5-2011

Finding acronyms and their definitions using HMM

Lakshmi Vyas
University of Nevada, Las Vegas

FINDING ACRONYMS AND THEIR DEFINITIONS USING HMM

by

Lakshmi Vyas

Bachelor of Engineering, Computer Science
Visvesvaraya Technological University, India
2006

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
May 2011

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Lakshmi Vyas

entitled

Finding Acronyms and Their Definitions using HMM

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
School of Computer Science

Kazem Taghva, Committee Chair

Ajoy K. Datta, Committee Member

Laxmi P. Gewali, Committee Member

Venkatesan Muthukumar, Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

May 2011

ABSTRACT

Finding Acronyms and Their Definitions using HMM

by

Lakshmi Vyas

Dr. Kazem Taghva, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

In this thesis, we report on design and implementation of a Hidden Markov Model (HMM) to extract acronyms and their expansions. We also report on the training of this HMM with Maximum Likelihood Estimation (MLE) algorithm using a set of examples.

Finally, we report on our testing using standard recall and precision. The HMM achieves a recall and precision of 98% and 92% respectively.

ACKNOWLEDGEMENTS

There are many people who have had a significant influence on my thesis research work. While it is not possible to list every contribution, I make an attempt to express my gratitude to those who have helped make my work a success.

Dr. Kazem Taghva, my thesis advisor, has been an immense source of knowledge and motivation. I am eternally grateful for his support, patience and guidance throughout my thesis study. I have learned so much in the last year of working with him.

I would like to thank the graduate coordinator, Dr. Ajoy Datta, for his vote of confidence on everything I have ventured to do during my Masters program. I would like to convey my sincere appreciation and gratitude to the members of my thesis advisory committee, Dr. Laxmi P Gewali, Dr. Venkatesan Muthukumar and Dr. Ajoy Datta. Their ready acceptance to serve on my committee has been a great source of confidence. I consider myself privileged for the opportunity to work under their guidance.

My acknowledgements would be incomplete without a mention of the support my husband and family have given me. I am humbled by their constant faith and encouragement without which I wouldn’t be where I am today.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The thesis discusses a method of Information Extraction called Hidden Markov Models (HMMs) [6]. Information Extraction can be carried out by the use of HMMS and other standard approaches such as hand-written regular expressions, Naïve Bayes [12] and Conditional Random fields (CRF) [13]. The main focus of the thesis is to understand Hidden Markov Models. It also looks into the working of the Viterbi algorithm and the use of Maximum Likelihood Estimation (MLE) [14].

Information Extraction is the task of retrieving structured information from unstructured or semi-structured documents. More specifically it is the task of extracting data that is relevant with respect to a category and context from a collection of documents in a certain domain. We look into the problem of finding acronyms and their definition in text using the formal method of information extraction i.e. Hidden Markov Models (HMMs).

Acronyms are a word formation that is composed of the first letters of words in a series of words. These acronyms are known to cause considerable confusion to readers who are unaware of its origins. It is therefore important to ascertain the acronym and what it stands for. The problem is one that has been studied before [3]. The algorithm [3] is based on an inexact pattern matching algorithm applied to text surrounding the possible acronym. Evaluation shows that the algorithm performs well, however, we go on to show that the use of HMMs for the same task overcomes some of the limitations of the ad-hoc methodology such as the length of the acronym, use of special characters in the acronym etc.

The idea of using HMMs to the task of extracting acronyms and their definitions is based on the significant success it has had to other language related tasks, including speech recognition [Rabiner 1989], text segmentation and topic detection [van Mulbregt 1998]. Like finite state automaton HMMs are composed of a finite set of states. HMMs are probabilistic tools that are used to model a sequence of most likely states given an observation sequence and other model parameters. The probabilities associated with every state in an HMM model are set using Maximum Likelihood Estimation (MLE) on tagged documents and the most likely sequence of states for the input data is decided by the Viterbi algorithm. We evaluate our results by using precision and recall.

1.1 Outline

Chapter 1 looks into Information Extraction in some detail and also explains the working of the Acronym Finder algorithm [3]. Chapter 2 discusses the working of the Viterbi algorithm and the statistical method of estimating model parameters using Maximum Likelihood Estimates (MLE). A detailed account of the design of the HMM model used for the task and other implementation specifics are discussed in Chapter 3. The methods used to train the model, test it and an overview of the results obtained is in Chapter 4. Chapter 5 summarizes and concludes this thesis.

CHAPTER 2

INFORMATION EXTRACTION AND AFP EXPLAINED

Information Extraction (IE) [15] can be defined as the task of extracting relevant information from the actual text of documents. Information Extraction is of great significance to companies that rely on drawing inferences from data, using transaction histories and archives of other happenings. Information Retrieval (IR), on the other hand, is the task of finding relevant documents from a collection of documents. It is likely that an Information Extraction system built for a specific need is preceded by some Information Retrieval task to categorize relevant documents from a larger collection.

A clear distinction between these two processes can be drawn by looking into an example. A system that classifies incoming emails as ‘Spam’ or ‘Not Spam’ is an example of an Information Retrieval system. These systems are quintessential in today’s day and age and categorize email messages into the above mentioned categories by looking into information encapsulated in the email headers.

Every email message consists of two parts – the body and the header, used by servers on the Internet as they deliver the message. The header tells us where the email is coming from, which route it has come through and the name of the different routing points. The names of some of these fields are Return-Path, Message ID, X-IP, X-UIDL. The IR system first tokenizes the header and analyzes these fields in some detail to ascertain if they are genuine and reliable. Ones that are inferred as Spam are categorized accordingly.

The system described above does not categorize emails based on the semantics of the body of content. Features such as Multiple Inboxes, provided by Google, allows a Gmail user to segregate their inbox. The segregation criteria can be a myriad of things such as

the Sender of the email message, the subject line, the priority, the domain name of the sender's email ID etc. Such a task is difficult for an IR system but not for an Information Extraction system. Information Extraction is not a stand-alone task that analysts engage in. It is an abstraction over a larger task intended to produce results without human intervention.

Extracting information from text to understand implicit patterns dates back to the early days of Natural Language Processing (NLP). In 1979, DeJong from Yale University developed a system called FRUMP. This NLP system analyzed news stories to generate a summary for users logged into the system. This system is reminiscent of modern day IE, since the generated summaries are essentially templates filled in by Fast Reading Understanding and Memory Program (FRUMP) [16]. DeJong's system uses hand-coded rules and the data structure that was populated by the system was called 'script', a term coined by his advisor, Schank. Other early attempts to extract information include the work of Silva and Dwiggins [17] for identifying information about satellite flights from multiple text reports. The system that was developed for this was Prolog based.

To encourage the development of IE techniques, in the late 1980s and early 1990s the US Government, DARPA, organized a series of Message Understanding Conferences (MUC) as a competitive task with standard data and evaluation procedures. IE was separated into several different tasks in later conferences, such as Named Entity (NE) task, Relation Extraction (RE) task and Scenario Template (ST) task. These conferences established a competitive environment that enabled rapid transfer of ideas and techniques and thus benefitted IE research. In the later years of the conference the government

focused its efforts on reducing the amount of human undertaking involved in generating rules for IE. Much success was achieved in this area.

Research in IE has continued to grow over the years since MUC. The definition of IE has broadened gradually to include many types of tasks that differ in their complexity, amount of resources used, training methodologies, etc. Recent approaches to IE also include incorporating machine-learning, including global information into IE systems than was possible with hand-crafted pattern based approaches.

Tim Bernes-Lee, inventor of the World Wide Web WWW, refers to the existing Internet as a document web. The Internet has a vast amount of data available but is very hard to manipulate and analyze by computers as it is in unstructured form. The task of IE is to transform this unstructured data into something that can be understood and manipulated. IE, therefore, is the process of extracting sub-sequences of text from this human-readable text form to populate some sort of a data base.

