE type. Since our concern is chunking NEs, those tokens with tag O will not be chunked. Later in the parameter estimation, tokens with no tags are by default given a tag O. NE Chunking require a form of tokenization to detect the boundary of tokens in the chunk. The consistent tokenization policy which was used across the system helps for the detection of tokens. The algorithm that is used to chunk NEs in AONER is shown on Figure 5.3.

```
For all token/tag sequence generated by the encoder
    Change the token sequence into character sequence
    If a token has O tag
        Ignore;
    For those tokens with tags of the form BI sequence
        Read the tokens until the next tag is B
        Combine the tokens and create a chunk
        Store the start and end+1 of the chunk in the sequence
        Assign the chunk a tag of the extension of B
        Create a chunk set of the form (start,end+1):tag
    End For

    For those tokens with tags of the form BB sequence
        Treat each token as a chunk
        Store the start and end + 1 of the chunk in the sequence
        Assign the chunk a tag of the extension of B
        Create a chunk set of the form (start,end+1):tag
    End For
End For
```

Figure 5.3: AONER's NE Chunking algorithm

5.3.4 Building the model

Model building/generation, can be referred to as *training*, is the procedure of estimating the parameter value that gives the best possible prediction for each training examples in the training data. Estimating parameters from data requires numerical optimization. AONER is implemented based on a linear chain CRF algorithm. The *model builder* finds the value of the weight vector, λ, for the linear chain CRF model. We have clearly shown in section 2.4.2 how training a CRF model is done. There are a number of popular methods of training a CRF model like *Stochastic Gradient training* (Elkan 2008), *Gibbs Sampling* (Finkel et al. 2005)

and *generalized iterative scaling* (Sha & Pereira 2003). All of them commonly work by maximizing the log-likelihood (section 2.4.2) of a given training set $T = \{(x_k, y_k)\}_{k=1}^{N}$:

$$\bar{L}(\tau) = \sum_{(\vec{x},\vec{y})\in\tau} \log p(\vec{y}|\vec{x})$$

where x_k represent the tokens and y_k represent the corresponding tag. LingPipe employs a special type of stochastic gradient training called *stochastic gradient descent*. Stochastic gradient descent makes several passes through the data, adjusting the parameters a little bit based on examples one at a time.

Estimating the model parameters require two main inputs. These are the training data and the extracted features. The features are in the form of node and edge features for estimation and tagging. The training data will be in the form of a corpus of chunking which was resulted from NE chunking. During estimating the parameters, the corpus will be treated as a corpus of tagging. This is done based on the specified string-based tokens and tags, an underlying character sequence, and arrays representing the position at which each token starts and ends.

There are also other local arguments that are needed to be organized before the actual operation of estimating the parameters commences. These arguments are required for stochastic gradient descent training algorithm. They are the tokenization policy, feature count, prior value, annealing schedule and epoch value. These arguments are described as follows.

Tokenization scheme: During estimation, the training data will be represented as a corpus of tagging. During that time the tokenization policy is used to turn an input sequence into a list of tokens.

Feature count: There is unbounded relaxation in feature selection in CRF as repeated extractions are allowed from the same context for different positions in the input sequence. This argument specifies the minimum number of times a feature must show up in the tagging corpus given the feature extractors to be retained for training. This is set to 1 for AONER.

Prior value: Priors are used to control the regularization or lack thereof used by the stochastic gradient descent optimizers. The priors all assume a zero mean (or position) for each dimension, but allow variances (or scales) to vary by input dimension.

Epoch: In stochastic gradient descent the training examples are sorted in random order, and the parameters are updated for each example sequentially. Epoch refers to one complete update for every example. LingPipe provides the opportunity of defining the minimum and maximum number of epochs for which to train. For AONER we set the minimum epoch to 10 and the maximum epoch value to 5000.

Annealing schedule: Annealing schedule determines learning rates for stochastic gradient descent training. LingPipe provides three types of annealing schedule. These are Constant Learning Rate (always returns the same learning rate, which is fixed at construction time), Exponential Decay (the learning rate undergoes exponential decay starting at the initial learning rate and decaying exponentially at a rate determined by the base of the exponent) and Inverse Scaling (lowers the rate more quickly than the exponential rates initially and then more slowly for later epochs). We used an exponential decay and a fairly high initial learning rate for AONER.

In addition to all the above arguments, the model builder needs other information like whether to cache the features, the value of minimum improvement in epoch to terminate the training. Caching features is critical because of the lattice for the forward-backward decoding. As a result, we allowed the estimator to cache features. After all the above discussed inputs and arguments are set, the model generation step begins. The parameter is estimated by the LingPipe‟s `ChainCrfChunker.estimate()` method. The completion of estimation procedure outputs a CRF model trained with Afan Oromo texts. So we can name the trained model as *Afan Oromo NE CRF model*. In our development environment, the estimation procedure took an average of around *23 min and 20 sec*.

5.4 Recognition Phase

As part of the machine learning component, the generated model (trained model) assists the recognizer to recognize possible NE tokens from the tokenized input text. The model performs the task of inference during this time. We have discussed, in section 2.4.3, how

inference in CRF is computed. Inference in NER task is a phenomenon associated with finding the most likely sequence of NE tags (labels) given plain texts (observation). This means recognizing NEs from the input tokens. In this section we will explain how the model recognizes NEs from the tokenized input text.

5.4.1 Recognition

The task of the recognizer in the recognition process is to convert the tokenized input plain text into character sequence with each character assigned a position number in the sequence. The actual recognition process is handled by the model. AONER"s trained model contains coefficients computed from the training data, extracted features, tokenization policy and a set of applicable NE tags. These are used to compute the probability of the input texts during the recognition. The model recognizes the possible NEs from the tokenized input text using *Viterbi's forward-backward algorithm*. In this algorithm, first a probability will be computed turn by turn for each token in the input text against the available NE tags. The obtained probability value for each token will be stored along with its tag identity. When the computation is done for all tokens, the forward algorithm completes. A move is now done in reverse direction to sketch a path that passes through points with a large probability value. This is called a backward algorithm. When the backward algorithm is finished, each token will be assigned with a tag that obtained a larger probability value. This all is done by the model. Since we used the BIO tagging scheme, the tag given to the tokens was in the form of B-X I-X B-Y O…

Once the tokens obtain their possible tags, the recognizer does the next task as follows. Those tokens tagged with tag sequence of the form B-X I-X will be chunked. In the sequence, if two tokens that follow each other got different tags (of the form B-X B-Y) each will be treated as a chunk. The rest of tokens with O tag will be ignored.

