Flexible Backing-off Strategies for HMM based Named Entity Recognition

A thesis submitted in partial fulfillment of the requirements for the degree of

Master Thesis in Computer Science

Computer Science Department
University of Saarland

Yasir Iqbal
(Matriculation # 2029833)

Supervisors: Prof. Dr. Gerhard Weikum
Max-Planck-Institut für Informatik (MPI)
Databases and Information Systems Group
Saarbrücken, Germany

Prof. Dr. Dietrich Klakow
Lehrstuhl für Sprachsignalverarbeitung (LSV)
Universität des Saarlandes
Saarbrücken, Germany

Begin: September 28, 2005
End: March 28, 2006
Eidesstattliche Erklärung

Hiermit erklärte ich an Eides statt, dass ich die vorliegende Mastersarbeit selbständig und ohne fremde Hilfe verfasst habe. Ich habe dazu keine weiteren als die angeführten Hilfsmittel benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet.


Yasir Iqbal
Abstract

Named Entities are a major target in most of queries for modern information retrieval systems like search engines and question answering systems. Usually these systems index large number of document collections in different languages. Where, of course, manual annotation is out of question even rule based annotation becomes tedious task because of complex grammar of natural languages and the required number of rules which are unmanageable. A language independent, stochastic and flexible framework could fill the vacuum. In this regard, several research groups around the globe have already done a lot of work. This thesis explores this active research field of statistical language modeling and extends the toolkit designed and developed by the Spoken Language Systems group at Saarland University. On top of the toolkit, an HMM-based, natural-language independent, named entity tagger is introduced with a heuristic stack decoder developed as part of this thesis. Although our target-named-entities, throughout this work, are MUC-7 named entities [7] but the tool can be configured for custom entities. We focus on giving flexibility in defining the language models, parameterization and in backing off schemes. Further, it has simplified the configuration problem in N-gram language modeling for complex languages where specifying $N$ previous tokens was not enough. We have applied the technique to rather smaller corpora for several European languages with minimal training. Current experiments show promising results, a little more effort in smoothing parameters and language modeling tricks results can compete with the industry standard.
Acknowledgments

First, I must thank to so many people at the University of Saarland who gave me a chance to build a successful career, particularly Susanne Schneider, Kerstin Meyer-Ross and Erich Reindel.

I am appreciative to Prof. Dr. Dietrich Klakow (Chair of Speech Signal Processing, LSV) and Prof. Dr. Gerhard Weikum (Databases and Information Systems Group at Max-Planck Institute of Informatics) for accepting me as a master student at their chairs and providing me the opportunity to carry out this research work at the LSV of the University of Saarland, Saarbrücken, Germany.

I am highly obliged to the whole LSV Group especially Prof. Klakow (the head) for providing me not only a family environment in his group but also the input from the computational linguistic domain with protracted flourishing discussions in order to finish this research work. Additionally, I am also thankful to Jochen Leidner, Stephan Schacht, Irene Cramer and Dipl. Inf. Andreas Merkel, because without their splendid support, I might have not been able to finish this thesis. They all helped me a lot in order to understand not only the problem domain but in resolving technical issues as well. I am thankful to Ghazala Iqbal, Lina Cepurnaite, and Ghulam Mujtaba for their immense support required for the experimental data for the research work and thankful to Mr. Dietmar Kuhn and Mrs. Diana Schreyer at LSV for providing me the environment to learn and speak German language.

Finally, I am deeply indebted to my family for their continuous support and patience during my extended study-period. In spite of the distance, they were an incessant source of encouragement. My gratitude for my wife Ghazala and my daughter Maryam is beyond the words. I owe my friends; especially Naveed Ahmed, Jumamurat Bayjanov, Sahbi Smaoui, Imran Rauf, Mohammed Seghir, Safdar Ali and so many others to list who kept me supporting morally and financially. It would be fair to say that there is more contribution of other actors than of myself.
Dedicated to

My kind & loving mother and patient wife and daughter
## Table of Contents

1 Chapter - Introduction .............................................................................................................. 11
  1.1 Motivation ......................................................................................................................... 11
  1.2 Goal .................................................................................................................................... 12
  1.3 Contributions ...................................................................................................................... 12
  1.4 Outline ............................................................................................................................... 12

2 Chapter - Background & Concepts .......................................................................................... 14
  2.1 Named Entities .................................................................................................................... 14
    2.1.1 Applications .................................................................................................................. 14
  2.2 Probability review ............................................................................................................... 15
  2.3 Bayes’ theorem .................................................................................................................. 16
  2.4 Zero Probability Problem ................................................................................................... 17
  2.5 Classification ....................................................................................................................... 17
  2.6 Named Entities (NE) Recognition ....................................................................................... 18
  2.7 Corpus ................................................................................................................................ 18
    2.7.1 Corpus Index .................................................................................................................. 18
    2.7.2 Training .......................................................................................................................... 18
    2.7.3 Word to class mapping .................................................................................................. 19
    2.7.4 Count Tree .................................................................................................................... 19
    2.7.5 Vocabulary .................................................................................................................... 21
  2.8 Statistical Language Modeling ............................................................................................. 21
    2.8.1 N-gram model and Variants .......................................................................................... 22
  2.9 Hidden Markov Models ....................................................................................................... 22
    2.9.1 Markov Models .............................................................................................................. 22
    2.9.2 Hidden Markov Models ................................................................................................ 22
  2.10 Smoothing .......................................................................................................................... 23
    2.10.1 Language model Smoothing ........................................................................................ 23
    2.10.2 Backing off in Language modeling .............................................................................. 24
  2.11 Related Work ...................................................................................................................... 25
    2.11.1 BBN Nymble .............................................................................................................. 25

3 Chapter - Problem Description & Our Approach ..................................................................... 28
  3.1 LM based Named Entities (NE) Recognition ........................................................................ 28
    3.1.1 What NE is NOT ............................................................................................................ 28
  3.2 Available approaches ............................................................................................................ 28
    3.2.1 Rules based NE .............................................................................................................. 28
    3.2.2 Statistical NE ................................................................................................................. 29
  3.3 LSVLM approach .................................................................................................................. 30
    3.3.1 Learning ........................................................................................................................ 30
    3.3.2 Unknown words ............................................................................................................. 30
    3.3.3 Probability calculation ................................................................................................... 31
    3.3.4 Smoothing and Backing-Off Schemes ......................................................................... 31
    3.3.5 Decoding ........................................................................................................................ 31
    3.3.6 Evaluation and Conclusions ......................................................................................... 31

4 Chapter - Implementation ....................................................................................................... 33
  4.1 System architecture .............................................................................................................. 33
    4.1.1 Description .................................................................................................................... 34
Chapter - Introduction

1.1 Motivation

Named entities are the most important targets in daily web search. From the telephone directory to intelligent question answering systems, the names entities are required to be distinguished from simple keywords. Proprietary invention at BBN [1], called Nymble, is a noble name-learning tool to recognize NEs out of the given textual input, which lead us, think toward the goal. In addition, stochastic approach for NE recognition has been longed at the Saarland University by several groups including Spoken Language Systems (LSV), Language Technology (LT, DFKI). Language modeling toolkit (LSVLM) designed and developed at LSV invited my interest therefore I found empty room in backing-off and LM smoothing.