There are many approaches to IE, some of which are Hand-written regular expressions, pattern matching, use of classifiers such as Naïve Bayes, sequence models like Hidden Markov Models, Chained Markov Models, Conditional random fields, etc.

In this thesis, we use Hidden Markov Models for Information Extraction (IE). The inspiration to use HMMs came from Dayne Freitag and Andrew McCallum [1]. Their experiments were based on two real world data sets; on-line seminar announcements and Reuter's newswire articles on company acquisitions. As HMMs have strong foundations in statistical theory there are many established techniques for learning the parameters of the HMM from labeled training data. Freitag and McCallum [1] got impressive results

when using HMMs for their specific tasks. The design of the HMM models used by them have the following characteristics:

- Each HMM extracts just one type of field. When multiple fields are to be extracted from the same document, a separate HMM is constructed for each field.
- The entire document is modeled without any pre-processing to segment the document into smaller parts. The entire text of the training document is used to train transition and emission probabilities.
- They contain two kinds of states – background states and target states. Target states are intended to produce the tokens we want to extract.
- The models of the HMM are not fully connected. The restricted structure captures the context that helps improve extraction accuracy.

The problem discussed in this thesis is to extract acronyms and their definitions (available in the same text) from a collection of documents (not necessarily from the same context). An HMM is designed for this specific purpose and the model is trained using labeled training data.

The problem is one that has been done before using an inexact pattern matching algorithm applied to text surrounding the possible acronym [3]. A lexicon is not used to validate words that are picked up by the program which essentially means that the spelling of the word is of little consequence to us. The Acronym Finder program consists of four phases namely initialization, input filtering, parsing the remaining input into words, and the application of the acronym algorithm.

Initialization

The input of the algorithm is composed of several lists of words, with the text of the document as the final input stream. These inputs are:

- A list of *stopwords* – words like “the”, “and”, “of”, that are insignificant parts of an acronym. These need to be distinguished from other words that make good matches for the acronym definitions.
- A list of *reject* words – words in the document that resemble acronyms but are words that are frequent in any document and that are known not to be acronyms. For e.g. “TABLE”, “FIGURE”, Roman Numerals.
- The text of the document to be searched.

Filtering the input

The input stream is preprocessed to remove lines of text that consists of words that are all uppercase (e.g. headings, titles). Identify a candidate acronym and compare it against the list of reject words. Once it is elected as a candidate a context window is selected around it. The text window is divided into two sub-windows, the pre-window and the post-window. The length of the sub-window is twice the number of characters in the acronym.

Word parsing

In order for the algorithm to work efficiently different types of words have to be identified and a priority should be assigned to each one of them. Stopwords (s) are normally ignored in traditional text but they can sometimes be part of the acronym. Precedence should be given to non-stopwords over stopwords in the matching process. Hyphenated words are a special case of words in which either the first letter of the word

(H) or every first letter of the hyphenated set of words (h) correspond to the acronym. Both cases need to be tested to find the best match. Acronyms (a) can themselves be part of the definitions of other acronyms. It is therefore necessary to see if the acronym that is part of the definition is the same or a different one. Words apart from the ones that have been defined are normal words (w).

When parsing the subwindow two symbolic arrays are generated; the leader array consisting of the first letter of every word and the type array that is composed of the type (defined above) of every word.

Applying the algorithm

The algorithm identifies a common subsequence of the letters of the acronym and the leader array to find a probable definition. For two sequences X and Y, we say that a sequence Z is a common subsequence of both X and Y. For example, X = $beacdab$ and Y =$acbdabe$, then cab is a common subsequence of X and Y of length 3. Notice that the subsequence need not necessarily be together, there can be characters in between. It must be ensured while deriving the subsequence, the order of occurrence of the characters should be maintained as in the original strings. Observe that $acab$ and $acda$ are also common subsequences of length greater than 4. The longest common subsequence (LCS) of any two strings is a common subsequence with the maximum length among all common subsequences. The length of the longest subsequence c[i,j]; where 'i' is the length of the prefix of a string, say X and 'j' is the length of the prefix of the comparison string, say Y; can be found recursively using the formula

$$c[i,j] = \begin{cases} 0 \; if \; i = 0, j = 0 \\ c[i-1, j-1] + \; 1 \;\; if \; i,j > 0 \; and \; X_i = Y_j \\ \max\left(c[i, j-1], c[i, j-1] \; if \; i,j > 0 \; and \; X_i \neq Y_j \right. \end{cases}$$

Example

Consider the text:

The displays use arrays of Organic Light Emitting Diodes OLED to project the image onto a screen contained within the armor much like a rear projection TV.

The pre-window of the acronym OLED is:

displays use arrays of Organic Light Emitting Diodes

leader array [d u a o o l e d]

type array [w w w s w w w w]

acronym is [o l e d]

The length of the LCS obtained by the algorithm is 4 using the equation defined above.

Index for acronym is

o – 1

l – 2

e – 3

d– 4

Indices for the pre-window will be:

d – 1

u - 2

a - 3

o - 4

o - 5

l - 6

e - 7

d - 8

	j	0	1	2	3	4	5	6	7	8
I		y_j	D	u	a	o	o	l	E	d
0	x_i	0	0	0	0	0	0	0	0	0
1	o	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ↖	1 ←	1 ←	1 ←
2	l	0	0 ↑	0 ↑	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ←
3	e	0	0 ↑	0 ↑	0 ↑	1 ↑	1 ↑	2 ↑	3 ↖	3 ←
4	d	0	0 ↑	0 ↑	0 ↑	1 ↑	1 ↑	2 ↑	3 ↑	4 ↖

Figure 1: Dynamic Programming Algorithm for equation c[i, j]

The arrows in the figure indicate how the current value in the cell was selected i.e. if value chosen was one among $c[i-1, j-1] + 1$, $c[i-1, j]$ or $c[i, j-1]$.

The acronym finder algorithm produces all ordered arrangements of indices for all possible subsequences. In our example, the two possible ordered pairs are

(1,4), (2,6),(3,7),(4,8)

(1,5), (2,6),(3,7),(4,8)

These indices are used to construct a vector notation of the possible definitions of the acronym. The vectors of the example we have chosen will be:

[0 0 0 1 0 2 3 4]

[0 0 0 0 1 2 3 4]

The last part of the algorithm selects the appropriate definition from the vectors that were generated. This is done by evaluating the candidate definitions for the number of *stopwords* that are part of the definition, the number of words in the acronym definition

that do not match the acronym, etc. The best possible match for the example we have considered is the second vector [0 0 0 0 1 2 3 4].

This is so as the first vector considers a stopword to be part of the acronym definition. We have discussed that a stopword has lower precedence as compared with a normal word. The definition of the acronym OLED is hence Organic Light Emitting Diode.

CHAPTER 3

ALGORITHMS

In statistics and machine learning, the most important decision to be made is the selection of the model among different mathematical models to best describe the data set. Our choice of Hidden Markov Models (HMM) to model data for the task of Information Extraction was made easy by the study of Dayne Freitag and Andrew McCallum [1]. This chapter explains HMM and also describes the algorithms that we used to extract acronyms and their definitions.

3.1 Hidden Markov Models

Consider a system that has N distinct states S_1, $S_{2,}$ $S_{3....}S_n$. The system undergoes a change of state at regularly spaced time intervals according to a set of probabilities associated with that state. These probability distributions govern the manner in which the system evolves over time. Such a system is referred to as a stochastic process. To predict the probability of the next state that would be traversed, a full description of the system would be required; that is the specification of the current state along with all the predecessor states. This system is otherwise known as a Markov model.

$$pr(q_t = S_i | q_{t-1} = S_j \cap q_{t-2} = S_k \cap \dots \cap q_0 = S_l)$$

An order 0 Markov model is one that takes no consideration of the history. It is commonplace to say that an Order 0 Markov model has “no memory”.

