UNLV Theses, Dissertations, Professional Papers, and Capstones

8-1-2012

Post Processing of Optically Recognized Text via Second Order Hidden Markov Model

Srijana Poudel
University of Nevada, Las Vegas, srizzana@gmail.com

POST PROCESSING OF OPTICALLY RECOGNIZED TEXT VIA SECOND ORDER HIDDEN MARKOV MODEL

by

Srijana Poudel

Bachelor in Computer Engineering
Institue of Engineering, Pulchowk Campus, Tribhuvan University, Kathmandu
2007

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

School of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2012

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Srijana Poudel

entitled

Post Processing of Optically Recognized Text Via Second Order Hidden Markov Model

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 College Representative

Thomas Piechota, Ph. D., Interim Vice President for Research and Graduate Studies
and Dean of the Graduate College

December 2012

ABSTRACT

by

Srijana Poudel

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

In this thesis, we describe a postprocessing system on Optical Character Recognition(OCR) generated text. Second Order Hidden Markov Model (HMM) approach is used to detect and correct the OCR related errors. The reason for choosing the 2nd order HMM is to keep track of the bigrams so that the model can represent the system more accurately. Based on experiments with training data of 159, 733 characters and testing of 5688 characters, the model was able to correct 43.38 % of the errors with a precision of 75.34 %. However, the precision value indicates that the model introduced some new errors, decreasing the correction percentage to 26.4 %.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Dr. Kazem Taghva, for helping me focus on the right problems, providing me with ideas and helpful insights, and guiding me through this work. My sincere appreciation and gratitude goes to the members of my thesis advisory committee, Dr. Ajoy Datta, Dr. Laxmi P Gewali and Dr. Venkatesan Muthukumar for believing in me and accepting to be a part of my committee.

I would like to acknowledge the Department of Computer Science, for providing me with a Graduate Assistantship for my master's degree. I would also like to express my appreciation to my husband, Roshan Gyawali, my family and my friends for their encouragement and support during my preparation of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Research in Hidden Markov Models (HMM) and its application to speech recognition, pattern recognition, string matching, and others has a long, robust history in computer science. HMMs are stochastic models which were introduced and studied in the late 1960s and early 1970s [1, 2, 3]. HMMs have been widely used in speech recognition [4, 5, 6]. More recently they have also been applied to handwriting recognition [7, 8]. Everything from text retrieval to speech recognition relies on efficient and reliable text correction [9]. Different approaches have been successfully used by many researchers in the area of correcting errors in text [9, 10]. In this thesis, we have discussed a OCR post processing system based on HMM approach. In HMM approach, the problem is, given the observation sequence, how to calculate the model parameters; and then given the model parameters and observation sequence, how to find the optimal state sequence. Many approaches have been used by researches for correcting the various types of errors in Optical Character Recognition (OCR) text [9, 11]. We contribute to OCR error correcting research by studying the HMM approach. We research whether HMM approach will be able to correct these OCR errors and if so, what types of errors.

1.1 Motivation

The trend to digitize paper based documents such as books and newspapers has emerged in the last years. The objective is to preserve these documents and make them fully accessible in digital form. For todays digital world knowledge contained in paper based documents is more valuablew when it is available in digital form. The first step towards digitizing a paper based archive is to scan the documents. The next step is to apply an OCR process, meaning translate that the scanned image of each document into machine processable text. Due to the print quality of the documents and the error-prone pattern matching techniques of the OCR process, OCR errors occur. On historic documents this error rate will be even higher because the print quality is likely to be lower. After finishing the OCR process several post-processing steps are necessary depending on the application for correcting OCR errors and spelling mistakes. Data which contains spelling mistakes or OCR errors is difficult to process. There has been much effort in the field of correcting spelling mistakes and OCR errors, some of which we is discussed in Section 1.2.

1.2 Previous Work

Studies have shown OCR errors significantly degrade the effectiveness in information extraction. In a study conducted by Taghva, Beckley and Coombs [12], they have concluded that steps should be taken to improve OCR texts, as OCR text can weaken the usefulness of information extraction process. A large amount of research have been directed to improving the OCR accuracy after the fact [9, 13, 14].

A semi-automatic OCR correction system has been proposed by Taghva and Stofsky [9], called OCRspell. OCRspell is especially designed as a semi-automatic approach. A learning mechanism is used based on the corrections applied by the user. By comparing error-prone token and the manual replacement by the user, dynamic mappings are derived, e.g. *iiiount@in* → *mountain* the mappings *iii* → *m* and @ → *a* are derived.