Language models (LMs) have so many applications such as speech recognition, spelling correction, machine translation and information retrieval. Speech recognition in particular is heavily dependent [2] on LMs and if someone were to divide a speech recognition system into two sub systems, one would be the acoustic model and the other, a LM. A statistical LM is a probability distribution over all word sequences in the language of interest. Detailed description of a speech recognition system itself is out of the scope of the document but in general, the recognizer combines the acoustic similarity of the observed data to a candidate token and the LM score (probability) of the candidate in order to select the candidate most likely to correspond to the spoken sound. Currently, the most widely used LMs are based on word N-gram counts – simple counts of contiguous occurrences of a word-pair (bi-grams) or word-triple (tri-grams) in a corpus. The main advantage of these models is the ease of model parameter estimation, while still being relatively accurate and powerful. However, they suffer from several drawbacks, specially the data sparseness problem and the lack of syntactic “well formed-ness”. In recent research history, many smoothing techniques have been suggested to handle these problems and additionally improve LM performance.
BBN-Nybmle name-finder leads us toward the NE research which uses fewer and fixed number of backing-off schemes which, at one stage, seem insufficient to cope with first problem described above.

1.2 Goal

Main goal of the thesis work was to provide simplification and sophistication to the backing off methods during the process of score assignment phase of the LM based NE recognizer. A top-down idea behind these schemes is to fallback with a simpler scheme, which has lower complexity of history configuration. N-gram configuration is the number of previous tokens to consider as history during probability (LM score) calculation. This problem becomes frustrating when we wish to skip few of the previous tokens (words or classes) and take into account the tokens before previous ones.

During the thesis work, we have tried to develop a method to specify a list of fallback options. The scorer method starts from the longest history configuration, if the longer history configuration fails to give us a non-zero probability then it takes a simpler configuration and it goes on and on until the simplest language model is left behind which is usually normal distribution over all the number of available word-classes for the word-token.

1.3 Contributions

As output of the thesis work there is full working named entity recognizer, which supports proprietary format of specifying the backing-off schemes in flexible manner. This tool performs well with most of the Unicode character set based languages. Due to unavailability of the data used in evaluation of the other tools, comparison of the results is not possible.

1.4 Outline

Chapter 2 of the document provides some background material, which is helpful to understand this work. Moreover, we review significant related work concerning NE recognition and smoothing methods. Chapter 3 further elucidates the problem statement
and the motivation behind this work along with the targets that we attempt to achieve. Then fourth chapter gives overview on the implementation, next is fifth chapter containing experiments and results. Finally, Chapter 6 concludes the work with potential future work in the direction of this thesis.
2 Chapter - Background & Concepts

This chapter gives introductions to the concepts and basics that might give the required background to understand the contents of the next chapters. The chapter provides general overview and background on important related topics mainly probability, classification, language modeling and its applications and then general concepts.

2.1 Named Entities

In natural language text, any conceptual keyword can be said an entity like boy, river, city or so but names used to point these concepts are said named-entities. For example, Bill Gates, Amazon River and Berlin are instances of these concepts. For this thesis work, standard MUC-7 NEs [7] are taken into account namely: names of organizations, person names, names of places, time expressions, date expressions, monetary values and percentage. The task can be modeled as a classification problem where the text tokens have to be classified into one of the above classes or OTHER as a special state class. As input we are given text S and a set of classes N that formally can be written as [3]:

Word sequence $S = W_1, W_2 ... W_n$

Named classes $N = N_1, N_2 ... N_n$

$P(NS) = P(W_1 W_2 ... W_n N_1 N_2 ... N_n)$

$= P(W_n | W_1 ... W_{n-1} N_1 ... N_n) \cdot P(W_1 ... W_{n-1} N_1 ... N_n)$

$= P(W_n | W_1 ... W_{n-1} N_1 ... N_n) \cdot P(N_n | W_1 ... W_{n-1} N_1 ... N_{n-1}) \cdot P(W_1 ... W_{n-1} N_1 ... N_{n-1})$

Finally:

$P(NS) = \prod_{i=1}^{n} P(W_i | W_1 ... W_{i-1} N_1 ... N_i) \cdot P(N_i | W_1 ... W_{i-1} N_1 ... N_{i-1})$ \hspace{1cm} EQ-2.1-

2.1.1 Applications

Recent evolution in information search engines has proved that named entities are the major targets in user queries. Apart from the named entities recognitions, statistical
techniques can be applied to different data domains. Major applications of named entities are:

- Information Extraction (e.g. answer generation in QA system)
- Summary generation (summarize the main entities and relations)
- Machine Translation (understanding of grammar or skip words which are names)
- Document classification
- Automatic indexing of books
- Increase accuracy of Internet search results (location Bath/The UK vs. verb bath)

### 2.2 Probability review

Probability is a way to summarize the amount of certainty or uncertainty of the truthfulness of some statements or occurrence of the events [4]. For the simplicity, usually numerical representation is used to measure the degree of the certainty or uncertainty.

In other words, we could define; the probability is the ratio of the number of favorable outcome of the total number of all possible outcomes. These all-possible outcomes are also called sample space.

If there are total \( n \) possible outcomes in a sample space \( S \), and \( m \) of those are favorable for an event \( A \), then probability of event \( A \) \( [P(A)] \) is given as:

\[
P(A) = \frac{\text{(# of favorable outcomes } m\text{)}}{\text{(total # of possible outcomes } n\text{)}}
\]

\[
= \frac{n(A)}{n(S)} \\
= \frac{m}{n}
\]

Axioms of probability: Probability values must be between 0 and 1, i.e. \( 0 \leq p \leq 1 \). If an event is not likely to occur or statement cannot be true at all then the probability value will be zero, and when the event is highly likely to occur then the value is one.

Example: - \( P(A \text{ pregnant mammal being a female}) = 1 \)

Example: - \( P(A \text{ mammal male being pregnant}) = 0 \).