Two parameters will be stored along with the created chunks. These are

- The tag of the chunk (one of PERSON, LOCATION, ORGANIZATION and MISCELLANEOUS) which is obtained from the extension of B during chunking
- The distance covered by the chunk this is obtained from the character sequence position.

5.5 Summary

Although the proposed architecture has two components, we implemented the machine learning component only. We developed the Afan Oromo NE corpus that consists of more than 23,000 words out of which around 3600 are NEs. The corpus is developed following CoNLL‟s 2002 standard. Four NE categories have been identified and implemented in this research work. These are PERSON, LOCATION, ORGANIZATION and MISCELLANEOUS categories.

The overall implementation of AONER has a total of seven sub-tasks. The Tokenization task is implemented by setting up a policy that will be consistent and compatible across all the system. This is required because of the reason that tokenization is required within many of the components of the system. According to the policy every word and punctuation marks are each treated as a token. However, apostrophe („) character is not treated as a token. Parsing is the task of parsing the corpus toward checking its correctness and reliability. It also checks the corpus‟s format against CoNNL‟s rules and regulation. Once the corpus is correct it is handed to the BIO Encoding task.

BIO Encoding is the task required to identify which word in the corpus is a token and which one is a tag. It detects the tokens based on the tokenization policy used. This information is required by during feature extraction and NE chunking. The tags are already stored. The BIO encoding then generates the token/tag sequence. Feature extraction task is associated with extracting features from tokens in the training data. For this research work, we used word shape, position, POS, normalized token, prefix and suffix features. The features are extracted based on the position of a token within a window size of one. NE chunking task is

implemented by first changing the generated tokens into a character sequence which gives each character in the token a position number. Those tokens with B-X I-X tag sequence will be combined into a single phrase called chunk. Those tokens of the form B-X B-Y will each be treated as a single chunk. Each chunk will be given a single NE tag taken from the extension of B tag. Each chunk also be given a start and end position number to represent the distance covered by it.

The model generation is implemented by inputs from NE chunking and Feature extraction using Stochastic Gradient Descent algorithm. The model is generated by making several passes through the data, adjusting the parameters a little bit based on examples one at a time. This outputs the trained model. The recognition of possible NEs from the plain text is implemented using the forward-backward algorithm. The recognition outputs the result in the form of chunks.

Chapter Six: Experimentation of AONER

This chapter is dedicated to the discussion of the experimentation aspects of our system, AONER. Besides our main objective of developing Afan Oromo NER system, we have developed a CRF based Afan Oromo POST (AOPOST) model. Its performance will be discussed in this chapter. First we give a summary of the experimentation environment. We will also give the evaluation metrics that are used to evaluate and experiment the performance of both models. The general overview regarding the nature and statistics of the test data prepared to evaluate the performance of both AOPOST and AONER will then follow. The experimentation conducted on AONER is done by setting up a number of evaluation scenarios. The procedures and the outcomes of each scenario will be explained in detail.

6.1 Experimentation Environment

The prototype was developed using java programming language. The language processing components of AONER are implemented and the AOPOST model is developed using an external java based API called *LingPipe*. A standard java classes from *java.io* and *java.util* packages were used in collaboration with LingPipe libraries to perform such tasks as accessing files, dealing with arrays and handling exceptions. We used NetBeans IDE 6.8 to develop both AONER and AOPOST.

Both systems are developed on a computer with the following specifications:

- Windows® 7 Ultimate™ operating system
- RAM size 2GB
- Hard disk size 250GB and
- Intel® Core™2 Duo Processor 2.20GHz.

6.2 Evaluation Metrics

LingPipe API provides a distinct evaluator to evaluate each tasks supported by the API. The evaluator requires a test data that is tagged in similar fashion to the training data. The general principle followed by the evaluator during performance evaluation is that, a manually tagged test data is supplied to the evaluator. The evaluator stores the tag of each token. The tokens are supplied to the model/system. The tokens are then tagged by the model/system. When the tagging completes, the output is supplied back to the evaluator. The evaluator crosschecks the output generated by the model/system against the corresponding manual tags that are previously kept and computes the performance.

For AOPOST, accuracy is used as a metric to evaluate the model. Accuracy calculates the percentage of correctly tagged tokens out of the total tokens. The confusion matrix is also computed and will be presented next.

$$\text{Accuracy} = \frac{\text{Number of correctly tagged tokens}}{\text{Total number of tokens in the test data}}$$

NER systems are commonly evaluated using three evaluation metrics: precision, recall and F-measure. All of them are represented in the form of percentage. Precision (P) calculates the percentage of correctly recognized NEs out of the total recognized NEs while Recall (R) calculates the percentage of the recognized NEs from the reference set[16]. Three values can help us easily calculate the evaluation metrics for our system. These are the True Positive (TP), False Positive (FP) and False Negative (FN).

- TP counts the number of NEs that are recognized by an NER system and are found in the test data.
- FP counts the number of NEs that are wrongly recognized by an NER system but are not in the test.
- FN counts the number of NEs that are left unrecognized by an NER system but are in the test data.

[16] Reference set is to refer to the manually tagged test data.

All TP, FN and FP are used to calculate Recall, Precision and F-measure. They are calculated as follows:

$$\text{Precision} = \frac{TP}{TP+FP} \qquad \text{Recall} = \frac{TP}{TP+FN}$$

F-measure (which is can also referred to as F1-measure) is the balanced harmonic mean of both precision and recall.

$$F1 = \frac{2(PR)}{P+R} = \frac{2TP}{2TP+FN+FP}$$

These values returned by the above equations are equivalent with the equations discussed in section 2.3.5.

6.3 Nature and Statistics of the Test data

The test data for both AOPOST and AONER is randomly taken from *Barissa* and *Kalacha Oromiyaa* news papers. We got POS tagged corpus data that was developed by (Getachew 2009). The corpus contains 159 sentences (with a total of 1621 words). This is added to the training data we have developed. We developed our own test data. The test data contains *65 sentences* which comprise *1088 tokens* (*1029 words* without punctuation marks). Each token in the test data was manually tagged with its POS tag by four Afan Oromo language experts (three *Afan Oromo MA* students and one *PhD* candidate). The experts were chosen based on their knowledge background on linguistic aspects of the language. The manually tagged texts are then cross checked for consistency.