Hauser and Schulz have proposed an unsupervised training algorithm. This algorithm uses a dictionary and the corpus itself to obtain training samples. For each corpus word, the dictionary retrieves similar words based on the Levenshtein-Distance. More techniques and details about correcting spelling mistakes can be found in Kukich [15].

1.3 Thesis Overview

Thede and Harper defined [16] ,HMM is a statistical construct that can be used to solve classification problems that have an inherent state sequence representation . Given an OCR text to be corrected and other model parameters, the output of the system is the corrected text, i.e., sequence of most likely states. The system uses Maximum Likelihood Estimates (MLE) to calculate the parameters of HMM. The Viterbi Algorithm is then used to correct the given OCR text. The fundamental problems of this system are to implement MLE for training a HMM, to implement a second order Viterbi Algorithm to find the most likely sequence of states, and to run our model for an OCR text to calculate the performance measures. There are two practical issues that are associated with the implementation of HMM. We will address those issues and solve them using numerical scaling and smoothing techniques. We have implemented the second order HMM and carried out a comparative study between the first order and the second order HMM .

1.4 Thesis Structure

This thesis is organized into different chapters starting from introduction in chapter 1. In chapter 2, we provide a standard definition of HMM and the formulation and algorithms used. We also discuss the classification of OCR errors and previous attempts to correct the different types of errors existing in an OCR text. Chapter 3 presents detailed design of our HMM model and the extended viterbi algorithm for a second order model. In chapter 4, we describe the implementation of our system. We discuss problems encountered during the implementation and explain approaches used to solve them. In chapter 5, we describe the outcome of experiments measuring the effectiveness of our HMM system for various OCR documents. Chapter 6 offers our conclusions and prospects for future experimentation.

CHAPTER 2

BACKGROUND

This chapter provide a formal definition of HMM and its parameter. These defintions are very much akin to those introduced by Rabiner and Juang [3]. We list the problems related to an HMM and describe the algorithms and formulations used to solve each of those problems.

2.1 Markov Chain

A Markov chain, sometimes also referred to as an observed Markov Model, can be seen as a weighted finite-state automaton, which is defined by a set of states and a set of transitions between the states based on the observed input. A markov model is a stochastic process. Consider a system that has N distinct states $S1, S2, S3, \ldots, Sn$. 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 known as a Markov model[17]. For a more formal description, the Markov chain is specified by,

- $S = \{s_{\mathrm{1}}, s_{\mathrm{2}}, \ldots, s_{\mathrm{n}}\}$ is a finite set set of N states
- $\pi = \{\pi_{\mathrm{i}}\}$, the initial state probabilities. π_{i} is the probability that the model will start in state i, with $\sum_{i=1}^{N} \pi_{\mathrm{i}} = 1$
- $A = \{a_{\mathrm{ij}}\}$,the state transition probabilities. a_{ij} is the probability that the process will move from state i to state j in one transition, with $\sum_{j=1}^{N} a_{\mathrm{ij}} = 1, , \forall i$

Figure 2.1 is a simple markov model with three states s_1, s_2 and s_3. Probabilities in the arcs represents the probability of going from state s_{i} to s_{j}.

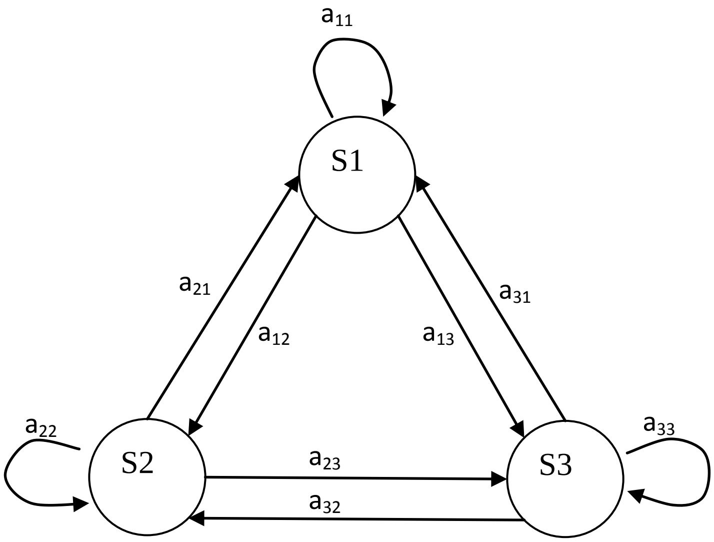


Figure 2.1: Illustration of a Markov model with 3 states

The Markov Model makes the assumption that the process was produced by a Markov source, a type of source in which symbols currently produced are dependent only on a fixed number of symbols that have been produced preceeding the current output. The order of the model specifies the number of preceeding outputs which are taken into account for the next symbol to be produced. Since the complexity of the model grows exponentially with the order and the added benefit of increasing the order of the model decreases as the order grows higher first order or second order Markov models are assumed to be sufficient for most of the applications. In a first order Markov model, the probability of a state only depends on the previous state, that is:

$$P(q_{\mathrm{t}} \mid q_{\mathrm{t\text{-}1}}, \ldots q_1) = P(q_{\mathrm{t}} \mid q_{\mathrm{t\text{-}1}})$$