$pr(q_t = S_i) = pr(q_{t'} = S_j)$ for t and t’ in a sequence.

A first order Markov model has a memory size of 1. So the probability of being in state S_i at a time t depends on the state S_j at time t-1.

$$pr(q_t = S_i | q_{t-1} = Sj)$$

An order ‘m’ Markov model is said to have a memory size of m. So the probability of the current state depends on m number of previous states.

The processes explained above could be called observable Markov models since the output is the set of states at each instant of time, where each state corresponds to an observable or physical event. Such a model is very restrictive to be applicable to real world problems. A Hidden Markov model is a Markov model where the stochastic process produces a sequence of observations output from states of the model but the states themselves are not seen. Consider a 3 state Markov model that models the weather of a city [7].

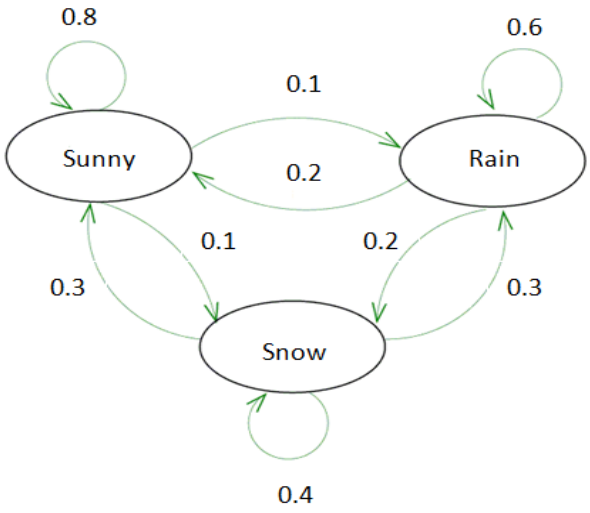

Figure 2: State Transition Diagram

The weather on a particular day can be any one of the three states mentioned below

State 1: Snow

State 2: Rain

State 3: Sunny

The probabilities associated with the weather changing between these states can be written in a matrix as follows

	Snow	Rain	Sunny
Snow	0.4	0.3	0.3
Rain	0.2	0.6	0.2
Sunny	0.1	0.1	0.8

Given these probabilities we can find the probability associated with a sequence of weather states such as 'sunny->sunny->snow->snow->sunny'. The probability is evaluated for the observation sequence, $O = \{S_3, S_3, S_1, S_1, S_3\}$

$P(O|Model) = P[S_3, S_3, S_1, S_1, S_3]$

$= P[S_3] * P[S_3|S_3] * P[S_3|S_1] * P[S_1|S_1] * P[S_1|S_3]$

$= 0.4 * 0.8 * 0.3 * 0.4 * 0.1$ *(assuming that initial probability of S3 is 0.4)*

$= 0.00384$

To someone who is oblivious to the weather conditions, because he is confined to a small closed space, it is possible to draw inferences on the weather based on the way his visitor is dressed i.e. if the guest is wearing a coat (C) or not (D). Consider the probability that the visitor wears a coat is 0.1 on a sunny day, 0.3 on a rainy day and 0.7 on the day it snows. Finding the probability of a certain type of weather q_i can be based on the observation x_i. The conditional probability $P(q_i|x_i)$ can be written according to Bayes' rule as

$$P(q_i|x_i) = \frac{P(x_i|q_i)P(q_i)}{P(x_i)}$$

or for n days, the weather sequence $Q = \{q_1, \ldots., q_n\}$, as well as the sequence of observations $X = \{x_1, \ldots., x_n\}$ as

$$P(q_1, \ldots., q_n | x_1, \ldots., x_n) = \frac{P(x_1, \ldots, x_n | q_1, \ldots., q_n) P(q_1, \ldots, q_n)}{P(x_1, \ldots, x_n)}$$

using the probability $P(q_1, \ldots, q_n)$ from above and $P(x_1, \ldots, x_n)$ of seeing a particular sequence of coat events. The probability $P(x_1, \ldots, x_n | q_1, \ldots., q_n)$ can be estimated as $\prod_{i=1}^{N} P(x_i | q_i)$, when it is assumed for all i that q_i, x_i are independent of all x_j and q_j for all $j \neq i$. The probability of seeing the visitor wear a coat is independent of the weather that we like to predict, so we can disregard $P(x_1, \ldots, x_n)$. This measure is now referred to as Likelihood.

Assume that the person knows that the day he entered confinement was Sunny. The visitor on the next day carries a coat with him. Using this information and the probabilities it is not difficult to analyze what the weather most likely weather condition outside is. This evaluation is done as follows:

Likelihood that the second day is sunny

$$= P(x_2 = C | q_2 = S_3). P(q_2 = S_3 | q_1 = S_3)$$

$$= 0.1 * 0.8 = 0.08$$

Likelihood that it is raining on the second day is

$$= P(x_2 = C | q_2 = S_2). P(q_2 = S_2 | q_1 = S_3)$$

$$= 0.3 * 0.1 = 0.03$$

Likelihood that it is snowing on the second day is

$$= P(x_2 = C | q_2 = S_1). P(q_2 = S_1 | q_1 = S_3)$$

$$= 0.1 * 0.7 = 0.07$$

The highest of these probabilities is 0.08. From the result obtained we find the highest probability is associated with Sunny weather even though the visitor brings a coat with him.

Formally said, it is possible to find the sequence of physical events when a string of observations generated by these events is available with the use of Hidden Markov models. Now that we have understood the basic idea behind HMMs, we delve into some of the specifics.

An HMM is characterized by a 5-tuple $(\mathrm{S}, \mathrm{V}, \pi, \mathrm{A}, \mathrm{B})$, where

- S is a finite set of N states $\{s_1, \dots, s_n\}$. Although the states are hidden, for many practical applications there is some significance associated to the states of the model.
- V is the set of M distinct symbols in the vocabulary of an HMM. The M observation/emission symbols correspond to the physical output of the system being modeled.
- $\pi = \{\pi_i\}$ are the initial state probabilities where $\pi_i = P[q_1 = S_i]$ and $1 \leq i \leq N$
- $A = \{a_{ij}\}$ are the state transition probabilities where

  $a_{ij} = P[q_{t+1} = S_i | q_t = S_j]$, when $1 \leq \mathrm{i,j} \leq \mathrm{N}$.
- $B = \{b_j(o_k)\}$ are the emission probabilities. The observation symbol probability distribution in state i is $B = \{b_i(k)\}$, where $b_i(k) = P[\{o_k \; at \; t | q_t = S_i\}]$ and $1 \leq i \leq N$ and $1 \leq k \leq M$.

We use $\lambda = (A, B, \pi)$ to denote the complete parameter set of the HMM. The constraints on the HMM are

$$\sum_{i=1}^{N} \pi_i = 1 \; for \; 1 \leq i \leq N$$

$$\sum_{j=1}^{N} a_{ij} = 1 \; for \; i = 1,2, \ldots N$$

$$\sum_{k=1}^{M} b_i(o_k) = 1 \; for \; i = 1,2, \ldots N$$

For the model to be useful in real world applications, three problems need to be addressed. These problems are:

1. **Evaluation:** Evaluating the probability of an observed sequence of symbols $O = o_1 o_2 \ldots o_T$ $(o_i \epsilon V)$, given a particular HMM, i.e. $p(O|\lambda)$.
2. **Decoding:** Finding the most likely sequence of states for the observed sequence. Let q = $q_1 q_2 \ldots q_T$ be a sequence of states. We want to find $q^* = argmax_q p(q|O, \lambda)$.
3. **Training:** Adjusting all the parameters λ to maximize the probability of generating an observed sequence, i.e., to find $\lambda^* = argmax_\lambda p(O|\lambda)$.