The test data for AONER is prepared by the researcher. Each sentence in the test data contains at least 1 NE. The test data contains *212 sentences* which is totally *4000 tokens*. AONER"s test data contains a total of *704 NEs*, out of which *234* are *Person names*, *235* are *location names*, *91* are *organization names* and *144 are miscellaneous* NEs.

6.4 Performance of AOPOST

Evaluating the performance of AOPOST before integrating into our system is helpful to analyze its contribution for AONER. We used Afan Oromo tag sets developed by (Getachew

2009) which are 17 in number. Given the tag sets, the AOPOST model is tested on *1088 tokens*. Out of this, it correctly tagged *877 tokens* and it wrongly tagged the rest *211 tokens*. Accordingly, our AOPOST obtained an accuracy of *80.61%*.

We also computed the confusion matrix of AOPOST and it is shown in Figure 6.1. The confusion matrix is like a summary Table which is used to represent a reference/response relationship. The tags listed on the row represent tags given to tokens in the test data and those on the column represent tags responded by AOPOST. The numbers inside the matrix specify the cardinality of the tags on the row that are assigned to the tags on the column by the model. For example, out of the total „NN" tags in the test data, AOPOST assigned 2 as JN, 1 as AX, 12 as JJ, 200 as NN, 2 as AD, 3 as PP, 1 as PR and 6 as VV.

The obtained accuracy is actually more than 50%. From the description of each tags, obtained from (Getachew 2009), we understood that person, location and organization NEs have NN, NP and NC tags. The Miscellaneous NEs have AX POS tags. his has a positive impact to our system.

6.5 Performance of AONER

We divided the total Afan Oromo NE corpus into 4 different data. The name and size of each data is given as follows:

- Training data – 8000 tokens
- Development data – 5000 tokens
- Test data – 4000 tokens and
- Experimentation data – 6100 tokens

The system is first trained on the training data. The development data is used for tuning purpose to make the model converge when good parameters are found. The performance of AONER is calculated on the test data[17]. The experimentation data is used for experimentation being added to the training data to investigate the effect of increasing the training data.

[17] The statistics of AONER"s test data is given is section 6.3.

	II	JN	AX	JJ	NP	NO	NN	AD	PN	PP	PR	CC	PS	JC	VV	NC	PC
II	**1**	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
JP	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
JN	0	**5**	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
AX	0	0	**65**	0	**1**	0	**1**	0	0	0	**1**	0	0	0	**19**	0	0
JJ	0	**2**	**1**	**94**	0	0	**51**	**2**	**1**	**1**	0	0	0	0	**6**	0	0
NP	0	0	0	**1**	**2**	0	0	0	0	0	0	0	0	0	0	0	0
NO	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
NN	0	**2**	**1**	**12**	0	0	**200**	**2**	0	**3**	**1**	0	0	0	**6**	0	0
AD	0	**1**	**2**	**7**	0	0	**8**	**32**	0	0	**1**	**1**	0	0	**3**	0	0
PN	0	0	0	0	0	0	0	0	**106**	0	0	0	0	0	0	0	0
PP	0	0	0	**4**	0	0	**14**	**1**	0	**40**	**4**	0	0	0	**2**	0	0
PR	0	0	**3**	0	0	0	**5**	**1**	0	**2**	**84**	0	0	0	**1**	0	0
CC	0	0	**2**	0	0	0	**1**	**1**	0	0	**1**	**12**	0	0	0	0	0
PS	0	0	0	0	0	0	0	0	0	**3**	**1**	0	0	0	0	0	0
JC	0	0	0	0	0	0	**1**	0	0	0	0	0	0	**3**	**1**	0	0
VV	0	0	0	**6**	0	0	**11**	**1**	0	0	0	0	0	0	**219**	**1**	0
NC	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	**14**	0
PC	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0

Figure 6.1: Confusion matrix of AOPOST

AONER is first trained on the training data. The training data contains a total of around *8000 token*s out of which *1206 are NEs*. Out of the NEs in the training data *411* are *person entities*, *266* are *location entities*, *296* are *organization entities* and *233* are *miscellaneous entities*. When trained on this amount of training data and tested on the test data, AONER correctly recognized *545 NEs* out of *704 NEs*. These are *TPs*. It missed *159 NEs*. These are *FNs*. It wrongly recognized *174 NEs*. These are *FPs*. With the original training data AONER

obtained *Recall* value *77.41%*, *Precision* value *75.80%* and an *F1-measure* value *76.60%*. These values were used as references for experimentation scenarios to observe the effects those scenarios have on the performance of AONER.

6.6 Evaluation scenarios of AONER

We conducted experiments regarding the performance of AONER by setting up experimental scenarios. Two evaluation scenarios have been set up to investigate and observe the performance of AONER given the factors in each scenario. The first scenario is aimed at investigating the performance of AONER by increasing the training data. The second one is aimed at examining the influence of features on the performance of AONER.

6.6.1 Influence of increasing the training data

To observe the effect of increasing the training data size, we added the experimentation data (of size *6100 tokens* out of which *920 are NEs*) to the original training data which made the training data to have a size of 14,100 *tokens*. The model is then trained on this training data and tested on the original test data. AONER now correctly recognized *548 NEs* (TP) missing *156 NEs* (FN). It wrongly recognized *177 NEs* (FPs). TP and FP increased while FN decreased. Table 6.1 shows the obtained evaluation result.

Table 6.1: Influence of increasing the training data

Metrics	**Performance with**	
	Original training data (8000 tokens)	**Increased training data (14,100 tokens)**
Recall	77.41%	77.84%
Precision	75.80%	75.59%
F1-Measure	76.60%	76.70%

The obtained result is not satisfactory when compared to the boring task of developing the experimentation data. The Recall and F1-measure values showed very minimum increment while precision is decreased. The value of Precision is decreased by *0.21%* while Recall is

increased by *0.43%*. F1-measure is improved because of the reason that Recall is increased with a larger value than Precision is decreased. The obtained results reveal that increasing the training data size has a positive impact on Recall but has a negative impact on Precision.