Probability theory is based on following axioms:
\[0 \leq P(E) \leq 1\]

\[P(E=\text{True}) = 1; \ P(E=\text{False}) = 0\]

\[P(EA \lor EB) = P(EA) + P(EB) - P(EA \land EB)\]

Where, \(\lor\) is the logical OR and \(\land\) is the logical AND operator.

For the brevity only important features of the probability theory is mentioned here.

Two events \(EA\) and \(EB\) is said to be independent if: \(P(EA \land EB) = P(EA) \times P(EB)\)

Conditional probability is used to learn the probability of an \(EB\), which occurs after an event \(EA\), and the result of the event \(EA\) is already known. It is computed by the following formula: \(P(EB \mid EA) = P(EB \land EA) / P(EA)\)

### 2.3 Bayes’ theorem

Simple extension to the conditional probability is known as Bayes Theorem. The theorem is used in many aspects in natural language processing. Obvious example would be Bayesian Rules. Bayes’ theorem can be derived from the conditional probability formulas:

\[P(EB \mid EA) = P(EB \land EA) / P(EA) \quad (1)\]

\[P(EA \mid EB) = P(EB \land EA) / P(EB) \quad (2)\]

From (1) and (2)

\[P(EB \land EA) = P(EB \mid EA) \times P(EA)\]

\[P(EB \land EA) = P(EA \mid EB) \times P(EB)\]

Since left hand sides of two equations are equal and equating right hand sides of these two equations would result in:

\[P(EB \mid EA) = P(EA \mid EB) \times P(EB) / P(EA)\]
2.4 Zero Probability Problem

We assign a zero probability to the event which can never occur in previous example a mammal seen a male is always false, hence with a zero probability. In most of the classification problems, we face the “zero probability problem” where some of the events are assigned zero probability that could be logically wrong. For example a logical statement saying “a horse is flying” can be given a zero probability on the other hand a statement like “dog was climbing a tree” could have some truth, therefore we cannot assign it zero probability.

In the context of language modeling, if the underlying mechanism predicts that a particular event could never happen is logically wrong. In general, we can say that even if a word sequence is very rare to appear nevertheless we are never sure that it will never appear, so the system should never assign a zero probability. Usually this problem appears when the system has never observed an event before like this, and the conditional probability became zero and overall result is zero.

To eliminate such a problem some salt-like parameterization and balancing techniques called smoothing are invented. Commonly these techniques supplement a little amount of extra weights from too common events and assign some weight-age to the rare events.

2.5 Classification

A task to categorize the things into groups also called as classes. Common examples of classification are as below:

- Information Extraction (e.g. answer generation in QA system)
- Speech recognition systems classify phonemes (tones or sound classes) into words
- Classifying the email messages into “legitimate” or “spam”
- Classifying the documents according to their type (business, legal… etc), language (for example German, English, Chinese …) or contents (i.e. payroll, bill, news, columns, feature)
- Classifying passengers on airports passport control as “NATIONAL”, “EU” or “NON-EU” (dividing into three classes)
2.6 Named Entities (NE) Recognition
NEs are a set of proper nouns and fact-holding expressions, which make up a document. For example, a news article is most probably about some events and entities, which are dependent to each other. In this thesis, we consider our named entities set as of MUC-7 NEs. Which consists of names of organizations (ORGANIZATION), person names (PERSON), place names (LOCATION), time expression (TIME), date expression (DATE), money (MONEY) and percent expression (PERCENT). If a token does not belong to all of these expressions then it is called “not an entity” (OTHER).

Formally, we can say it a set of classes (class vocabulary).

Set of classes $C = \{\text{TIME}, \text{PERSON}, \text{PERCENT}, \text{MONEY}, \text{DATE}, \text{ORGANIZATION}, \text{LOCATION}, \text{OTHER}\}$

NE recognition is also a classification problem where each word has to be classified to a class out of above set.

2.7 Corpus
In general, corpus is collection of documents or text [4]. If this collection is annotated with some predefined tags then this collection becomes a training data for some a machine learning based AI tool. In case we have to annotate this collection then this collection is test data for our tool. For this thesis, MUC-7 data has been used for the training and test for English language. For other languages (mainly German and Lithuanian), a manually crafted test and annotated corpus has been used.

2.7.1 Corpus Index
Text contained in the corpus is converted into machine-readable index. To do this each unique token in the corpus is assigned a unique integer representation (index key).

2.7.2 Training
Training data is a collection of documents (human annotated) that are used to train a model. A small training data might cause more errors because of lower frequencies of observations of the events. Below is a real training-sentence example from MUC-7 training (English).

```
<ENAMEX TYPE="LOCATION">WASHINGTON</ENAMEX> The <ENAMEX TYPE="ORGANIZATION">Pentagon</ENAMEX> has denied a request that top <ENAMEX TYPE="LOCATION">U.S.</ENAMEX> commanders in <ENAMEX TYPE="LOCATION">Hawaii</ENAMEX> in <TIMEX TYPE="DATE">1941</TIMEX> be absolved of blame for failing to be on alert for the Japanese attack on <ENAMEX TYPE="LOCATION">Pearl Harbor</ENAMEX>
```

### 2.7.3 Word to class mapping

From the training data we have the following information:

A word token and a class tag (or respective class id) for this word. For simplicity, we map our corpus in the following way:

\[
W_0 \longrightarrow C_0 \\
W_1 \longrightarrow C_1 \\
W_2 \longrightarrow C_2 \\
\vdots \\
W_n \longrightarrow C_n
\]

#### 2.7.4 Count Tree

The count tree is built, using a tool part of the thesis, from the training corpus and given vocabularies. A count tree contains the vocabularies, sequence of vocabularies, depth and levels of the tree. Each level in the tree contains heterogeneous node types.
2.7.4.1 Word node
This node type has words encapsulated in it, having its index, number of children, and pointer to the first child.

2.7.4.2 Class node
Node representing a class of token is class; a unique collection of these nodes produces class vocabulary. In the following illustration a word node sequence is shown with possibility of having different class node associations (adopted from Nymble), Initial and ending state are START-OF-SENTENCE and END-OF-SENTENCE special states.
2.7.5 Vocabulary

Collection of unique tokens from the given corpus is called vocabulary. In this system, we will be talking about two types of vocabularies representing their respective token types. Initially vocabularies are extracted from the training data.

2.7.5.1 Word vocabulary

The word vocabulary consists of unique word token out of the given text collection. Token separated by a space is a new word for the system.

2.7.5.2 Class vocabulary

Collection of unique class tags, each given to the word token in the corpus (i.e. training data).

2.8 Statistical Language Modeling

In general, language model LM is mapping of domain entities into a model as a language. Like natural language, it can have vocabulary and grammar. The goal of Statistical Language Modeling is to build a statistical language model that can estimate the distribution of natural language as accurate as possible [5]. A statistical language model
(SLM) is a probability distribution \( P(S) \) over strings \( S \) that attempts to reflect how frequently a string \( S \) occurs as a sentence.