For a second order Markov model the probability of being in state S_{i} at a time t depends on the state S_{j} at time $t-1$ and $t-2$. 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 are also 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.

2.2 Extending the Markov model to a Hidden Markov Model

The Hidden Markov Model is a doubly stochastic variant of the Markov Model with an underlying stochastic process that is not observable hidden but can only be observed through another set of stochastic processes that produce the sequence of observed symbols. In a regular Markov model the state transition probabilities are the only parameters. In a HMM, each state has a set of observation symbols. HMM can be used to solve classification problems that have an inherent state sequence representation. The model can be visualized as an interlocking set of states. These states are connected by a set of transition probabilities, which indicates the probability of travelling between two given states. A process begins, in some states, then at discrete time intervals, the process ”moves” to a new state as dictated by the transition probabilities. As the process enters each state, one of a set of output symbols is emitted by the process. Exactly which symbol is emitted is determined by a probablility distribution (emission probabilities) that is specific to each state. The output of the HMM is a sequence of output symbols. In general, there are many possible state sequences which generate an observation sequence. Hence, the state sequence is hidden [16].

Consider a system with three states s_1, s_2 and s_3. Let 0 and 1 be the observation symbols at each state. The HMM model for this system is shown in Figure 2.2. Each circle represents a state, value in the arcs represents the probability of going from state s_{i} to s_{j}. b_{ij} represents the emission probabilities.

2.2.1 Formal Definition and Parameters of HMM

As we discussed in earlier section, each state in the model has a number of parameters associated with it. In 1989 [5], Rabiner defined HMM as a 5-tuple (S, V, π, A, B) where

- $S = \{s_1, s_2, \ldots, s_{\mathrm{n}}\}$ is a finite set set of N states

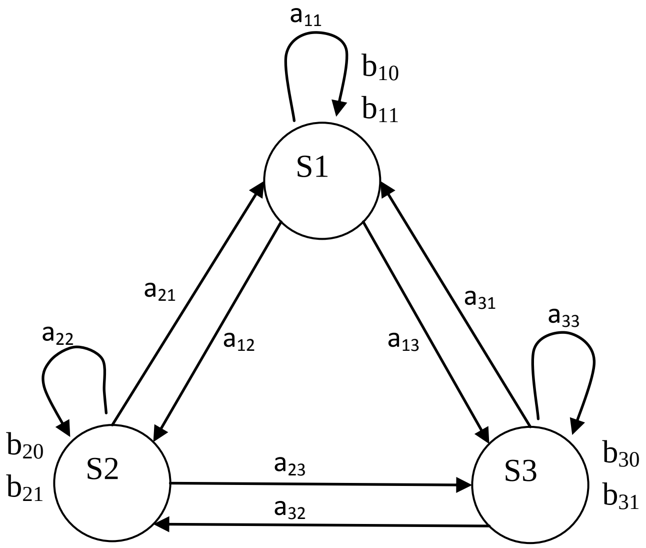


Figure 2.2: Illustration of a Hidden Markov model with 3 states

- $V = \{v_\mathrm{1}, v_\mathrm{2}, \ldots, v_\mathrm{m}\}$ is a set of M possible symbols in a vocabulary. The M observation/emission symbols correspond to the output of the system being modeled.

- $\pi = \{\pi_\mathrm{i}\}$, the initial state probabilities. π_i is the probability that the model will start in state i.

- $A = \{a_\mathrm{ij}\}$,the state transition probabilities. a_ij is the probability that the process will move from state i to state j in one transition.

- $B = \{b_\mathrm{i}(v_\mathrm{k})\}$, the output or emission probabilities. $b_\mathrm{i}(v_\mathrm{k})$ is the probability of generating symbol v_k at state i.

Usually compact notation $\lambda = \{A, B, \pi\}$ to denote the complete parameter set of the HMM. The constraints on the HMM are

$$\sum_{i=1}^{N} \pi_\mathrm{i} = 1 \text{ for } 1 \leq i \leq N$$

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

$$\sum_{k=1}^{M} b_{\mathrm{i}}(v_{\mathrm{k}}) = 1 \text{ for } i = 1, 2, \ldots, N$$

2.2.2 Three Fundamental Problems

Rabiner [3] states in his paper that HMM should be characterized by three fundamental problems.