6.6.2 Influence of Features

In order to examine the impact of features on AONER system, we prepared a mechanism of dropping one feature at a time and check the performance of the system with the absence of that feature. AONER will then be trained on the original training data (of size *8000 tokens*) and tested on the test data. If the performance of the model is increased, then the feature is of no use for AONER and it is degrading the system. If the performance is decreased, then the feature is vital for AONER. Given a feature removed, the more decrease in performance the more influential and essential the feature will be for AONER. In the sections that follow, the evaluation of the original performance will be presented along with the evaluations obtained in each section for easy comparison.

6.6.2.1 Influence of Word-Shape feature

The experimentation regarding the influence of features is started by removing word-shape feature from feature lists of AONER. This time, AONER‟s feature extractor was redesigned to extract other features without word-shape feature. The Model Generator generated a different model than the original model. The new model is then tested on the test data. It correctly recognized *535 NEs* (TPs) while it missed *169 NEs* (FNs). It wrongly recognized *192 NEs* (FPs). TPs reduced while FNs increased. FPs radically increased. With the absence of Word-shape feature, the result of evaluation is presented in Table 6.2.

Table 6.2: Influence of Word-Shape feature

Metrics	Performance with	
	All feature sets	**Absence of word-shape feature**
Recall	77.41%	75.99%
Precision	75.80%	73.59%
F1-Measure	76.60%	74.77%

Both Recall and Precision were decreased by *1.42%* and *2.21%* respectively. F1 is also decreased by *1.83%*. From this experimentation we understood that the absence of Word-Shape feature has a negative influence on the performance of AONER system. This on the other hand reveals that Word-Shape feature plays an essential role in the recognition of Afan Oromo NEs. This is true. Because, for example, capitalization is one of the word shape feature used and it is the essential feature for detecting NEs in Afan Oromo.

6.6.2.2 Influence of Suffix feature

To examine the influence of suffix feature on AONER system, we removed the feature from the feature lists. The extracted features were supplied to the model generator and the model is generated. This time it correctly recognized *529 NEs* (TPs) and missed *175 NEs* (FNs). The model recognized an additional of *200 NEs* (FPs) wrongly. When compared with the original output, there was a reduction and large increase values of TPs and FPs respectively. Table 6.3 shows the obtained performance evaluation with the absence of suffix feature.

Table 6.3: Influence of Suffix feature

Metrics	**Performance with**	
	All feature sets	**Absence of suffix feature**
Recall	77.41%	75.14%
Precision	75.80%	72.57%
F1-Measure	76.60%	73.83%

The performance showed that Recall, Precision and F1-measure are each reduced by *2.27%*, *3.23%* and *2.77%* respectively. We understood that suffix is also a decisive feature for recognizing Afan Oromo NEs. We also concluded that though both the word shapes and the suffixes are used for NER in AONER, suffix features are more influential than word shape features.

6.6.2.3 Influence of Prefix feature

Prefixes are one of the features used in the AONER"s feature set. We have conducted an experiment to explore the influence prefixes have in recognizing Afan Oromo NEs. As we did for other features, the influence of prefixes is explored by removing them from AONER"s feature set and letting the feature extractor extract the rest features only. The model generator trained the model with the extracted features and the model is tested on the test data. The obtained *TP value is 542*, *FN value is 162* and *FP value is 184*. The obtained evaluation is shown in Table 6.4.

Table 6.4: Influence of Prefix feature

Metrics	Performance with	
	All feature sets	**Absence of prefix feature**
Recall	77.41%	76.99%
Precision	75.80%	74.66%
F1-Measure	76.60%	75.80%

All the metrics showed a fall in value with *Recall decreased by 0.42%*, *Precision decreased by 1.14%* and *F1-measure decreased by 0.80%*. From the evaluation we have known that using prefix as a feature was the right choice. It helped AONER to recognize Afan Oromo NEs though it is less influential than other features like suffix.

6.6.2.4 Influence of POS feature

After obtaining a promising performance from AOPOST, we used POS of a word as a feature to recognize Afan Oromo NEs. In order to measure the importance of POS of a word in recognizing NEs, we removed the POS from a feature list and trained the model. The model then correctly recognized *542 NEs* failing to recognize *162 NEs* from the test data. It additionally recognized *182 NEs* which are not actually NEs. The obtained evaluation is given in Table 6.5.

Table 6.5: Influence of POS feature

Metrics	Performance with	
	All feature sets	**Absence of POS feature**
Recall	77.41%	77.00%
Precision	75.80%	74.86%
F1-Measure	76.60%	75.91%

With the absence of POS feature, the performance of the model showed a slight decrease with *0.41%* for *Recall*, *0.94%* for *Precision* and *0.69%* for *F1-measure*. So, using POS of a word as feature has helped AONER. POS feature is found to be the least influential feature of AONER when compared to other features being observed.

In addition to the features investigated in the above sections, AONER also used a normalized token feature. This feature is designed for *miscellaneous* category and is not applicable for other NE categories. Observing its influence and evaluating the system with its absence cannot result in a balanced evaluation. As a result, we didn‟t experiment the influence of normalized token feature.

6.6.2.5 Summary

The summary of the performance evaluation of the system with the absence of each experimented features is given in Figure 6.2 for recall, in Figure 6.3 for precision and in Figure 6.4 for F1-measure.