The original (and is still the most important) application of SLMs is speech recognition, but SLMs also play a vital role in various other natural language applications as diverse as machine translation, part-of-speech tagging, intelligent input method and Text To Speech system.

2.8.1 N-gram model and Variants

N-gram model is the most widely used SLM today [5]. Without loss of generality, we can express the probability of a string \( S \): \( P(S) \) as

\[
P(S) = P(W_1).P(W_2|W_1).P(W_3|W_1W_2)...P(W_i|W_1...W_{i-1}) = \prod_{i=1}^{n} P(W_i|W_{i-n+1}...W_{i-1})
\]

2.9 Hidden Markov Models

2.9.1 Markov Models

A Markov model is a probabilistic process [6] over a finite set, \( \{S_1, ..., S_n\} \), usually called its states. Each state-transition generates a character from the alphabet of the process. We are interested in matters such as the probability of a given state, \( P(S_i) \), and this may depend on the prior history.

2.9.2 Hidden Markov Models

A Hidden Markov Model (HMM) is simply a Markov Model in which the states are hidden. If we use HMM then a sentence is represented by a particular sequence of models \( M \) and the probability would be \( P(M|X) \) where:

\[
P(M|X) = \frac{P(X|M)P(M)}{P(X)}
\]

**Graphical HMM:** Three state typical hidden Markov model
2.10 Smoothing

In natural language, it is usual that the frequencies of some of the words (even in any unbiased corpus) are more than the others. For instance, in English language we can see that the frequency of words like “the”, “of”, “and”, “to” is very high but words like “adept” or “adjourn” are very rare. In N-gram language modeling, probability to an N-gram sequence is assigned based on its relative frequency, which is later normalized.

If the training corpus is not large enough, may actually possible word successions may not be well observed. The core issue of smoothing is to assign a non-zero probability to any given token or sequence.

2.10.1 Language model Smoothing

Language models are important factor in information retrieval [3]. More frequently used language model is a statistical language model, where a word (sequence of words) is assigned a weight (or a vector of weights) depending on its distribution throughout a training data. The statistical language model that is built upon distribution of every word is not robust enough. Therefore, most language models are based on distribution of contiguous occurrences of two words (bi-gram) or three words (trigram) in a training data. Building such statistical language model is relatively easier, but it has pitfalls as well. When such model is trained on a small training data model will not work well in practice. There could be no occurrence of a word (or bi-gram, tri-gram, n-gram…) in the training data, though it appears in an input text. Problems arisen from data sparseness are solved by smoothing. The statistical language model that is based on n-gram counting is just a Markov chain of an order n and it is defined as:
\[ P(S) = \prod_{n=1}^{n} P(W_i | W_{i-n+1} \ldots W_{i-1}) \]

where \( S = W_1, W_2 \ldots W_n \) is a string of words, \( n=2 \) is bi-gram and \( n=3 \) is a tri-gram, and so on.

In the tri-gram case:

\[ P(W_i | W_{i-2}, W_{i-1}) = \frac{N(W_{i-2}, W_{i-1}, W_i)}{N(W_{i-2}, W_{i-1})} \]

Where \( N(a, b) \) is the number of contiguous occurrences of words \( a \) & \( b \) in a training data. When such model is used it would assign zero probability if there is no occurrence of an input bi-gram in a training data. Hence this would make \( P(W) = 0 \), which restricts usage of such model. Backing-off, which is one of the smoothing techniques, is widely used to get rid of such limitations. Assume words \( a, b \) and \( c \) are came together in an input text, but there is no contiguous occurrence of these words in the training data. The system with backing-off smoothing filter backs-off to more general model, that is, to a bi-gram model. Probabilities are computed in a bi-gram model and small probability mass is subtracted from the probabilities of seen words and this small probability mass is distributed over unseen words. Therefore, the system with backing-off smoothing would assign \( P(W) > 0 \), unless there is no occurrence of even a single word from word sequence \( W \). Backing-off smoothing makes a language model more robust and applicable to noisy texts as well.

### 2.10.2 Backing off in Language modeling

Conditional probability may not be able to yield a non-zero value during the calculation because of unavailability of the particular (passed in history or condition) events it has observed during training. Backing off schemes are to fall back to a model with shorter history. As shown in the following diagram, we take the top model with the longest history first and on failure we consider with the shorter previous words or classes.
2.11 Related Work

In this section, we describe significant relevant work done in area of NE and language models smoothing.

2.11.1 BBN Nymble

This system was the most significant motivation to our NE tagger work, most of our system, in theory, resembles to BBN Nymble. This statistical, a variant of standard HMM variant, fast name learner can learn from the hand crafted training and finds non-recursive NEs out of the text.

2.11.1.1 Model

In-formally speaking, the whole model is a combination of conditional probabilities and handcrafted classes, according to the features, of the word tokens. They extract the properties like the token being a numeric, capitalized or stressed by punctuation. The following features were extracted [1]:

\[
\begin{align*}
\Pr(NC | NC_{-1}, w_{-1}, w_0) \\
\Pr(NC | NC_{-1}, w_{-1}) \\
\Pr(NC | NC_{-1}) \\
\Pr(NC) \\
1 \\
\text{#name-classes}
\end{align*}
\]
2.11.1.2 Probability calculation

They used several techniques to enhance performance over a basic HMM. Most notably, they used bi-gram probabilities: they differentiated between the probability of generating the first word of a name and subsequent words of a name. The probability of generating the first word was made dependent on the prior state; the probability of generating subsequent words was made dependent on the prior word. The probability of a state transition was made dependent on the prior word. This had to be combined with smoothing to handle the case of unseen bi-grams.

Named class bi-gram (1):

\[
\Pr(NC \mid NC_{-1}, w_{-1}) = \frac{c(NC, NC_{-1}, w_{-1})}{c(NC_{-1}, w_{-1})}
\]

First word bi-gram (2)

\[
\Pr(\langle w, f \rangle_{\text{first}} \mid NC, NC_{-1}) = \frac{c(\langle w, f \rangle_{\text{first}}, NC, NC_{-1})}{c(NC, NC_{-1})}
\]

Non-first word bi-gram (3)

\[
\Pr(\langle w, f \rangle \mid \langle w, f \rangle_{-1}, NC) = \frac{c(\langle w, f \rangle, \langle w, f \rangle_{-1}, NC)}{c(\langle w, f \rangle_{-1}, NC)}
\]

Here \( c \text{ (event) } \) means the number of occurrences of the event in the training corpus.
2.11.1.3 Backing Off schemes

In Nymble, it seems it has fixed number of backing off schemes and in case all of above bi-gram models fail, it uses the normal distribution. It falls back from top to down with respect to model strength. We employ the same scheme, but our backing off schemes is not fixed with bi-grams but can be extended easily with more steps during fall to a weaker model.