- Evaluation: Given an observation sequence $O(O = o_1, o_2, \ldots, o_{\mathrm{T}})$ and a model λ, how do we compute the probability that the observation sequence was produced by the model. The problem of evaluation is solved using the Forward and Backward iterative algorithms [5].

- Decoding:.Given a model and a sequence of observations, the problem is the selection of an optimal sequence of states traversed to create this observation. Many possible state sequence could produce a given observation sequence. For each observation of a sequence, we have to choose those states that have the highest individual probability of producing that observation in order to find the optimal state sequence. A well known efficient method is a dynamic programming algorithm known as the Viterbi Algorithm. The algorithm obtains a solution recursively which is illustrated in Section 2.2.3.

- Training: The problem is to adjust all the parameters of the model, i.e, λ to maximize the probability of generating an observed sequence. We need to define the characteristics of our model based on previous observation sequences or training examples. The most common method of estimating a model parameters is the Maximum Likelihood Estimation (MLE) which is explained in Section 2.2.4.1. Baum Welch algorithm is also a common iterative restimation method for training a model [18].

For our thesis, we are only concerned with the decoding and training problem. We have used the MLE for solving the training problem and the Viterbi Algorithm for decoding purposes. We now proceed with the formulation for each of these methods in the following section.

2.2.3 Viterbi Algorithm

The most common solution to the decoding problem is the Viterbi Algorithm (VA). The Viterbi algorithm is used closely with Hidden Markov Models (HMMs). It is a dynamic programming algorithm that computes the most likely state transition path given an observed sequence of symbols. The idea of the VA is to find the most probable path for each intermediate state and finally for the terminating state. At each time only the most likely path leading to each state survives. To find the best state sequence $Q = \{q_1, q_2, \ldots, q_{\text{T}}\}$ for a given observation sequence $O = \{o_1, o_2, \ldots, o_{\text{T}}\}$, one need to define the quantity

$$\delta_{\text{t}}(i) = \max_{q_1, q_2, \ldots, q_{\text{t-1}}} P[q_1, q_2, \ldots, q_{\text{t}} = s_{\text{i}}, o_1, o_2, \ldots, o_{\text{t}} \mid \lambda]$$

which means the highest probability along a single path, at time t, which accounts for the first t observations and ends in state s_{i}. By induction one have:

$\delta_{\text{t+1}}(j) = [\max_i \delta_{\text{t}}(i) a_{\text{ij}}] b_{\text{j}}(o_{\text{t+1}})$ To be able to retrieve the state sequence, one need to keep track of the argument which maximized the above equation for each t and j. This is done via the variable $\psi_{\text{t}}(j)$ [5]. Assuming the observation sequence is T unit length as $o_1, o_2, \ldots,_{\text{T}}$ and an HMM with N states, the complete procedure for finding the best state sequence can be stated as follows.

Algorithm 2.1 Viterbi Algorithm

INPUT: HMM Model File , Observation file

OUTPUT: Most likely sequence of states for the test file

Step 1: Initialization

Define

$$\delta_1(i) = \pi_{\mathrm{i}} b_{\mathrm{i}}(O_1) \ ; \ 1 \leq i \leq N$$
$$\psi(i) = 0 \ ; \ 1 \leq i \leq N$$

Here $\delta_1(i)$ is the probability that symbol O_1 occurs at time $n = 1$ and at state i. $\psi(i)$ stores the optimal states.

Step 2: Recursive Computation

For $2 \leq n \leq T$ and $1 \leq j \leq N$

$$\delta_{\mathrm{n}}(j) = \max_{1 \leq i \leq N}(\delta_{\mathrm{n\text{-}1}}(i) a_{\mathrm{ij}}) b_{\mathrm{j}}(O_{\mathrm{n}})$$

The right hand side of the above equation is the maximum value for the probability that the partial sequence $O_1, O_2, \ldots, O_{\mathrm{n}}$ occurs at state j at time n where $n \leq T$; and j could be any state.

$$\psi_{\mathrm{n}}(j) = arg\max_{1 \leq i \leq N}(\delta_{\mathrm{n\text{-}1}}(i) a_{\mathrm{ij}})$$

$\psi_{\mathrm{n}}(j)$ stores the value of state i at $n - 1$ that makes the probability $\delta_{\mathrm{n}}(j)$ the highest,i.e. the most probable state is i. For example,at $n = 2$ and $j = 1$, if $\delta_1(3) a_{31}$ is the highest, then $\psi_2(1) = 3$