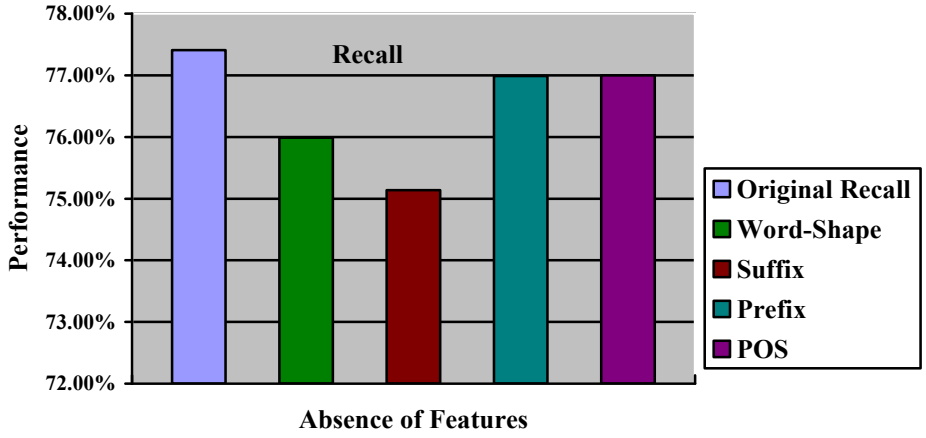

Figure 6.2: Influence of features on Recall value

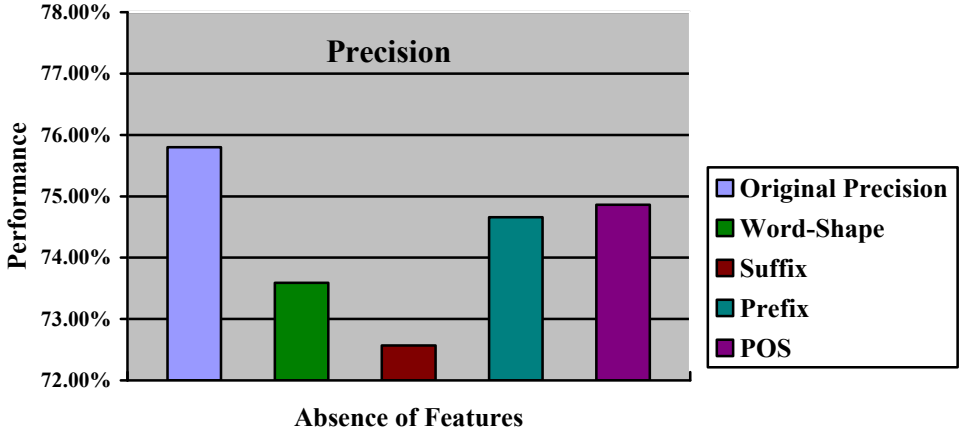

Figure 6.3: Influence of features on Precision value

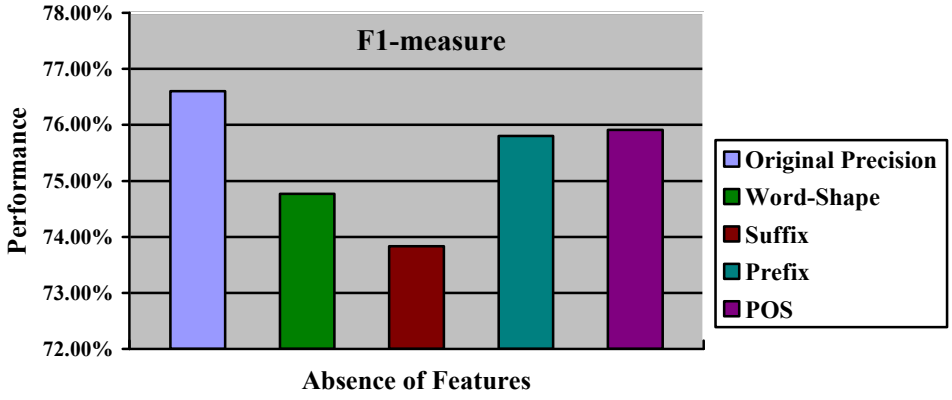

Figure 6.4: Influence of features on F1 value

6.7 Discussion

In this chapter we presented the experimentation aspects associated with both AOPOST and AONER. We used accuracy to measure the performance of AOPOST. Recall, Precision and F1-measure are used as evaluation metrics for AONER. AOPOST obtained an accuracy value of 80.61% by correctly tagging 877 tokens out of 1088 tokens. The confusion matrix of AOPOST is also computed and demonstrated in this chapter.

Being trained on a training data of size 8000 tokens and tested on a test data of size 4000 tokens, AONER performed 77.41% Recall, 75.80% Precision and 76.60% F1-measure. These results were used as a baseline to experiment on AONER. We set up two main experimentation scenarios to experiment on AONER. The first scenario is conducted by increasing the size of the training data. By almost doubling the size of the training data, the values of Recall and F1-measure showed an increase while the value of Precision showed a decrease.

The second scenario was aimed at observing the influence of features. This was conducted by dropping each feature from the feature set of AONER turn by turn. Four features were considered during this scenario: Word-Shape, Suffix, Prefix and POS features. Although the percentage by which the values were decreased differs from feature to feature, with the absence of each feature, the value of all metrics was decreased. This reduced the overall performance of AONER. From this we concluded that each feature has a positive influence and is vital for AONER. Suffix feature is observed to be the most influential feature for AONER with Word-Shape, Prefix and POS comes next respectively. In general, the outcomes verify that we used a good feature combination to develop AONER.

Both scenarios have showed that, keeping the original training data constant, the performance of AONER is highly dependent on a feature than on increasing the training data size. Almost doubling the training data size has shown less contribution than the least influential feature, the POS feature.

Chapter Seven: Conclusion & Future Work

The existence of NLP discipline has enabled computers understand human languages and process them. It also opened a track for the emergence of different tasks like IR and IE. The task of IR is to search and retrieve documents in response to queries for information while IE selectively structures and combines data which is found, explicitly stated or implied, in one or more texts. IE is more robust than IR since it focuses on analysing the facts rather than analysing the documents. Nowadays, researches are shifting to IE as the current world needs a concrete response to its request. Being one of the tasks in IE, NER is concerned with the identification and recognition of NEs from natural language texts. NEs are words like the name of a person, location, organization and the likes. They carry important information about the document or the sentence they are found in. As a result, NER is a first step toward intelligent IE. Researches in NER have far-reached for European and some of Asian languages. However, this is not the case for under resourced languages like that of Ethiopian languages.

In this thesis we developed an NER system, called *AONER*, for Afan Oromo language. Before developing AONER, we have studied some of the NER systems developed for English language. NER approaches have also been studied to select an approach that can give the best performance given the constraints of Afan Oromo language. The nature, structure and pattern of Afan Oromo language has also been studied before developing the system.

7.1 Conclusions

This study proposed and designed a system called AONER to solve Afan Oromo NER problem. The system is designed based on a hybrid approach: machine learning and rule based. The rule based component is not implemented due to time constraints and we left it as a future work. We implemented the machine learning component using CRFs algorithm. For training the system, we developed Afan Oromo NE corpus of more than 23,000 words out of which around 3600 are NEs. We believe this will be the largest Afan Oromo corpus for NLP task ever. The corpus is developed using CoNLL"s 2002 standard. Four NE categories have

been identified and dealt with in this research work. These are person, location, organization and miscellaneous.