2.11.1.4 Decoding

A Viterbi decoder was used in Nymble, the model generates state lattice with weights on states. The decoder search determines the most probable state sequence out of search space (generated lattice). Using dynamic programming approach they achieve efficiency of $O(m)$ where m is the number of tokens in a sentence.
3 Chapter - Problem Description & Our Approach

LM based Named Entities Recognitions and Backing-Off in LSVLM

3.1 LM based Named Entities (NE) Recognition

Exponential growth of data generated in today’s computing pushed recent boom in research of information retrieval and extraction. Due to importance of named entities in any corpus, it is necessary that they must be annotated and extracted.

NE involves identification of proper names in text, and classification into a set of predefined categories (classes) of interest. For instance, we consider major NEs (as defined in MUC-7) including person name, location, organization, date/time expression and measurements (percent, amount etc).

3.1.1 What NE is NOT

- NE is not event recognition.
- NE recognizes entities in text, and classifies them in some way, but it does not create templates, nor does it perform co-reference or entity linking, though these processes are often implemented alongside NE as part of a larger IE system like search engines or question-answering systems.
- NE is not just matching text strings with pre-defined lists of names. It only recognizes entities, which are being used as entities in a given context.
- NE is not easy task!

Why it is not easy task? Because it involves many pitfalls, even a system is well trained.

3.2 Available approaches

As human finds daily life problems as he faces, in the past several approaches were used (and still being used) to attain the results. We can divide them into two major categories.

3.2.1 Rules based NE

Hand constructed rules by a linguistic (or domain specific) specialist are used to build rules for the tagging. Even some specialist programs, e.g. Brill’s tagger [8], can build
these rules semi-automatically. These rules are more similar to a language grammar defining parts of the speech in the text like noun, verb, subject, object and so many others. Each part is a class here and one or several rules are defined to detect a tag out of text. SPRACH-R is another bright example of rule-based NE system [9]. Basic idea behind such systems is like multiple if conditions in a computer program.

IF $CAPITALIZED\_WORD$ AND PRECEEDED BY “Mr.” THEN PERSON\_NAME

Although these systems outperform in accuracy, but there are several drawbacks of rule-based systems which make them a bad choice for today’s computational linguistics people.

- Too many rules to be defined, each has some exceptions (too much manual work involved).
- Only domain specialist can define these rules
- For each natural language separate rules must be defined because of their grammars.
- Rules might be domain specific and if domain changes, new rules to be written.

Today we have machines, which must be utilized for such jobs; for the large corpora rule-based systems do not perform well because the corpora might have different domains (data-sparseness problem). Sometimes such systems use Gazetteers or dictionaries for increasing accuracies, but it usually does not make a big difference.

3.2.2 Statistical NE

Statistical analysis and probability applications on training data can give good hint about the named class of a word node. Literally, counts over the occurrence in the training and co-occurrence of the patterns (as language models), binary feature extractions & manipulation (as in decision trees) and maximum entropy models are most popular. As in rule-based systems, stochastic based systems also have some drawbacks like lower accuracy and locally limited features. But there are more benefits than its drawbacks, few of them are:

- Formally well defined, mathematics behind the counts.
• Minimal manual work is required; i.e. training and parameter tuning.
• Easily portable to other languages, tokenizer required for each language.
• Much faster than rule based systems.

3.3 LSVLM approach

As a classification problem, similar to speech recognition, a stochastic approach is the most appropriate to go for. HMM based language modeling is one of the most successful techniques where the models are trained on human generated (supervised training) data and then tested for test data, ideally on domain specific to the training data.

3.3.1 Learning

After pre-processing the cleaning training data it is converted into system-specific format, corpus index, for the learner module. Out of corpus index, training process generates statistical language models based on event counts and defined window-size. Count trees, a proprietary data structure at LSV, are the format of these language models. An efficient technique has been implemented to query the tree nodes for its children or its branches. Few examples of the language models are given below.

Calculate probability of current class given previous word, previous word class and current word:

\[ P(C_0 \mid w_{-1}, C_{-1}, w_0) \]

Calculate probability of current class given previous word-class and current word:

\[ P(C_0 \mid C_{-1}, w_0) \]

Our model takes into account solely the history pattern and ignores (for the simplicity) the features of these word nodes. It begins from the longest logically valid history and falls back to the shorter history.

3.3.2 Unknown words

For a small training, it is quite possible that test data item was never observed and does not exist in the system word-vocabulary. Theoretically, it has no choice but fail (because we do not care the features of the word) here but to behave human-like system, it adds
such a word in word history with zero total observations. But then we cannot use it as a history item, i.e. this would return zero

\[ P(C_0 | w_{-1}, C_{-1}, w_0) \]

Then we have to ignore the current word and use

\[ P(C_0 | w_{-1}, C_{-1}) \]

### 3.3.3 Probability calculation

Probability calculation is performed as suggested by the two parts of the equation 2.1 (chapter 2). System takes the main LM (with the longest history) for each term of the equation and adding the score to the older hypothesis.

More details can be found in implementation chapter and/or in appendix A.

### 3.3.4 Smoothing and Backing-Off Schemes

As shown in the learning subsection, we need to pass a discounting parameter in LM definition file; the parameter cannot be said as a correct and must be optimized on the training data. It must be \( > 0 < = 1 \) (greater than 0 but less or equal to 1). Often 0.7 is a good value to start, but can test it with increasing or decreasing it.

Backing off schemes can be thought of as; shorter and shorter than that … till the shortest possible history available for the conditional probability.

### 3.3.5 Decoding

Our decoder module is an example of single pass heuristic-stack-decoder, which has no cost estimation for the path evaluation. Complexity of the decoder is \( O(m * n) \) where \( m \) is the window size (usually 3 or 4, it is length of HistConfig) of the sentence and \( n \) is class_vocabulary size. After adding each \( m \) nodes (or total number of hypotheses reaching to MAX_HYPO_ALLOWED) it filters the results and takes the top \( K \) hypotheses for the next iteration. If the next sentence indicator “<s>” is found in words it flushes its entire stack with given tags.

### 3.3.6 Evaluation and Conclusions
Results of the tagger are tested against its training data and later on *unseen* test data. If we can regenerate the original training (very low discounting parameter must be passed) and decoded correctly that indicates that system is working as expected. On unseen data it has reasonable recall and precision for English and German language but for Lithuanian it requires further investigation. Further evaluation is being done which requires discounting parameter to be tuned to improve the results.
4 Chapter - Implementation

This chapter includes some details and overview of the overall system.