In this thesis, we solve problem 2 with the help of the Viterbi algorithm and problem 3 using Maximum Likelihood Estimation (MLE). Problem 1 is not addressed in this study.

3.2 Viterbi Algorithm

The Viterbi algorithm is used closely with Hidden Markov Models (HMMs). It is most useful when calculating the most likely path through the state transitions of these HMMs over time. The motivation behind this algorithm arises from the fact that, given N states and T moments in time, calculating the probabilities of all transitions over time would be

N^T probability calculations. The observation made by Viterbi is that for any state at time t there is only one likely path to that state. Therefore, if several paths converge at a particular state at a time t, instead of recalculating them all when calculating the transitions from this state to states at time $t + 1$, the less likely paths can be discarded and the most likely paths used. When this is applied, it reduces the number of calculations to $T * N^2$ which is of lesser complexity than the method discussed earlier.

To illustrate how the algorithm finds the shortest path, we need to represent the process pictorially. This can be done in a trellis diagram (Figure 3). In a trellis, each node corresponds to a distinct state at a given time q_t, and each arrow represents a transition a_{ij} to some new state at the next instant of time. The most important aspect of the trellis diagram is that for every possible state sequence there is a unique path through it.

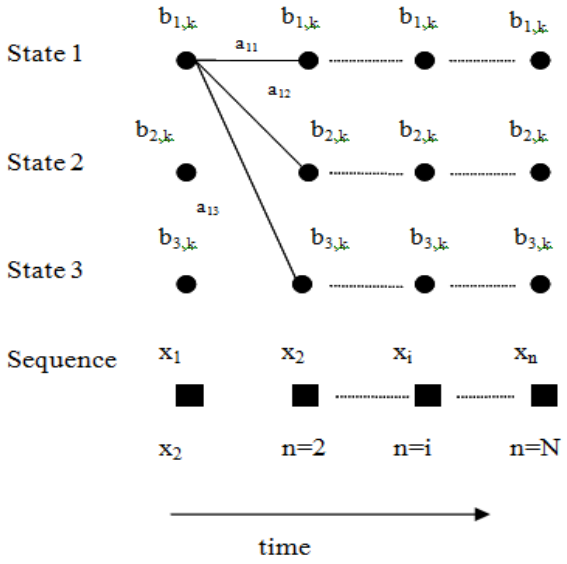

Figure 3: Trellis Diagram

The idea of the Viterbi algorithm is to find the most probable path for each intermediate state and finally for the terminating state in the trellis. At each time only the most likely path leading to each state survives. For an HMM with N states the Viterbi algorithm has 4 phases namely Initialization, Recursion, Termination and Backtracking. This makes requires us to define two variables:

$\delta_n(i)$ is the highest likelihood of a single path among all the paths ending in state s_i at a time n.

$$\delta_n(i) = \max_{q_1,q_2,\dots,q_{n-1},q_n} p(q_1, q_2, \dots., q_{n-1}, q_n = s_i, x_1, x_2, \dots, x_n|O)$$

and a variable $\psi_n(i)$ which allows us to keep track of the 'best path' ending in state s_i at a time n.

$$\psi_n(i) = \operatorname{argmax}_{q_1,q_2,\dots,q_{n-1},q_n} p(q_1, q_2, \dots., q_{n-1}, q_n = s_i, x_1, x_2, \dots, x_n|O)$$

1. Initialization

$$\delta_1(i) = \pi_i . b_{i,x_1}, i = 1, \dots., N$$

$$\psi_1(i) = 0$$

where π_i is the probability of being in state i at a time $n = 1$.

2. Recursion

$\delta_n(j) = \max_{1\le i\le N}(\delta_{n-1}(i).a_{ij}).b_{j,x_n}$, $2 \le i \le N$ and $1 \le j \le N$

$\psi_n(j) = \max_{1\le i\le N}(\delta_{n-1}(i).a_{ij})$, where $2 \le i \le N$ and $1 \le j \le N$

3. Termination

$$p^*(X|O) = \max_{1\le i\le N} \delta_N(i)$$

$$q_N^* = \arg\max_{\le i\le N} \delta_N(i)$$

Find the best likelihood when the end of the observation sequence $t = T$ is reached.

4. Backtracking

   $Q^* = \{q_1^*, \dots, q_N^*\}$ so that $q_n^* = \psi_{n+1}(q_{n+1}^*)$, $n = N-1, N-2, \dots, 1$

   In this phase the best sequence of states is got from the ψ_n vectors.

An example of how the algorithm works in the HMM we have designed is explained later in this chapter.

3.3 Maximum Likelihood Estimation (MLE)

The idea behind Maximum Likelihood estimate is to determine the parameters that maximize the probability or likelihood of the sample data. MLE methods are considered to be robust and versatile and so they are used for most models and for different types of data. Using Maximum Likelihood estimation with HMMs to determine the parameters of the model from labeled training data is also known as Supervised Training.

The process of computing the statistical parameters of an HMM involves the calculation of emission probabilities and the transition probabilities that are associated with states. The underlying sequence of states associated with the data is known by the trainer. It is possible to find the number of distinct states associated with the HMM by parsing the file that is used for training purposes. To estimate the parameters of the HMM we maintain a count of the transitions that are seen between states for all the possible combinations of states and also maintain counts for the number of times symbols belonging to the defined HMM vocabulary are generated from states. This count is later used to estimate emission and transition probabilities.

$$Transition\ Probability, a_{ij} = \frac{number\ of\ transitions\ from\ state\ s_i\ to\ state\ s_j}{number\ of\ transitions\ from\ state\ s_i}$$

$$Emission\ probability, b_i(k) = \frac{number\ of\ times\ the\ symbol\ k\ is\ generated\ from\ state\ s_i}{total\ number\ of\ symbols\ generated\ from\ state\ s_i}$$

Maximum Likelihood estimation assigns a zero probability to state transitions and state-emission combinations that are unseen in the training data. This problem if left unchecked can cause erroneous results. It is most often handled with the use of some type of Smoothing technique. In this study, we use Absolute discounting and this is explained in chapter 5.

It must be noted that the restrictions in transition topology placed on the design are reflected in the parameters of the model only if the document is tagged/labeled appropriately.

CHAPTER 4

DESIGN

The design of the HMM to extract acronyms with their definitions is similar to the design chosen by Frietag and McCallum [1] to model two data sets namely, online seminar announcements and Reuter's newswire articles on company acquisitions. The HMM model for acronyms has 4 states:

- Prefix (0)
- Acronym (1)
- Definition (2)
- Suffix (3)

The states of special consequence are the Acronym and the Definition state. These states are also called the target states; the words that are generated by them are candidate acronyms and candidate definitions respectively. To capture context certain restrictions have been placed on the transitions between states. This is illustrated in the figure below