The implemented system has different components. The parser checks the correctness and validity of the corpus against CoNLL"s standards. The BIO Encoder component identifies tokens from their tags and generates token/tag sequence. The generated token/tag sequence will be supplied to NE Chunker and Feature Extractor components. NE Chunker combines a sequence of tokens that follow each other and belong to the same NE category to create a NE chunk. Feature Extractor extracts features from the generated tokens/tag sequence. Position, word shape, POS, normalized tokens, prefix and suffix features have been extracted for each token in the training data. The outputs of the feature extractor and NE chunker are used to generate Afan Oromo NE model trained with Afan Oromo texts. The model generation is done by the Model Builder and it is implemented using Stochastic Gradient Descent algorithm.

The input plain text entered by a user is first tokenized by the tokenizer and supplied to the NE Recognizer. The NE Recognizer recognizes the possible NE tokens from the tokenized text with the help of the trained model. The recognition was implemented using forward-backward algorithm.

In parallel with the development of AONER system, we developed a CRF based Afan Oromo POS tagger for the generation of POS features. Afan Oromo POS corpus of size around 5000 tokens was developed to train and test the model. The model obtained accuracy of 80.61% by correctly tagging 877 tokens out of 1088 tokens. The model is integrated to AONER system to supply POS of a token to the feature extractor.

This work is the first of its type for Afan Oromo language. The system"s evaluation showed a promising performance. Being tested on a test data of 4,000 tokens, it obtained 77.41% Recall, 75.80% Precision and 76.60% F1-measure. The system is being evaluated in two scenarios by increasing the training data size and reducing the feature set. By almost doubling the size of the training data, the system"s performance was increased by 0.43% and 0.1% for Recall and F1-measure respectively. The precision is however reduced by 0.21%. Reducing a

feature from the feature set reduced the system‟s performance for all features. In average the value of Recall, Precision and F1-measure decreased by 1.42%, 2% and 1.52% respectively. From this result, we conclude that features play a vital role than the change in the training data size. This, on the other hand, justifies that we used the best feature combination for the development of the system. In general, given different constraints our algorithm obtained good performance compared with resource rich languages like English.

7.2 Contribution of the work

Among the major contributions of this thesis are:

- Proposing a new architecture for Afan Oromo NER system with the state-of-the art approach.
- Identifying the nature and pattern of Afan Oromo NEs and how they can be processed to be understood by a machine.
- Identifying and proposing the procedures, techniques, algorithms and tools used for the development of Afan Oromo NER system.
- Proving that machine learning models can be adapted, trained and can work for Afan Oromo language.
- Confirming that NER systems are dependent on features than on increasing the training data size.
- Enabling the development of Afan Oromo NE corpus with a reasonable size.
- Preparing an encouraging environment for the development of other Afan Oromo NLP studies that need NER as a component in their work.

7.3 Future works

Named entity recognition is a new study for Afan Oromo and also for Ethiopian languages. The task is very complex for such under resourced languages. NER systems are vital components for systems like Question-Answering and Text Summarization. Naturally, NEs carry important information about a document or a sentence. They are helpful in locating the important points. The developed Afan Oromo NER system has portions that require further improvements that we want to recommend them as future works.

- Although we proposed a hybrid approach based architecture for the system, we implemented the machine learning part only. Interested researchers can develop a rule based on the nature and pattern of Afan Oromo NEs and integrate into the system we have developed.
- It is also possible to develop a rule generation mechanism so that the system can learn the structure of Afan Oromo NEs and generate rules by itself. This will enable the system to enhance its performance from time to time without freezing on a constant performance.
- We used CRFs as a machine learning algorithm. One can use other machine learning algorithms like Hidden Markov Model or Maximum Entropy Model using the corpus data we have developed and compare the performance with the one we obtained.
- Our feature extraction algorithm works within one window size. One can increase the window size and the feature set to obtain an improved performance.
- The system developed in this research work is just a prototype. Any interested person can do a project to develop a full-fledged Afan Oromo NER system that can be easily integrated into different applications.

References

Allen, J. 1996. Natural Language Understanding. 2nd Ed. California: Redwood, Benjamin/Cummings Publishing Company, Inc.

Amanuel, Alemayehu. 2006. The Syntax of Interrogatives in Oromo. MA Thesis. Addis Ababa University, Addis Ababa.

Appelt, D. and Israel, J. 1999. Introduction to Information Extraction Technology, A Tutorial Prepared for IJCAI-99, Artificial Intelligence Center, Menlo Park, CA.

Asahara, Masayuki; and Matsumoto, Yuji. 2003. Japanese Named Entity Extraction with Redundant Morphological Analysis. In Proc. of HLTNAACL 2003, pages 8–15.

Borthwick, A., Sterling, J., Agichtein, E., & Grishman, R. 1998. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In Proceedings of the Sixth Workshop of Association for Computational Linguistics ACL on Very Large Corpora New Brunswick, New Jersey.

Borthwick, A.; Sterling, J.; Agichtein, E.; Grishman, R. 1998. NYU: Description of the MENE Named Entity System as used in MUC-7. In Proc. Seventh Message Understanding Conference.

Bose, Ranjit. Natural Language Processing: Current State and Future Directions, International Journal of the Computer, the Internet and Management, Vol. 12#1 (January – April, 2004), pp 1 – 11.

Budi, I. and Bressan, S. 2003. "Association Rules Mining for Name Entity Recognition", Proceedings of the 4th International Conference on Web Information Systems Engineering.

Carpenter, B. and Breck. 2010. LingPipe 3.9.2. Java Documentation and Tutorials.

Catherine, Griefenow-Mewis. 2001. A grammatical sketch of written Oromo. ISBN 3-89645-039-5. Köln : Köppe. Germany.

Cowie, J.; and Lehnert, W. 1996. "Information Extraction", In Communications of the ACM , Vol.39(1), 1996, pp.83-92.

Cunningham, H. 2005. "Information extraction, automatic," Encyclopedia of Language and Linguistics. 2nd ed.

Diriba, Megersa. 2002. An Automatic Sentence Parsing for Oromo Language using Supervised Learning Techniques. MSc Thesis, Addis Ababa University, Addis Ababa.