Step 3: Terminal States

$$p^* = \max_{1 \leq i \leq N}[\delta_{\mathrm{T}}(i)]$$
$$q^*_{\mathrm{T}} = arg\max_{1 \leq i \leq N}[\delta_{\mathrm{T}}(i)]$$

At the final time unit $n = T$, there are N probabilities $\delta_{\mathrm{T}}(i)$; $i = 1, \ldots, N$. The highest probabilities among these probabilities is the candidate for the optimal state sequence. ψ_{T} stores the corresponding terminal state. The final task, now, is to backtrack to the initial state following the value of ψ_{n}.

Step 4: BackTracking

For $n = T - 1, T - 2, \ldots, 1$ compute

$$q^*_{\mathrm{n}} = \psi_{\mathrm{n+1}}(q^*_{\mathrm{n+1}})$$

Set of $\{q^*_1, q^*_2, \ldots, q^*_{\mathrm{T}}\}$ is the optimal state sequence.

2.2.4 Training an HMM

Training or learning is the process to compute the parameters that best models a system, given a set of sequences that originated from this system. During the training process of an HMM, we compute the statistical parameters $\lambda = (A, B, \pi)$ of the HMM. The input to a training algorithm is a database of sample HMM behaviour, and the output is the transition, emission, and initial probability distribution of HMM. There are two standard approaches to the learning task namely supervised and unsupervised training. The choice is based on the data available for training process.If the training examples contain both the inputs and outputs of a process, we can perform supervised training. But if only the inputs are provided in the training data, we must use unsupervised training. Unsupervised training guesses a model that may have produced those observations. Maximum Likelihood Estimation (MLE) comes under supervised training, and Baum-Welch Algorithm comes under unsupervised training. For training our HMM, we have used the MLE approach.

2.2.4.1 Maximum Likelihood Estimation

Maximum-likelihood estimation is a method of estimating the parameters of a statistical model. When applied to a data set and given a statistical model, maximum-likelihood estimation provides estimates for the model's parameters. MLE methods are considered to be robust and versatile and so they are used for most models and for different types of data. 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. HMM parameters are estimated by MLE as follows:

$$\text{Transition Probabilities, } a_{\mathrm{ij}} = \frac{\text{Number of transitions from state } s_{\mathrm{i}} \text{ to } s_{\mathrm{j}}}{\text{Total number of transitions out of state } s_{\mathrm{i}}}$$

$$\text{Emission Probabilities, } b_{\mathrm{i}}(v_{\mathrm{k}}) = \frac{\text{Number of symbol } v_{\mathrm{k}} \text{ is emitted from state } i}{\text{Total number of symbols emitted out of state } i}$$

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.

2.3 Classification of OCR errors

Before errors can be corrected they have to be identified and classified. A proper classification is important in order to know which kind of errors occur. In related work [9], the classification is based on each steps of the OCR procedure. A brief explanation of the standard OCR procedure is described by Taghva [9] is explained in Appendix A. For our study, we use the categorization introduced by Esakov, Lopresti and Sandberg [19]. Our system doesn't work with insertion and deletion errors that results to division and concatenation of words. Some typical example for each type of the errors is shown in Table 2.1.

1. Deletion of a character
2. Insertion of a character
3. Substitution of one character for another (1:1 Substitution)
4. Substitution of two characters for one (1:2 Substitution)
5. Substitution of one character for two (2:1 Substitution)
6. Substitution of two character for two others (2:2 Substitution)

Type	Error Type	Example
1	Deletion of a character	deer → dee
2	Insertion of a character	cat → c at
3	1:1 Substitution	r → t; e → c; a → n
4	1:2 Substitution	n → ii; u → ii; m → ni
5	2:1 Substitution	ni → m; ii → u; tl → k
6	2:2 Substitution	rm → nn; rw → nr

Table 2.1: OCR Error Example

CHAPTER 3

HIDDEN MARKOV MODEL DESIGN: FIRST AND SECOND ORDER

In this chapter we describe the detail design of our HMM model. We have designed an ergodic model, in which all states can be reached from any state. The design consists of three major modules: the preprocessing, the trainer, and the decoder. In the following section, we will discuss the functionality of the three modules.