4.1 System architecture

As mentioned earlier that NE recognizer is a part of LSVLM toolkit and it would be better if we mention overall architecture of System and interactions with other modules. The toolkit itself included a set of tools for pre-processing the training data, training of the system and a configuration files for building the language models and backing off models. Language modeling factory loads these hand crafted configuration files, training data from the tree data structures to quickly offer the calculations to the stack decoder. The following illustration shows the overall toolkit and system architecture.

![Diagram of system architecture](image)

*Abbildung 4-1 LSVLM NE Tagger and only related modules shown*

The diagram covers both major use cases (training, testing) of the system. Here training is handcrafted annotated data; document collection is corpus to be annotated by the tagger.
4.1.1 Description
As a first step the documents, in whatever format they are (i.e. XML, HTML or TXT), are processed to fetch out only pure text, then it is converted into tokens by the tokenizer. These tokens further get their integer representation for easier and faster manipulation. If training is the use case, then according to the configured histories knowledge is generated in count-tree format. During the test use case these language models are loaded from these trees to generate probabilities. Then these probabilities are used to generate HMM states out of test tokens. Decoder module then finds the optimal state path. Final output can be generated as tagged document or bi-column word-state format. By default only top-scorer hypothesis is printed, an extra parameter can be passed to the tagger for printing N-top hypothesis.

4.1.2 Use cases

4.1.2.1 Training

![Abbildung 4-2 Use-case Training the system](image)

System is overall system, which has command line set of tools to interact with user. In order to train the system, the user needs to prepare the training data and history setting files. History settings tell the learner how many and which tokens to take into account for learning from the training data.

4.1.2.2 Testing

![Abbildung 4-3 System in action (testing)](image)

Testing the system involves some preprocessing of test data, language model and parameter definition, backing-off schemes and history configuration definition.
4.1.3 Class diagrams

Image 4.1.3 UML class diagram for LM (part-1, part-2)
4.1.3.1 Examples (Structures & Code)

4.1.3.2 Vocabularies
Vocabularies, namely word and class, are generated out of training data by identifying them uniquely. For each language, these must be generated from the respective training data of the language. The data snippets in the following tables are the two real examples of these vocabularies (English). The first column explains how many tokens are in the vocabulary.

<table>
<thead>
<tr>
<th>Word vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td># Vocabulary 12641</td>
</tr>
<tr>
<td>temporarily</td>
</tr>
<tr>
<td>F-14A's</td>
</tr>
<tr>
<td>pesky</td>
</tr>
<tr>
<td>flaw</td>
</tr>
<tr>
<td>flat</td>
</tr>
<tr>
<td>Margaret</td>
</tr>
<tr>
<td>flap</td>
</tr>
<tr>
<td>explosive.</td>
</tr>
<tr>
<td>flag</td>
</tr>
<tr>
<td>dive,</td>
</tr>
<tr>
<td>write-ups</td>
</tr>
<tr>
<td>Martin's</td>
</tr>
<tr>
<td>eight</td>
</tr>
<tr>
<td>lead</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class vocabulary (NE classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td># Vocabulary 8</td>
</tr>
<tr>
<td>TIME</td>
</tr>
<tr>
<td>PERSON</td>
</tr>
<tr>
<td>PERCENT</td>
</tr>
<tr>
<td>MONEY</td>
</tr>
<tr>
<td>OTHER</td>
</tr>
</tbody>
</table>
### 4.1.3.3 History Configuration (HistConfig)

How many previous word tokens or classes should be taken into account to guess the next word or class? Typical HMM based systems follow the idea of previous tokens to calculate the probability for the next token. Our technique, of specifying how long the previous tokens-history should be passed in conditional probability equation, is very flexible. History configuration file is passed to the flexible count tree for generation of the tree based on the given patterns.

Abbildung 4-4 a training sentence, combination of word and class tokens

<table>
<thead>
<tr>
<th>Ex-1. History configuration (HistConfig)</th>
<th>Ex-2. History configuration (HistConfig)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(can be translated as: It’s length is four, consider previous word and its class ALSO previous to previous word and its class)</td>
<td>(can be translated as: It’s length is three, consider previous class ALSO previous to previous word and previous word’s class)</td>
</tr>
<tr>
<td># HistConfig 4</td>
<td># HistConfig 3</td>
</tr>
<tr>
<td>Hist[0] w0</td>
<td>Hist[0] c0</td>
</tr>
<tr>
<td>Hist[1] c0</td>
<td>Hist[1] w1</td>
</tr>
</tbody>
</table>

Image 4.1.3.23.1 hand crafted history configurations, preprocessing (HistConfig examples)
4.1.3.4 History configuration for the Tagger

The format of history configuration remains same at the decoding time, except internal interpretation of it. Language modeling factory automatically reverses the order of the history items for the decoder. In case the history provided at the test time is longer or incompatible to the training HistConfig then it automatically trims the longer history and take the shorter but compatible (if available) part. For example if training was made with w0c0w1c1 all the subsets of this configuration are compatible HistConfigs.

For example, the simplest model would be the simple unigram P(w0) where the token frequency, in the training data, is the only criteria and we do not care the words in its context that would look like Ex-3. But if we want to make it simple bi-gram, we need to specify that previous word should be as history as in Ex-4.

<table>
<thead>
<tr>
<th>Ex-3. History configuration (HistConfig)</th>
<th>Ex-4. History configuration (HistConfig)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(can be translated as: It’s length is one, consider previous word and its class ALSO previous to previous word)</td>
<td>(can be translated as: It’s length is two, consider previous word ALSO previous to previous word)</td>
</tr>
<tr>
<td># HistConfig 1</td>
<td># HistConfig 2</td>
</tr>
<tr>
<td>Hist[0] w0</td>
<td>Hist[0] w0</td>
</tr>
<tr>
<td></td>
<td>Hist[1] w1</td>
</tr>
</tbody>
</table>

*Image 4.1.3.24 hand crafted history configurations, preprocessing (HistConfig examples)*

4.1.3.5 Language model configuration

A language model configuration file consists of several parts, which are necessary for the LMFFactory to work properly.
As seen in the language model instance in above image, Name property, this is arbitrary string as any variable name to refer it. Type field indicates LMFactory that which class is the appropriate to load for the score or probability calculation, discounting parameter is used to normalize the score; Tree is a knowledge tree, built during the training session. History configuration file can be referred by HistConfig property. Most important tag is the BackOffLM, which tells the LMFactory “take into consideration to this LM, in case of failure to calculate something reasonable”. Usually the main language model has the longest history configuration but the backing off LM should have a shorter history. In the next section we show how this mechanism works.