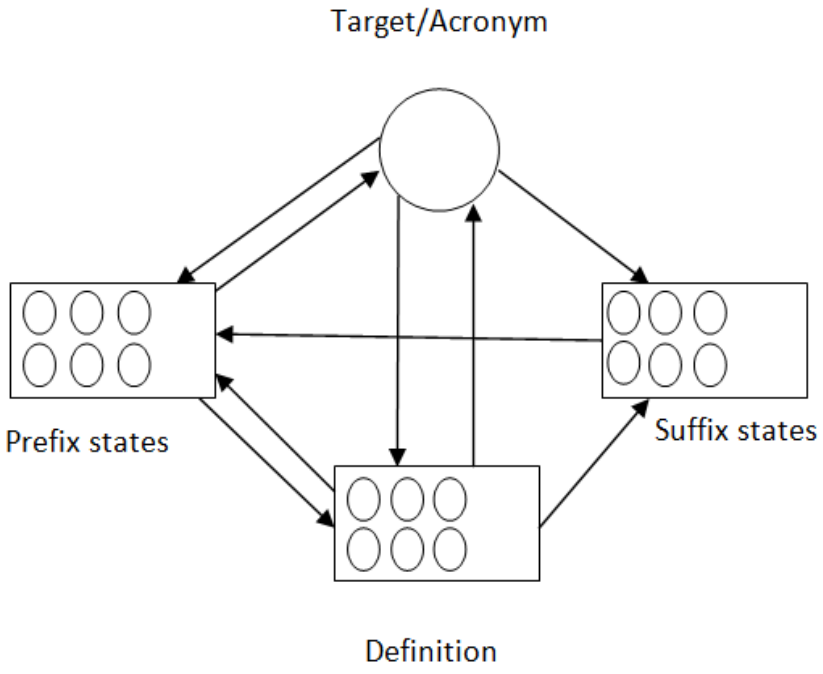

Figure 4: HMM for Acronyms and their Definitions

As can be seen from the figure, the HMM is not entirely ergodic. While tagging data to train the HMM model (for calculating probabilities) it is important to keep the topology of transitions in check. In the event that acronyms occur in quick succession the target states can transition to the prefix state without traversing to suffix states. It is however not possible to reach target states from the suffix state, the transition must happen through the prefix state. The rules states above are what the model implies.

It has been stated in section 3.1 that an essential part of an HMM is a definition of its emission vocabulary. The vocabulary associated with the acronym finder HMM consists of three symbols, each of which represents a type of word. The symbols are

- A – for the acronym that is made of all uppercase letters

- D – for possible definition of the acronym that starts with an uppercase letter
- n – for all other words in the text.

The document collection is first pre-processed before it can be used in the learning module or the decoding module. This processing is explained clearly in the next chapter. When the document collection (pre-processed) is parsed by the routines; every word in the document is translated to one of the above symbols. The routines that have been written do not capture context or semantics; they are only concerned with the sequences of symbols the words translate to.

Maximum Likelihood Estimate (MLE) is calculated on a tagged document of words, rather symbols with states. The probabilities that are calculated in this manner are transition probabilities between all combinations of states and symbol emission probabilities associated with every state. Initial probabilities are set by hand. Let us consider the following statement

The example explains how Maximum Likelihood Estimate (MLE) works in our thesis.

Preprocessing and tagging this statement would result in a sequence such as the one shown below.

The 0

example 0

explains 0

how 0

Maximum 2

Likelihood 2

Estimate 2

MLE 1

works 3

in 3

our 3

thesis 3

The routine that uses Maximum Likelihood Estimation (MLE) reads this file one line at a time. The very first thing that is done by the routine is the translation of words into one of the defined emission symbols. The routine keeps track of the current state and the previous state and keeps a count of the number of times the combination is encountered in the file. In the example that is considered it can be seen that the transition from state 0 to state 0 happens 3 times and the total number of transitions from state 0 in the text is 4. The probability associated with the transition from state 0 to state 0 is therefore 0.75. In a manner similar to what is described the emission of symbol ‘n’ from state 0 happens 3 times in our example and the symbol ‘D’ is generated once from state 0. The probability associate with the state 0 for emitting symbol ‘n’ is the number of times symbol ‘n’ is emitted from state 0 divided by the total number of emissions from state 0 i.e. 0.75.

```
InitPr 4
0 0.7
1 0.15
2 0.15
3 0
OutputPr 12
0 A 0.0369804
0 D 0.0928197
0 n 0.870133
1 A 0.997147
1 D 3.06805e-05
1 n 3.06805e-05
2 A 0.00596771
2 D 0.891985
2 n 0.101324
3 A 0.0211513
3 D 0.0781888
3 n 0.900427
TransPr 16
0 0 0.97688
0 1 0.000740519
0 2 0.0223791
0 3 7.3978e-07
1 0 0.161549
1 1 3.07654e-05
1 2 0.0615617
1 3 0.776858
2 0 0.00398397
2 1 0.250497
2 2 0.741535
2 3 0.00398397
3 0 0.0653865
3 1 2.56408e-06
3 2 0.000643583
3 3 0.933967
```

Figure 5: Sample set of probabilities

The Viterbi algorithm uses these probabilities to find the best sequence of states for the input string. Consider another example statement:

this example shows how the Acronym Finder Program AFP works.

Step 1: Initialization

n = 1, emission vocabulary = n

$\delta_0(0) = \pi_0 . b_o(n) = 0.7 * 0.870133 = \mathbf{0.6090931} \quad \psi_0(0) = 0$

$\delta_0(1) = \pi_1 . b_1(n) = 0.15 * 3.06805 * 10^{-5} = 0.4602075 * 10^{-5} \quad \psi_0(1) = 0$

$\delta_0(2) = \pi_2 . b_2(n) = 0.15 * 0.101324 = 0.0151986 \quad \psi_0(2) = 0$

$\delta_0(3) = \pi_3 . b_3(n) = 0 * 0.900427 = 0 \quad \psi_0(3) = 0$

Step 2: Recursion

n = 2, emission vocabulary = n

$\delta_1(0) = \max(\delta_0(0).a_{00}, \delta_0(1).a_{10}, \delta_0(2).a_{20}, \delta_0(3).a_{30}).b_0(n)$

$= \max(0.06497379 * 0.97688, 0.4602075 * 10^{-5} * 0.161549, 0.13379775$

$* 0.00398397, 0 * 0.653865)\ 0.870133$

$= \max(0.063471, 0.074340 * 10^{-5}, 0.000533, 0)\ 0.870133$

$= \mathbf{0.0552282}$

$\boldsymbol{\psi_1(0) = 0}$

$\delta_1(1) = \max(\delta_0(0).a_{01}, \delta_0(1).a_{11}, \delta_0(2).a_{21}, \delta_0(3).a_{31}).b_1(n)$

$= \max(0.06497379 * 0.000740, 0.4602075 * 10^{-5} * 3.07654 * 10^{-5}, 0.13379775$

$* 0.250497, 0 * 2.5640 * 10^{-6})\ 3.06805 * 10^{-5}$

$= \max(0.0000480, 1.4158467 * 10^{-10}, 0.0335108, 0)\ 3.06805 * 10^{-5}$

$= 0.1028 * 10^{-5}$

$\psi_1(1) = 2$

$\delta_1(2) = \max(\delta_0(0).a_{02}, \delta_0(1).a_{12}, \delta_0(2).a_{22}, \delta_0(3).a_{32}).b_2(n)$

$= \max(0.06497379 * 0.0223791, 0.4602075 * 10^{-5} * 0.0615617, 0.13379775$

$* 0.741535, 0 * 0.000643583)\ 0.101324$

$= \max(0.001454, 0.0283311 * 10^{-5}, 0.0992157, 0)\ 0.101324$

$= 0.01005$

$\psi_1(2) = 2$

$\delta_1(3) = \max(\delta_0(0).a_{03}, \delta_0(1).a_{13}, \delta_0(2).a_{23}, \delta_0(3).a_{33}).b_3(n)$

$= \max(0.06497379 * 7.3978 * 10^{-7}, 0.4602075 * 10^{-5} * 0.776898, 0.13379775$