Ekbal, A.; Haque, R.; Das, A.; Pokan, V. and Bandyopadhyay, S. 2008. "Language Independent Named Entity Recognition in Indian Languages". Proceedings of the IJCNLP-08 Workshop on NER for South and South East Asian Languages, pages 33–40, Hyderabad, India.

Elkan, Charles. 2008. Log linear models and conditional random fields. Notes for a tutorial at CIKM'08. UC Berkeley.

Erik F. Tjong Kim Sang. 2002. Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition. In: *Proceedings of CoNLL-2002*, Taipei, Taiwan, pp. 155-158.

Finkel, Jenny R.; Trond, Grenager; and Manning, Christopher. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370. http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf

Gamta, Tilahun. 1992. "The Oromo language and the latin alphabet". Journal of Oromo Studies. http://www.africa.upenn.edu/Hornet/Afaan_Oromo_19777.html.

GATE Information Extraction, http://gate.ac.uk/ie retrieved on December 3, 2009.

Getachew, Mamo. 2009. Part-Of-Speech Tagging for Afaan Oromo Language using Transformational Error Driven Learning Approach. MSc Thesis. Addis Ababa University, Addis Ababa.

Goldman, S. and Zhou, Y. 2000. Enhancing supervised learning with unlabeled data. In proceedings of the seventeenth International Conference on Machine Learning. PP. 327-334.

Grishman, R; & Sundheim, B. 1996. "Message Understanding Conference -6: A Brief History". In proceedings of the 16th conference on Computational linguistics - Volume 1, PP. 466 – 471, Copenhagen, Denmark.

Isozaki, H. 2001. "Japanese named entity recognition based on a simple rule generator and decision tree learning". In Proceedings of Association for Computational Linguistics, pages 306– 313.

J. Lafferty, A. Mccallum, and F. Pereira. 2001. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in Proc. 18th International Conf. on Machine Learning. Morgan Kaufmann, San Francisco, CA, pp. 282–289.

Jurafsky, D.; and Martin, J. H. Speech and Language Processing, 2nd Ed., Pearson Prentice Hall, New Jersey, 2000.

Klinger, R.; and Tomanek, k.; "Classical probabilistic models and conditional random fields," Technische Universität Dortmund, Dortmund, Electronic Publication, 2007.

Kozareva, Z. 2006. Bootstrapping named entity recognition with automatically generated gazetteer lists. In EACL, The Association for Computer Linguistics.

Kozareva, Zornitsa; Boyan, Bonev; and Montoyo, Andres. 2005. Self-training and Co-training Applied to Spanish Named Entity Recognition. In MICAI 2005: 770-779.

Li, W.; and McCallum A. 2003. Rapid Development of Hindi Named Entity Recognition using Conditional Random Fields and Feature Induction. In: ACM Transactions on Asian Language Information Processing (TALIP), 2(3): 290–294.

Liddy, E.D. 2001. Natural Language Processing. In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc.

Louis, A.; De Waal, A.; and Venter, C. 2006. "Named Entity Recognition in a South African Context". Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, Pages: 170 – 179, Somerset West, South Africa.

Mansouri, A.; Affendey, L. S.; and Mamat, A. 2008. Named Entity Recognition Approaches. IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.2, PP. 339-344.

McDonald, D. 1996. Internal and External Evidence in the Identification and Semantic Categorization of Proper Names. In B. Boguraev and J. Pustejovsky editors: Corpus Processing for Lexical Acquisition. Pages 21-39. MIT Press. Cambridge, MA.

Mikheev, Andrei, Claire Grover, and Marc Moens. 1999a. XML tools and architecture for named entity recognition. Markup Languages: Theory and Practice, 1(1).

Mohammed, Hussen. 2010. Part Of Speech Tagger for Afaan Oromo Language using Transformational Error Driven Learning (Tel) Approach. MSc Thesis. Addis Ababa University, Addis Ababa.

Nadeau, D.; and Sekine, S. 2007. A Survey of Named Entity Recognition and Classification. In: Sekine, S. and Ranchhod, E. Named Entities: Recognition, classification and use. Special issue of Lingvisticæ Investigationes. 30(1) pp. 3-26.

Nadeau, David. 2007. Semi-Supervised Named Entity Recognition. Ph.D. thesis, Ottawa-Carleton Institute for Computer Science, University of Ottawa, Ottawa, Canada.

Nafa, Abarra. 1988. Long Vowel in Oromo. A Generative Approach. MA Thesis. Addis Ababa University, Addis Ababa.

Razvan, C. Bunescu. 2007. "Learning for Information Extraction: From Named Entity Recognition and Disambiguation to Relation Extraction", Ph.D thesis, The University of Texas at Austin, Texas.

Sang, Erik F. Tjong Kim; and Meulder, Fien De. 2003. "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition". In Walter Daelemans and Miles Osborne, editors, Proceedings of CoNLL-2003, pages 142–147. Edmonton, Canada.

Sassano, M. and Utsuro, T. 2000. "Named Entity Chunking Techniques in Supervised Learning for Japanese Named Entity Recognition", In Proceedings of COLING 2000.

Sekine, S. and Eriguchi, Y. 2000. "Japanese named entity extraction evaluation - analysis of results". In Proceedings of 18th International Conference on Computational Linguistics, pages 1106–1110.

Sha, F., & Pereira, F. 2003. Shallow parsing with conditional random fields. *HLT-NAACL-2003*

Solorio, Thamar. 2004. Improvement of Named Entity Tagging by Machine Learning. Reporte Técnico No. CCC-04-004, Coordinación de Ciencias Computacionales, Instituto Nacional de Astrofísica, Óptica y Electrónica.

Spiegel, M. 1985. "Pronouncing surnames automatically". Proceedings of the American Voice Input/Ouput Society.

Srikanth, P and Kavi N. Murthy. 2008. Named Entity Recognition for Telugu. In Proceedings of the IJCNLP-08 Workshop on NER for South and South East Asian Languages, pages 41–50, Hyderabad, India.

Sutton, Charles; and McCallum, Andrew. 2006. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, Introduction to Statistical Relational Learning. MIT Press.

Wu, Y.C.; Fan, T.K.; Lee, Y.S.; and Yen, S.J. 2006. “Extracting Named Entities Using Support Vector Machines", Spring-Verlag, Berlin Heidelberg.