4.1.3.6 Backing off in Language Models

For the HMM decoder we need two language models according to equation shown in background chapter.

\[ P(N \mid S) = \prod_{i=1}^{n} P(W_i \mid W_1 \ldots W_{i-1} \ N_1 \ldots N_i) \cdot P(N_i \mid W_1 \ldots W_{i-1} \ N_1 \ldots N_{i-1}) \]

We have to provide two separate hierarchical model definitions as below (4.6).
Abbildung 4-6 LM parameters, and falling back and LM properties
4.1.3.7 How the LMFactory actually works

The activity diagram shows how the LMFactory is created and started then routs and loads the language models.

![Diagram of LMFactory process]

Language models definition file is passed at the start time, out of this definition factory is able to load all the hierarchy of language models as well as trees and other related configuration.

4.1.4 Decoder

Main task of decoder module is to find out the best scoring class for the word node in a sequence. It looks up by extending the word node with all (until a specific length) possible sequences and by scoring the whole sequence by multiplying the scores.
Our custom stack decoder module takes the word sequence as input and builds a window size as customized by the user. As we can see, the tree of nodes expands exponentially, so it is necessary to filter out the bad results (scoring below a threshold value). The decoder automatically flushes the best hypothesis for a word sequence as soon as limit of its maximum number of hypothesis approaches.

The algorithm of the stack decoder goes as below:

For each word Wi in the input IDX
   For each Hypothesis h in list of OldHypotheses OR first entry
      For each NE class Cj in Class Vocabulary
         If first entry then
            Create new Hypothesis h for expansion
         Else
            Take h for expansion
         End if
      // first part of equation of conditional prob.
   For each item d according to HistConfig1 (for LM1)
get word or class (W i+d) OR (C j+d)
add to the path1
End for
Score the LM1

// second part of equation of conditional prob.
For each item d according to HistConfig2 (for LM2)
    get word or class (W i+d) OR (C j+d)
    add to the path2
End for
Score the LM2

Update the score for hypothesis H and update paths
Add to OldHypotheses list
End for
If MaxHypo limit of OldHypoThesys then
    Filter and keep top N
End if
If next sentence
    Print results and Reset OldHypoThesys
End if
End for
End for

Image 4.1.3.64.1.4-3 – Decoding algorithm as implemented in the system.
5 Chapter - Experimentation

5.1 Data Sets
As any other machine learning application (given it is supervised learning), LSVLM NE Tagger also requires the training data. Data was available for 3 natural languages where for English and German was used provided standard MUC-7 (MUC Task). For Lithuanian language it was manually prepared using web-news from website of leading news site of Lithuania [Courtesy Ms. Lina Čepurnaitė]. These news articles were manually, with full care, annotated and rechecked according to MUC Task definition (using Mitre Callisto) [10]. For training and testing, we had separate data sets.

5.1.1 Training and Test Data
For English and German we used the training provided in MUC-7, while ignoring the events and related stuff. For Lithuanian we had 50 news articles as training and 150 articles for the test purpose. There were some other divisions of the data for the testing purpose and estimation of the discounting parameter using 80%-20% formula.

5.1.2 Porting to Multilingual
Test and training data had to be pre-processed for the tagger; the same tool was used to tokenize the stream, which was not a good idea for the Lithuanian language because of its strange endings of potential NEs in different context. Even though the system should theoretically work with any natural language given that, the same tokenizer is used for training and test purposes. We have not tested with any other languages, it would be interesting to investigate the languages where alphabets change their look and size according to context, i.e. Chinese, Arabic, and Urdu etc.

5.2 Experiments
Initial experiments were held with different criteria to observe the behavior of the system without proper automated of evaluation.

5.2.1 Effect of Training size
Size of the training had significant effect on accuracy of the system, as the observations were unavailable when the training size was 20% of the total data. On 100% of training, still it was felt that unseen events are quite common.

5.2.2 Effect of discounting parameter

Discounting parameter was another important factor, which we had to keep the minimal, as we had not enough of “too often” observations. Therefore, we could not observe much difference. The discounting parameter near 1 had worst results and near to 0 had the best.

5.2.3 Effect of Vocabulary Size

With smaller vocabulary size (minimal training), it had to add unseen events in its dictionary as incoming events with zero count. Larger the vocabulary we had easier for the system was to look into the counts and children of word nodes. If new word was added, it had a normal distribution of probability (according to class vocabulary size).

5.2.4 Effect of History Size

Length of previous words and class tokens to be taken into account had a lot of meaning for the system. History of two previous nodes and their classes had the maximum acceptable range, but extensions to the history length worsen the results. It is proof of Markov’s “rule of few previous tokens” modify the next token. It was still possible to define a longer (tried length=6) and possible to skip few of the previous nodes. Performance of the application (speed/ram) got worst when a longer history was specified.

5.2.5 With limited backing off

Although it is possible to have between 1 and many backing off models each with different history configuration. We have tested with 3 backing off language models for
the main language model and then Zero language model for the worst case. This requires more investigation to find out more about the less language models.

5.3 Evaluation Strategy

We did some evaluation using custom tool (for evaluating F-measure) but it was not clear where the things go wrong therefore, extensive evaluation was not possible. Main criteria for the “truth-proving” were to regenerate the training data out of a trained system with same data.

Extensive testing and evaluation requires a lot of time and effort, which was not required for this thesis work. As the system provides flexibility of specifying backing off schemes and related parameters it is enough.
6 Chapter - Conclusions and future work

6.1 Observations and Gain

As the model gives opportunity to specify, theoretically infinite number of backing off schemes for the tagger has much more possibilities to grab the correct model during fall back to the weaker model. During development and experiments, the following observations have been made:

- Too far in history of previous tokens does not give much gain in accuracy but complexity
- More the observed events are seen higher the accuracy is
- Grammar-strict language would require less training than grammar-free
- Language with dynamic noun modification has lower accuracy hence requires some kind of smart-trimmer or requires much more training as compare to language where nouns do not change their endings. For example in Lithuanian language city/location name is changed according to the subject. city name Vilnius could have Vilniuje, Vilniaus varied names)
- Discounting parameter plays major role in smoothing, especially new word cases and too often words
- Word features are important for the numerical entities, as Nymble considers
- As in vocabularies are generated out of training data, at pre-processing step of the test process, it is possible to have any type or number of classes. This opens the doors of applications like biological entities (e.g. protein, DNA, RNA, cell-type etc) recognition in research material.

In addition to having our own named entity tagger, major gain lies is flexibility it gives especially LM definition and the backing off schemes.