$* 0.00398397, 0 * 0.933967)\ 0.900967$

$= \max(0.48066 * 10^{-7}, 0.357533 * 10^{-5}, 0.0005330, 0)\ 0.900967$

$= 0.00048045$

$\psi_1(3) = 2$

n=3, emission vocabulary = n

$\delta_2(0) = \max(\delta_1(0).a_{00}, \delta_1(1).a_{10}, \delta_1(2).a_{20}, \delta_1(3).a_{30}).b_0(n)$

$= \max(0.0552282 * 0.97688, 0.1028 * 10^{-5} * 0.161549, 0.01005$

$* 0.00398397, 0.00048045 * 0.653865)\ 0.870133$

$= \mathbf{0.046944}$

$\boldsymbol{\psi_2(0) = 0}$

$\delta_2(1) = \max(\delta_1(0).a_{01}, \delta_1(1).a_{11}, \delta_1(2).a_{21}, \delta_1(3).a_{31}).b_1(n)$

$= \max(0.0552282 * 0.000740, 0.1028 * 10^{-5} * 3.07654 * 10^{-5}, 0.01005$

$* 0.250497, 0.00048045 * 2.5640 * 10^{-6})\ 3.06805 * 10^{-5}$

$= 0.0077238 * 10^{-5}$

$\psi_2(1) = 2$

$\delta_2(2) = \max(\delta_1(0).a_{02}, \delta_1(1).a_{12}, \delta_1(2).a_{22}, \delta_1(3).a_{32}).b_2(n)$

$= \max(0.0552282 * 0.0223791, 0.1028 * 10^{-5} * 0.0615617, 0.01005$

$* 0.741535, 0.00048045 * 0.000643583)\ 0.101324$

$= 0.0007551$

$\psi_2(2) = 2$

$\delta_2(3) = \max(\delta_1(0).a_{03}, \delta_1(1).a_{13}, \delta_1(2).a_{23}, \delta_1(3).a_{33}).b_3(n)$

$$= \max(0.0552282 * 7.3978 * 10^{-7}, 0.1028 * 10^{-5} * 0.776898, 0.01005 * 0.00398397, 0.00048045 * 0.933967)\ 0.900967$$

$$= 0.0004042$$

$$\psi_2(3) = 3$$

The recursion step continues till n=10 for all emission symbols in the same manner as above.

From the values calculated this far, we can see that $\delta_0(0)$, $\delta_1(0)$ and $\delta_2(0)$ have the highest probabilities in their group. So backtracking would give us the sequence $\psi_2(0) = 0$, $\psi_1(0) = 0$ of states and the start state is 0. When we run the example through the decoding module of our program we get:

this 0

example 0

shows 0

how 0

the 0

Acronym 2

Finder 2

Program 2

AFP 1

works 3

CHAPTER 5

IMPLEMENTATION

The program that was written to discover acronyms with their definitions was written entirely in C++. The features that were implemented include a routine that strips the input file off any punctuation, the decoding algorithm called Viterbi algorithm that finds the best sequence of states for the input file, the algorithm that learns the parameters of the HMM (Maximum Likelihood Estimation), the routine that ascertains the type of the input word and also the function that estimates the smoothing constant for the purpose of absolute discounting. In this chapter we explain the various modules of our program in detail.

The program consists of three C++ files; one is the main file that analyzes the command line arguments and determines the action to be performed, the second file contains all method and variable declarations and the other consists of the definitions of the same. The program consists of two modules namely,

- Learning Module
- Decoding Module

Before we explore each of these modules in greater detail we talk about the aspects that are common to both. To run the program certain command line arguments need to be specified. They are:

- The first argument is a symbol that signifies the module to be invoked.
- The second is the number of states that are in the HMM.
- The third is the file that contains the probability distributions associated with the HMM (determined while learning and used while testing).

- The fourth is the name of the test file or the tagged training file.
- The fifth argument specifies the name of the output file.

The number of states of the HMM are determined during our design phase. The documents that are used for training and testing/decoding require to be pre-processed by a routine that removes punctuation marks and transcribes white space characters into new line characters.

5.1 Learning Module

The main goal of this module is to use Maximum Likelihood estimation to determine the transition probabilities between states of the HMM and symbol emission probabilities associated with each state. The module is invoked by passing the right set of command line arguments.

Preparing the tagged training document file is the very first step. As has been mentioned, the documents are collected and pre-processed. The file is manually tagged with one of four states ensuring that the topology of transitions is not violated. This completed tagged file is uploaded into the directory where our code is placed.

The document is parsed one line at a time. Every line of the tagged document has two entries – the word and the state that it corresponds to. The word is translated into one of the emission symbols in the following manner:

- If the word starts with a capital letter and is followed by smaller case letters, it is translated to the symbol ‘D’
- If the word comprises of only capital letters, it is translated to the symbol ‘A’
- Any other word is translated to the symbol ‘l’

The symbols and the state are assigned to a character and an integer variable respectively. Counters are set up to keep an account of the number of times the combination of the symbol and the state are encountered and the number of times the transition from the previous state to the current state is seen in the training document. The counters are incremented by 1 every time. The routine also keeps track of counters that are used to normalize the probabilities. This runs till all the lines of the input training file are read.

A function is called after counting the number of transitions encountered and the number of times the symbols are emitted from states. This function calculates the actual probabilities by using the formulas we have discussed in Chapter 3.

These formulae are implemented as they have been discussed but the code would only be useful for a very short sequence of symbols. This is because many quantities would get extremely small as the sequence gets longer. This problem could be addressed in two ways:

- Normalization, and
- Working on the logarithm domain

Working with logarithm would mean conversion of the product of small quantities into a sum of the same small probabilities. The logarithm domain is not the best alternative for counting, normalization is an easier method to solve the underflow problem for Maximum Likelihood estimate. A smoothing constant is calculated depending on the number of states that are in our HMM. The smoothing constant is one-thousandth of the number of states. The probabilities are calculated by adding this calculated constant to the counter and dividing this sum by a sum of the product of the number of states and

smoothing constant and the normalization counter for every state. The formulae are written below:

State transition probability, A[i][j] between state i and j is calculated as

$$A[i][j] = \frac{smoothconstant + CounterA[i][j]}{N * smoothconstant + NormA[i]}$$

Symbol emission probability of symbol 'j' from state 'i' is

$$B[j][i] = \frac{smoothconstant + CounterB[j][i]}{SYMNUM * smoothconstant + NormB[i]}$$

SYMNUM is a constant that is defined in the header file that is assigned to an integer number 94. The symbol 'j' in the above formula corresponds to the ASCII value of the character. The list of symbols we consider is shown in Table 1. Symbol emission probabilities are calculated for each of these symbols in our code although the symbols that are relevant to us are only just 3 as we have explained. The code is built to accommodate other HMM designs with a different number of states and different emission vocabulary sets.

The probabilities that are calculated are written to the file whose name is specified in one of the command line arguments. The initial probabilities were later added in the file manually after making assumptions about the most probable initial states. It was decided that these probabilities would be set by hand as most often the initial state is a prefix state in a document which would result in zero probabilities for the other states.