3.1 Preprocessing

The preprocessing step consists to two steps, filtering and creating a tagfile. All the characters are converted to lower cases. Then the file is parsed to converts all characters other than those having ASCII values from 97 to 122,i.e., from a to z to whitespaces. The next step is to create a tagfile, which is fed to the training function. Preprocessed OCR and non-OCR version of training data set is used to create a tagfile. Each of preprocessed files are read simultaneously, and each character of the files are written in a new file to make the tagfile. Given this tagfile, our HMM uses MLE to calculate emission and transition probabilities. HMM uses this tagfile to calculate emission and transition probabilities. The observation file which is given as input to the decoding module should also be filtered.

3.2 Trainer

The tagfile created by the preprocessing module is fed as input to this trainer module. The output of this module is the modelfile that includes all the model parameters for the designed model. For our system, we have used the Maximum Likelihood Estimation method for calculating the model parameters. Figure 3.1 shows an overview of this phase.

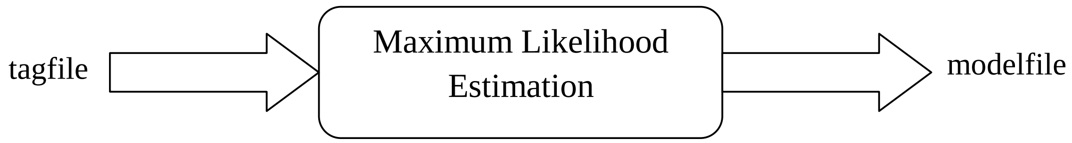


Figure 3.1: Training Module

3.2.1 Model Parameters for First Order Model

A first-order HMM makes two assumptions; first, the probability of a state is only dependent on the previous state:

$$P(s_{\mathrm{t}} \mid s_{\mathrm{t-1}}, \ldots, s_1) = P(s_{\mathrm{t}} \mid s_{\mathrm{t-1}})$$

Second, the probability of an output observation v_{t} is only dependent on the state that produced the observation, s_{t} and not on any other observations or states:

$$P(v_{\mathrm{t}} \mid s_{\mathrm{t}}, s_{\mathrm{t-1}}, \ldots, s_1, v_{\mathrm{t-1}}, \ldots, v_1) = P(v_{\mathrm{t}} \mid s_{\mathrm{t}})$$

3.2.1.1 States and Symbols

For English language, there are 26 letters in the alphabet. These 26 letters corresponds to 26 states of our first order HMM, i.e, $N = 26$. The next step is the identification of M optimal symbols $\{v_1, \ldots, v_{\mathrm{M}}\}$. The model states and observation symbols for the first order design are identified as follows.

States in the Hidden Markov Model: $S = \{a, b, c, \ldots, z\}$

Distinct symbols observed in each state: $V = \{a, b, c, \ldots, z\}$

Parameters $\lambda = \{A, B, \pi\}$ are computed using MLE.

Consider a model with States, $S = \{a, b, c, d, e\}$ and $V = \{a, b, c, d, e\}$.

Pictorially, the HMM can be modeled as in figure 3.2. Each circle in the figure represents a state. The connectors between two states are the transition probabilities. In each state, the symbol that can be observed are $a, b, c, d,$ and e . Probabilities of observing each symbol in states is given by emission probabilities.

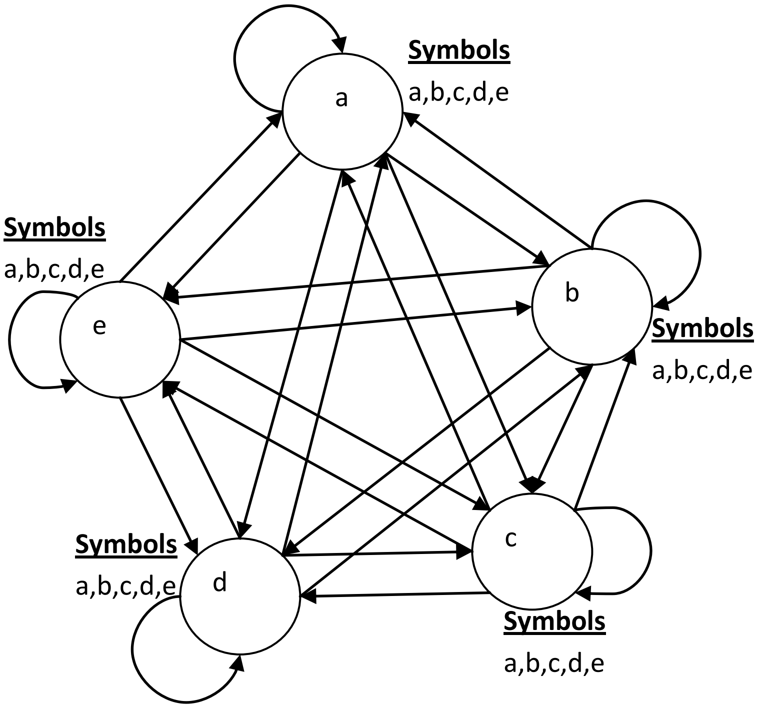


Figure 3.2: HMM Model

3.2.1.2 State Transition Probability:

For a first order model, the probability of being in state S_{i} at a time t depends on the state S_{j} at time $t-1$. For a model with 26 states, there are 26 × 26 number of first order transition probabilities. Transition matrix $A = a_{\mathrm{ij}}$, the probability of transition from state s_{i} to state s_{j},for a first order model is calculated using MLE as:

$$\text{Transition Probabilities, } a_{\mathrm{ij}} = \frac{\text{Number of transitions from state } \mathrm{s_i} \text{ to } \mathrm{s_j}}{\text{Total number of transitions out of state } s_{\mathrm{i}}}$$

3.2.1.3 State Emission Probability:

For our proposed model with 26 states and 26 symbols, there are 26 × 26 number of first order emission probabilities. Emission matrix $B = b_{\mathrm{i}}(v_{\mathrm{k}})$, the probability that symbol v_{k} is emitted at state s_{i}, for a first order model is calculated using MLE as:

$$\text{Emission Probabilities, } b_{\mathrm{i}}(v_{\mathrm{k}}) = \frac{\text{Number of symbol } v_{\mathrm{k}} \text{ is emitted from state } i}{\text{Total number of symbols emitted out of state } i}$$

3.2.1.4 Initial State Probability:

Initial state probability π_{i} is the probability that the model will start in state i. It is the probability of state S_{i} being the start state in an observation sequence. For our model with 26 states, there are 26 number of initial state probabilities.

3.2.2 Model Parameters for Second Order Model

A second-order HMM makes two assumptions; first, the probability of a state is only dependent on the two previous state:

$$P(s_{\mathrm{t}} \mid s_{\mathrm{t\text{-}1}}, \ldots, s_1) = P(s_{\mathrm{t}} \mid s_{\mathrm{t\text{-}1}}, s_{\mathrm{t\text{-}2}})$$

Second, the probability of an output observation v_{t} is only dependent on the state that produced the observation and its previous state

$$P(v_{\mathrm{t}} \mid s_{\mathrm{t}}, s_{\mathrm{t\text{-}1}}, \ldots, s_1, v_{\mathrm{t\text{-}1}}, \ldots, v_1) = P(v_{\mathrm{t}} \mid s_{\mathrm{t}}, s_{\mathrm{t\text{-}1}})$$

3.2.2.1 States and Symbol

The second order HMM is modelled with 27 states. In each states, 27 symbols can be observed. The model states and observations symbols for the second order design are identified as follows.

States in the Hidden Markov Model: $S = \{a, b, c, \ldots, z, \mathit{whitespace}\}$

Distinct symbols observed in each state: $V = \{a, b, c, \ldots, z, \mathit{whitespace}\}$

Parameters $\lambda = \{A, B, \pi\}$ are computed using MLE.

The reason for adding *whitespace* as a state and symbol is explained in Section 4.3.2.

3.2.2.2 State Transition Probability:

For a second order HMM, probability of transitioning to a new state depends not only on the current state, but also on the previous state. The transition matrix $A = \{a_{\mathrm{ijk}}\}$, the probability of transition from state $\mathrm{s_j}$ to state $\mathrm{s_k}$ given the state before $\mathrm{s_j}$ is state $\mathrm{s_i}$, is defined as follows.

$$a_{\mathrm{ijk}} = P(s_{\mathrm{t}} = s_{\mathrm{k}} \mid s_{\mathrm{t\text{-}1}} = s_{\mathrm{j}}, s_{\mathrm{t\text{-}2}} = s_{\mathrm{i}})$$

$$a_{\mathrm{ijk}} = \frac{\text{Number of transitions from } s_{\mathrm{j}} \mid t-1 \text{ and } s_{\mathrm{i}} \mid t-2 \text{ to state } s_{\mathrm{k}} \mid t}{\text{Total number of transitions from } s_{\mathrm{j}} \mid t-1 \text{ and } s_{\mathrm{i}} \mid t-2}$$

with the constraint $\sum_{k=1}^{N} a_{\mathrm{ijk}} = 1 \; ; \; 1 \leq i \leq N \; , \; 1 \leq j \leq N$
where N is the number of states in the model, and s_{t} is the actual state at time t. For a model with N states, there are $N \times (N \times N)$ transition probabilities.