6.2 Conclusions

Statistical language modeling has upper hand in techniques for the named entity recognition as well as other application like speech recognition. LM Smoothing is a core part of any toolkit as in LSVLM. Flexible backing-off schemes extend the smoothing strategies gaining more precision. Such a stochastic system could gain more if hybridized with some manual schemes like feature constructing. Some of the parameters like
discounting parameter, are usually domain specific, cannot be determined easily like, for optimal parameter we need more experiments and training data from different domains. Nymble lead us throughout the research and development phases, adding simplicity in backing off schemes would give new meaning to the further work.

Although during planning and development phase of the system it took some time for building, background and studying related literature. Finally, we have our own product for some future research work.

6.3 Future work

The system fulfills the requirements defined at the beginning of the thesis work and is highly customizable for any sort of functionality. To take full advantage out of it, I must say that it is not perfect yet; it requires some work on optimizing decoder module to decrease the complexity. Overall system can be optimized for minimizing the system resources usage and increase the performance. To claim fully natural language independent it requires some effort in preprocessing and tests on other languages. As described in observations, it would require smarter trimmer for the languages having modified entity-names-endings.

As it was supposed, next steps for the work would be to integrate the system to the Question-Answering system (QA) being developed at LSV group of Saarland University. At answer extraction step of the QA system, it requires the answer type (according to question type i.e. what, who, where, how much, what percentage, when types). At this stage, named entities play a major role, as they could in any other information extraction task.

Main task would be to obtain or prepare domain specific training. Then defining appropriate language models and finding optimal discounting parameters. Input and output formats should be agreed and customized using available tools in LSVLM toolkit. To automate the tagging it would require some scripts to keep working on any cluster of computing power.
Another possible extension would be to offer the service over the web-service so that we could keep our software at home and still serving other research groups.
7 References & Literature


[8] Brill’s POS Tagger http://research.microsoft.com/%7Ebrill/


8 Appendix A

Troubleshooting and Working with the Software (FAQ)

Q.1 Where to get LSVLM toolkit?

Answer: LSVLM and all its modules (including this thesis work) become part of LSVLM toolkit, which cannot be disclosed at the stage of writing of this document. You may contact LSV research group at Saarland University http://www.lsv.uni-saarland.de for the details. No queries are responded by the author.

Q.2 Why my build fails with the following error:

```
$ make tools
.
.
.
.
.
.
.
ar ts liblmd.a
g++ -static test_prob.od liblmd.a -o test_prob.d -lm -lnrd
/usr/lib/gcc-lib/i586-suse-linux/3.3.1/../../../../i586-suse-linux/bin/ld:
cannot find -lnrd
collect2: ld returned 1 exit status
make: *** [test_prob.d] Fehler 1
rm compress.od zerolm.od hypothesis.od sectoutstream.od corpusline.od
histconf.od params.od lmfactory.od dynparams.od sectinstream.od
align_cntmgramlm.od classlm.od util.od loglinearlm.od ftree2d.od cntmgramlm.od
indexcorpus.od linearlm.od classmap.od vocabulary.od
```

Symptoms:
Library “Numerical Recipes” (libnr.a and libnrd.a) is missing from LIBPATH.

Solution:
Provide nr library in the lib search path (e.g. /usr/lib)

How?
- Extract the contents from provided nr.tar bundle (using: tar xvpf nr.tar)

- execute make_lib script.
- copy to the lib: cp nr/libnr*.a /usr/lib
Q.3 How to model the LMs and test the system?

Answer: Modeling the HMMs (Testing the system)
This step is limited to define language models and test the system using LM file configuration.

Following steps to do (In all configuration files variable names are FIXED and are case sensitive):

For each backing off scheme we need to have a training/count tree, discounting parameter and next fallback scheme. These settings can be placed in language model file in only specified format. Each model consists of some backing off schemes, which are falling back alternatives when no decoding estimate can be found on a longer history.

0) generate the vocabularies (use LSVLM or any other custom script)

```plaintext
(contents of class vocabulary file, saying its header and the length in first line)

# Vocabulary 8
TIME
PERSON
PERCENT
MONEY
OTHER
DATE
ORGANIZATION
LOCATION
```

1) Create history configuration file for the tree1 and tree2 (each for relevant LM)

```plaintext
(contents of Main_Hist_LM1)
# HistConfig 2
Hist[0] c0
Hist[1] w0
```

```plaintext
(contents of Main_Hist_LM2)
# HistConfig 2
Hist[0] c0
Hist[1] c1
```

2.1) Create LM configuration file for each LM (real example LM1.lm)
2.2) Real example LM2.lm

```plaintext
# Parameters 1
MainAlignCntMGramLM BiGram
# LMDefinition 6
Name   BiGram
Type   AlignCntMGramLM
Tree   StdTreeLM2_1
BackOffLM UniGram
Disc   0.6
HistConfig main_hist2_bigram_c0c1.his
# LMDefinition 6
Name   UniGram
Type   AlignCntMGramLM
Tree   StdTreeLM2_2
BackOffLM Zero
Disc   0.6
HistConfig c0_unigram_classes.his
# LMDefinition 2
Name   Zero
Type   Zero
# TreeDefinition 2
Name   StdTreeLM2_1
File   tr_lm2_1_bigram_c0c1.cnt
# TreeDefinition 2
Name   StdTreeLM2_2
File   tr_lm2_2_unigram_c0.cnt
```
3) To generate a tree `alig_treehistcnt` tool can be used (syntax):

```bash
cat <corpus_index> | alig_treehistcnt <longest_hist_configs> >
<count_tree_file>
```

for example:

```bash
$cat corpus_index.idx | alig_treehistcnt hist1_bigram_c0w0.his >
tr_lm1_1_bigram_c0w0.cnt.info
```

3.1) To view the tree information `cnt2info2d` tool can be used; the output can be piped to
a file or print on the screen.

```bash
$cat tr_lm1bigram_c0w0.cnt | bin/cnt2info2d > tr_lm1_1_bigram_c0w0.cnt.info
```

4) After building both the language model configuration files and count generated files
we need to test the system.

Syntax:

```bash
$ cat <test_tokens> | <tagger_name> <word_vocabulary> <class_vocabulary>
/history_lm1> < history_lm2> <lm1_config> <lm2_config>
<number_of_hypothesesToPrint=1>
```

Example (will print top 2 hypothsysis for test file test1_eng.txt):

```bash
$ cat test1_eng.txt | tagger.d VWords.wl VClasses.wl
main_hist1_bigram_c0w0.his main_hist2_bigram_c0c1.his lm1_hist_c0w0.m2_exp1.lm
lm2_hist_c0c1.m2_exp1.lm 2
```