5.2 Decoding Module

The main objective of this module is to find the best possible sequence of states for the sequence of words belonging to documents that are isolated for the

Decimal	ASCII	Decimal	ASCII	Decimal	ASCII	Decimal	ASCII	Decimal	ASCII
33	!	53	5	73	I	93	]	113	Q
34	“	54	6	74	J	94	^	114	R
35	#	55	7	75	K	95	_	115	S
36	$	56	8	76	L	96	`	116	T
37	%	57	9	77	M	97	a	117	U
38	&	58	:	78	N	98	b	118	V
39	‘	59	;	79	O	99	c	119	W
40	(	60	<	80	P	100	d	120	X
41	)	61	=	81	Q	101	e	121	Y
42	*	62	>	82	R	102	f	122	Z
43	+	63	?	83	S	103	g	123	{
44	,	64	@	84	T	104	h	124	|
45	-	65	A	85	U	105	i	125	}
46	.	66	B	86	V	106	j	126	~
47	/	67	C	87	W	107	k		
48	0	68	D	88	X	108	l		
49	1	69	E	89	Y	109	m		
50	2	70	F	90	Z	110	n		
51	3	71	G	91	[	111	o		
52	4	72	H	92	\	112	p		

Table 1: Characters and their ASCII codes

purpose of testing. The decoding algorithm used is the Viterbi algorithm and the probabilities required by it, state transitions and symbol emission probabilities for every state, are calculated in the previous module. The inputs required are the name of the file in which the number of states and probabilities associated with the HMM are present along with the name of the file that needs to be tested against the model. The names of these files are passed as command line arguments when the module is invoked.

It must be ensured that the test file is preprocessed in the manner that has been described before it is used in further processing. The probabilities associated with the HMM model must first be saved into appropriate data structures. The number of states is read and assigned to an integer variable. The initial probability associated with each state is stored in a one-dimensional array, while the transition probabilities and symbol emission probabilities are placed in two-dimensional arrays.

Maximum Likelihood Estimation assigns a probability of zero to unseen emission – state combinations in the training file. This is potentially harmful to the decoding process and requires to be addressed. The problem is resolved by using a concept called absolute discounting. Absolute discounting involves subtracting a small amount of probability p from all symbols assigned a non-zero probability at a state s. Probability p is then distributed equally over symbols given zero probability by the Maximum Likelihood estimate. If v is the number of symbols that are assigned non-zero probability at a state s and N is the total number of symbols, emission probabilities are calculated by

$$P(w|s) = \begin{cases} P(w|s)_{ml} - p & if\ P(w|s)_{ml} > 0 \\ \dfrac{vp}{N - v} & otherwise \end{cases}$$

There is no best way to calculate the value of p. We calculate a value proportional to the non-zero emission probability MLE assigned to the state. The manner in which the concept is implemented in our code is explained below.

Once all the values are read from the probabilities file, we iterate through the arrays making note of the indices of the values that are assigned a zero probability. Similar to the calculation that is used in the learning module, a function is called and the smoothing constant is determined. The arrays are iterated through once again and for every entry we calculate the portion that is to be subtracted from itself. A function is called to calculate this portion and the parameters we pass to this function are the current probability associated with the state and the smoothing constant that was determined in the previous step. The function returns a value of type ‘double’ which is then subtracted from the current probability. The returned values are added to a variable to record the overall sum.. This sum is then evenly distributed among the entries that were observed to have zero entries.

Absolute discounting is also used to smooth the transition probabilities as well as initial probabilities to ensure that zero entries have no negative effects on the code. The process is the same as the one described above. The program is built to handle large amounts of data and to overcome the problem of underflow (explained in the learning module). The decoding algorithm is done entirely in the logarithm domain as opposed to the use of normalization.

The working of the Viterbi algorithm is already discussed in chapter 3. The implementation of the algorithm is done in much the same way. Two composite data types: struct data types are declared to handle the complex nature of the task involved.

One struct type is used to keep track of the path that has been taken to get to the current state. It has an integer member variable that stores the current state and another member of the same struct type that holds the path; that is, the best sequence of states to get to the state previous to s. Another struct type is used for the implementation of the trellis diagram (The significance trellis diagram has been explained in Chapter 3). This composite data type keeps track of the best path, the probability of the best path and the probability once the path is extended to include the current state.

As we are working on the logarithmic domain, the product of two values is converted to the sum of the logarithm of the same values. Two arrays of objects are created for each of the struct types. One keeps track of the current best path till the current state and the other records trellis information for every state. The input file is read one line at a time. Once a word is read the first task is to translate these words into one of our emission symbols. The default character is the symbol set aside for a normal word; this is used when the input word does not fall into any one of our defined categories.

The very first step in the algorithm is to determine the best possible start state. The sum of the logarithms of the initial probability for every state and the emission probability of the symbol associated with the same state results in a probability. This value is assigned to the member variable of the trellis data type that records the probability of the path. In this manner start probabilities of every state are recorded.

$trellis[i].pr = log10(I[i]) + log10(B[cIndex][i])$ where I[i] is the probability that i is the start state and B[cIndex][i] is the emission probability of cIndex (ASCII value of symbol) associated with state i.

In the recursion phase of the Viterbi algorithm we build our trellis diagram to find the best way to get from one state to the next for the observed data. A temporary variable is used to store the best 'From' state; this is required to resolve contention. Nested 'for' loops are used to find the best sequence of states and the probability associated with every combination of the same to determine the most likely transition. The value is a sum of the logarithms of the probability calculated in the previous step, the transition probability between the previous state and the current state and the emission probability associated with the symbol (type of next word that is read) from the current state. The new probability and the best 'From' state are recorded in the object that is created to hold the best possible values for the states in question. All the words in the document are put through the same process with the use of a while loop.

$thisPr = trellis[j].pr + log10(A[j][i]) + log10(B[cIndex][i])$ where trellis[j].pr is the probability associated with the current path till state j, A[j][i] is the transition probability from state j to state i and B[cIndex][i] is the same as in the equation above.

The best path is extracted from the array of trellis objects created, by using an iterator object on it. The result of the decoding process is a file (if specified) that has the string of words and the state to which they belong listed adjacent to them.

The output file is then analyzed to determine how well the HMM model performed. The evaluation measures and findings of our experiment are explained in the next chapter.

CHAPTER 6

EXPERIMENTS

Following the phases of the software development life cycle the problem description was provided in Chapter 1, the design was explained in Chapter 3 and the code explained in Chapter 4. This chapter talks about the experiments that were conducted to evaluate the performance of the HMM model.

To build an HMM model that generalizes well and has high accuracy it is important to gather large amounts of data for training. The better trained the model is, the better the model performs against new data sets. As there are no pattern matching algorithms and regular expression matching algorithms in place, it is not possible to work with context windows for possible acronym occurrences. There is therefore a requirement for large amounts of data across different domains for the Maximum Likelihood Estimate (MLE) to provide the set of probabilities that are associated with HMMs. Different domains ensure the use of acronyms with their definitions in various patterns. Every author has their own pattern of writing and when these authors are picked from different domains, their writing styles seldom coincide; this allows the HMM to train on different types of data sets.

To achieve what has been explained, a set of 200 documents were randomly collected from various articles available on the Internet. The only pre-requisite that was to be satisfied by every file was the occurrence of at least one acronym with the definition in its vicinity. The collection of documents was divided into two categories. One set of 100 documents were used to train the HMM model and the other was used to test the same model so as to evaluate the performance of the HMM model that was designed. In this

study we discovered that one of the easiest ways to collect data was with the help of the glossary of acronyms defined for a specific area of study. For example, from the Glossary of Education Terms and Acronyms we gathered acronyms and assimilated data as shown below.

Allan Odden is Professor of Educational Leadership and Policy Analysis at the University of Wisconsin-Madison. He also is Co-Director of the Consortium for Policy Research in Education (CPRE), a national center studying how to improve state and local education policy. He formerly was professor of education policy and administration at the University of Southern California (1984-1993) and Director of Policy Analysis for California Education (PACE), and from 1975-1984 held various positions at the Education Commission of the States. He was president of the American Educational Finance Association in 1979-80 and received AEFA's distinguished Service Award in 1998. His research and policy emphases include school finance redesign and adequacy, effective resource allocation in schools, the costs of instructional improvement, and teacher compensation. Dr. Odden has written widely, publishing over 200 journal articles, book chapters, and research reports, and 30 books and monographs. His newest book, co-authored with Marc Wallace, is entitled How to Create World Class Teacher Compensation (St. Paul: Freeload Press).