Zhang, Tong; & Johnson, David. 2003. “A Robust Risk Minimization based NER System”. In proceedings of CONNL, PP. 204-207, New York, USA.

Zhou, GuoDong; & Su, Jian. 2002. “Named Entity Recognition using HMM-based Chunk Tagger”. Proceedings of the 40th Annual Meeting of the ACL, Philadelphia.

APPENDICES

Appendix A: Feature spaces for AONER

Feature	Person	Organization	Location	Date/Time	Percentage	Monetary
Case	– Starts uppercase	– Starts Uppercase – All Uppercase – Mixed Case	– Starts Uppercase			
Punctuation	– Ends with period	– Ends with period – Contains period inside – Contains back slash (/) inside				– Contains period – Contains comma
Digit		– Word with digit		– Two digit – Four digit		
Morphology	-n, -tiin, -iin -dhaan, -f, -tiif -iif, -ti, -s	-n, -ni, -s, -tiin, -tiif	-tti, -s, -rraa, -ttis	-rratti, -tti	-tti	-tiin
List of Entity Clues before the entity name	Obbo, Adde, Aadde, durbee, dubree, weellisaa, Pirezedaanti, dargaggoo, Piroofeesar, barataa, barattuu, Atsee Ispeektar, Saajin, Dr., Doktar, Ambaasaaddar,	Dhaabbata/i, mootummaa, Korporeeshinii, Ejansii, Komishinii/a, I.M.X, Baankii, Waldaa, Yuuniyenii, Waajjira, Hoteela, Hospitaala, inistiitiyuutii, paartii,	Magaalaa, aanaa, godina, naannoo, ganda, addaa, ardii, biyya, iddoolee, eddoo, atileetonni, atileetotni, taphatootni, taphatonni, raayyaa, waltajjii, istaadiyoomii, tabba, Ambaasaadarri, uummanni/uummatni, qorattootni/qorattoonni,	Bara, gaafa, saʻa, saʻatii, ji'a, waggaa, kurmaana, daqiiqaa, sakondii, sekondii, beellama, ALA, ALH	Dhibbeentaa, dhibbantaa, %	Qarshii, qar., saantima, birrii, paawundii, yuuroo, yeenii, doolaara, naaqfaa, shilingii, dirhaamii,
	Kaapteen,	paartiin, garee/n,	lammii			

	Komaandar, atleeti, atleet, Mr., Mis, Mistar, Koloneel, Jeneraal, Haaji, Luba, Konistaabil, B/B, Barsiisaa, B/saa, waajjirichaa, R/himataa, R/himataan, R/himatamaan, R/himatamaa, M/A/Mirgaa, M/A/Idaa, M/A/M, M/A/I, Sh/, Shaalaqaa, A/, S/r, Siistar, Oliyyataa, D/Kennaa, Sheek, Qees, H/, Lij	kilabii/n, kilaba, Yunivarsitii/n, Faawundeeshinii, M/M/G, B/M, B/N/M, M/M/A, M/M/Ol/G, M/Q/M, M/M/Ol/Go/A, M/M/Q/M, W/A/Q/D/F, Biiroo/n, Kubbaaniyaa/n, Kolleejjii/n, warshaa, giruuppii/n, Industirii, Boordii, Embaasii, Hoogganaa				
List of Entity Clues after the entity name	eeraniiru, dubbataniiru, dhaamaniiru, himaniiru, ibsaniiru, beeksisaniiru, mirkaneessaniiru, dhiheessaniiru, addeessaniiru, dabalanii	gabaaseera, beeksiseera, giruup, P.L.C, W.A	keessatti	Keessa, ALA, ALH, pm, am		

Appendix B: Some of Afan Oromo Numerals

Afan Oromo	English		Afan Oromo	English
Tokko	One		Afurtama	Forty
Lama	Two		Shantama	Fifty
Sadii	Three		Jaatama	Sixty
Afur	Four		Torbaatama	Seventy
Shan	Five		Saddeetama	Eighty
Jaʻʻa (Jaha)	Six		Sagaltama	Ninety
Torba	Seven		Dhibba	Hundred
Saddeet	Eight		Kuma	Thousand
Sagal	Nine		Kuma kudhan	Ten thousand
Kudhan	Ten		Kuma dhibba	Hundred thousand
Digdama	Twenty		Miiliyoona	Million
Soddoma	Thirty		Biiliyoona	Billion

Appendix C: Name of days and months in Afan Oromo

Days:

Afan Oromo	English
Dafinoo / Wiixata	Monday
Facaasaa / Kibxata	Tuesday
Robii / Arbii	Wednesday
Kamisa	Thursday
Jimaata	Friday
Sambata	Saturday
Dilbata	Sunday

Months:

Afan Oromo	English
Fulbaana	September
Onkololeessa	October
Sadaasa	November
Mudde	December
Amajji	January
Gurraandhala	February
Bitootessa	March
Ebla	April
Caamsaa	May
Waxabajji	June
Adoolessa	July
Hagayya	August

Appendix D: Word Shape feature values

The following Table lists a word shape feature value when extracted for a token. The value under the column „*category*" will be appended with a word shape feature of a token based on its character shape.

Category	Description
NULL-TOK	Zero-length string.
1-DIG	A single digit.
2-DIG	A two-digit string.
3-DIG	A three digit string.
4-DIG	A four digit string.
5+-DIG	String of all digits five or more digits long.
DIG-LET	Contains digits and letters.
DIG--	Contains digits and hyphens.
DIG-/	Contains digits and slashes.
DIG-,	Contains digits and commas.
DIG-.	Contains digits and periods.
1-LET-UP	A single uppercase letter.
1-LET-LOW	One lowercase letter.
LET-UP	Uppercase letters only.
LET-LOW	Lowercase letters only.
LET-CAP	Uppercase letter followed by one or more lowercase letters.
LET-MIX	Letters only, containing both uppercase and lowercase.
PUNC-	A sequence of punctuation characters.
OTHER	Anything else.

Declaration

This thesis is my original work and has not been submitted as a partial fulfillment for a degree in any university and that all sources of the materials used for the thesis have been duly acknowledged.

_____

MANDEFRO LEGESSE KEJELA

OCTOBER 2010

The thesis has been submitted for examination with my approval as university advisor.

_____

DEJENE EJIGU (Ph.D)