Figure 6: Sample Data 1

The National Association of Colleges and Employers (NACE) tracks entry salary offers in a variety of occupations within each of the fifty states, for graduates of select universities. In a 2003 school finance adequacy study, Arkansas reviewed starting salaries for the following occupations and compared them to the typical teacher—agriculture and natural resources, health and related occupations, and humanities and social science, and the following occupations for math and science teachers—business, computer technology, engineering, and science. The final issue in making salary comparisons is whether teacher salaries should be "adjusted" to account for the fact that the typical teacher "works" only 9 or 10 months of the year, or even just 5-6 hours a day. This is a hotly debated issue within the education policy community, with many arguing for an adjustment and others arguing just as vociferously for no adjustment. But as Allegretto et al. (2004) learned from the U.S. Bureau of Labor Statistics (BLS), such an adjustment is not warranted because it is difficult to determine how many hours or even weeks that teachers work. Teachers prepare lessons and correct papers outside of the regular school day and often engage in training or curriculum development over the summer months. In comparing salaries among professions, the BLS makes no adjustments when "work" hours are difficult to determine, such as the number of hours airplane pilots work, or college professors work, and suggest that salary comparisons for such jobs, including teachers, be made on an annual salary basis.

Figure 7: Sample Data 2

Before either set of documents were used further, each of the set of documents were put through a pre-process phase. Pre-processing of the training documents and the set of

testing documents were conducted separately although the process involved was essentially the same. The decision to make the tradeoff between saving time and accuracy makes the results of the experiments credible. A routine was written to do two tasks; strip the document collections of any punctuation and special characters, white space characters were replaced with new line characters in order to allow easy tagging and easy scanning of the result set. The output of the pre-processing step is a file that has no punctuations, no special characters and has only one word per line.

The file obtained after pre-processing the 100 documents set aside for training are tagged in a simple text editor such as notepad or wordpad. Tagging involves the assignment of one of the 4 states to every word in the file which is just writing the state adjacent to the word. The decision of the assignment is made by the trainer who is well aware of the topology of transitions that the HMM model allows. The tagged file is saved as a simple text file with a .txt extension and saved in ASCII encoding. Transition and Emission probabilities are calculated by the use of MLE on this file. Initial probabilities are set by hand so that all states have a fair chance of being the start state. If MLE were to decide these initial probabilities, only the prefix state would be assigned a large probability while the others would have a probability closer to zero as documents start at the prefix state most often. The output of the learning phase is a file with probabilities associated with the HMM model.

The 100 documents set aside for testing use the probabilities calculated in the previous step. The output of the decoding phase is a file similar in appearance to the training file. This file is analyzed by a human observer (not necessarily aware of the design) for words

that are tagged with state 1 and 2 as they are candidate acronyms and definitions respectively. The file is checked for True Positives, False Positives and False Negatives.

Standard measures of Precision, Recall and F1 measure are used to evaluate the performance of the HMM.

Precision is defined as

$$Precision = \frac{Number\ of\ True\ Positives}{Number\ of\ True\ Positives + False\ Positives}$$

Recall is defined as

$$Recall = \frac{Number\ of\ True\ Positives}{Number\ of\ True\ Positives + False\ Negatives}$$

F1 measure is

$$F1 = \frac{1}{\frac{1}{Recall} + \frac{1}{Precision}}$$

The results obtained as shown in table below.

True Positive	False Positive	False Negative	Precision	Recall	F1
196	16	4	0.9245	0.98	0.95144

Table 2: Results

CHAPTER 7

CONCLUSION AND FUTURE WORK

The main objective of this thesis is to elucidate that HMMs can be used for the task of Information Extraction. Here, we addressed the problem of finding acronyms and their definitions using HMMs. We designed an HMM, implemented Viterbi algorithm and Maximum Likelihood Estimator in C++ and compared our findings to the Acronym Finder Program [3]. The experiment can be concluded as successful and hence it establishes that HMMs can be used for the task of extracting relevant information from documents.

The experiments in this thesis were conducted on a small set of 200 documents. To build an HMM that generalizes well and has high accuracy a large amount of data is required. Testing the model on a large collection of data and comparing results against the ad-hoc algorithm can be performed to establish which of the two methods performs better.

BIBLIOGRAPHY

1. Dayne Freitag and Andrew Kachites McCallum, “Information extraction with HMMs and Shrinkage,” In Proceedings AAAI-99 Workshop Machine Learning and Information Extraction, 1999.
2. Kazem Taghva, Jeffrey Coombs, Ray Pereda and Thomas Nartker, “Address Extraction Using Hidden Markov Models”, Proc. IS&T/SPIE 2004 Intl. Symp. on Electronic Imaging Science and Technology
3. Kazem Taghva and Jeff Gilbreth, “Recognizing Acronyms and their definitions”, International Journal on Document Analysis and Recognition, Volume 1, Number 4, 191-198, DOI: 10.1007/s100320050018
4. Wayne Grixti, Charlie Abela and Matthew Montebello, “Name Finding From Free Text Using HMMs”
5. ChengXiang Zhai, “A Brief Note on the Hidden Markov Models”, University of Illinois at Urbana-Champaign, March 16, 2003
6. Lawrence R Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”, Proceedings of the IEEE, Vol 77, No. 2, pp. 257-286, February 1989
7. Barbara Resch, “Hidden Markov Models”
8. Kazem Taghva, Russell Beckley and Jeffrey Coombs, “The Effects of OCR Error on the Extraction of Private Information,” Document Analysis Systems 2006: 348-357
9. Daniel M Bikel, Scott Miller, Richard Schwarts and Ralph Weishedel, “Nymble: A High-Performance Learning Name Finder,” Proceedings of the Fifth Conference on Applied Natural Language Processing, 1997, pp. 194-201

10. Daniel M Bikel, “An Algorithm that Learns What’s in an Name”, Machine Learning, BBN Systems and Technologies, Cambridge 1999
11. M. Banko, M. Cafarella, S. Soderland, M. Broadhead and O. Etzioni, “Open Information Extraction from the Web,” Magazine Communications of the ACM - Surviving the data deluge Volume 51 Issue 12, December 2008
12. Harry Zhang, “The Optimality of Naïve Bayes”, American Association for Artificial Intelligence 2004
13. Charles Sutton and Andrew McCallum. “An introduction to conditional random fields for relational learning”. In Lise Getoor and Ben Taskar, editors, Introduction to Statistical Relational Learning. MIT Press, 2006.
14. ReliaSoft Corporation, “ MLE (Maximum Likelihood) Parameter Estimation”, Accelerated Life Testing Reference Appendix B : Parameter Estimation
15. Christopher D Manning, Prabhakar Raghavan and Hinrich Schutze, “Introduction to Information Retrieval”, Cambridge University Press. 2008.
16. Gerald DeJong, “ Skimming Newspaper Stories By Computer”, Yale University 1977
17. Georgette Silva and Don Dwiggins, “Towards a Prolog Text Grammar”, ACM SIGART Bulletin 73 October 1980, Page 20-25.

VITA

Graduate College
University of Nevada, Las Vegas

Lakshmi Vyas

Degrees:
Bachelor of Engineering in Computer Science, 2006
Visvesvaraya Technological University

Thesis Title: Finding Acronyms and Their Definitions using HMM

Thesis Examination Committee:
Chair Person, Dr. Kazem Taghva, Ph.D.
Committee Member, Dr. Ajoy K. Datta, Ph.D.
Committee Member, Dr. Laxmi P. Gewali, Ph.D
Graduate College Representative, Dr. Venkatesan Muthukumar, Ph.D.