3.2.2.3 State Emission Probability:

Similar to the extension of the transition probabilities, the approximation for the emission probabilities can also be modified to include the second order information. The emission matrix $B = b_{\mathrm{ij}}(v_{\mathrm{k}})$, probability that symbol $\mathrm{v_k}$ is emitted at state $\mathrm{s_j}$ given the state before $\mathrm{s_j}$ is state $\mathrm{s_i}$, is defined as follows.

$$b_{\mathrm{ij}}(v_{\mathrm{k}}) = P(v_{\mathrm{t}} = v_{\mathrm{k}} \mid s_{\mathrm{t}} = s_{\mathrm{j}}, s_{\mathrm{t\text{-}1}} = s_{\mathrm{i}})$$

$$\mathrm{b_{ij}}(\mathrm{v_k}) = \frac{\text{Number of times symbol } v_{\mathrm{k}} \text{ is emitted at state } s_{\mathrm{j}} \text{ at } t \text{ when } s_{\mathrm{t\text{-}1}} = s_{\mathrm{i}}}{\text{Total number of symbols emitted by state } s_{\mathrm{j}} \text{ when } \mathrm{s_{t\text{-}1}{=}s_i}}$$

with the constraint $\sum_{k=1}^{N} b_{\mathrm{ij}}(v_{\mathrm{k}}) = 1 \; ; \; 1 \leq i \leq N \; , \; 1 \leq j \leq N$
where N is the number of states in the model. For a model with N states and M symbols, there are $N \times (N \times M)$ emission probabilities.

The extended viterbi algorithm for the second order model that we have used for our HMM design uses only the first order emission probabilities. Therefore, for our design the second order emission probabilities are same as those of the first order model.

3.2.2.4 Initial State Proabilities:

The initial probabilities of the second order model are same as those of the first order model.

3.3 Decoder

The most common solution to the decoding problem of an HMM is the Viterbi Algorithm. We have also used the same for our decoder module. The input to this module is the modelfile created by the trainer module and the observation file. The output of this module is the corrected file. Figure 3.3 shows an overview of this phase.

The Viterbi algorithm for a first order model is described in Section 2.2.3. In this following section we describe the extended viterbi algorithm for a second order HMM.

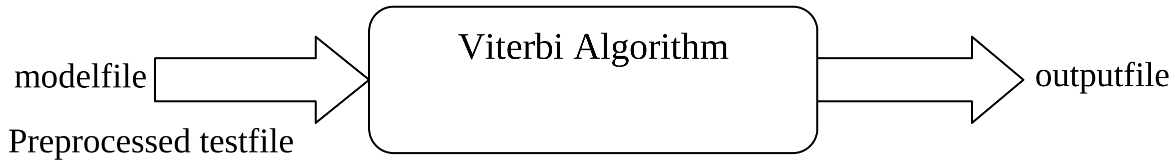


Figure 3.3: Decoding Module

3.3.1 Extended Viterbi Algorithm from Second Order HMM

In this section, we present the viterbi algorithm for second order HMM. This algorithm was proposed by Yang, Ke [20]. In 1988, Kundu and Bahl have used this algorithm for handwritten word recognition and have shown the advantage of second order model over the first order one [7].

Algorithm 3.1 Extended Viterbi Algorithm for Second Order Model

```
  INPUT: HMM Model File , Testfile
  OUTPUT: Most likely sequence of states for the test file
  Step 1: Initialization

  For $1 \leq l \leq N$,
    $\delta_1(l) = \pi_{\mathrm{l}} b(o_1 \mid l)$

  For $1 \leq l \leq N$,
    For $1 \leq m \leq N$
       $\delta_2(l, m) = \delta_1 a_{\mathrm{lm}} b(o_2 \mid m)$

  Step 2: Recursive Computation

  For $3 \leq i \leq T$,
     $For 1 \leq m \leq N,$
       $For 1 \leq n \leq N,$
         $\delta_{\mathrm{i}}(m, n) = \max_{1 \leq l \leq N}[\delta_{\mathrm{i\text{-}1}}(l, m) a_{\mathrm{lmn}}] b(o_{\mathrm{i}} \mid n)$
         $c_{\mathrm{i}}(m, n) = arg_{\mathrm{l}} \max_{1 \leq l \leq N}[\delta_{\mathrm{i\text{-}1}} a_{\mathrm{lmn}}]$

  Step 3: Determination of the last two states

     $s_{\mathrm{T}} = arg_{\mathrm{n}} \max_{1 \leq m \leq N, 1 \leq n \leq N}[\delta_{\mathrm{T}}(m, n)]$
     $s_{\mathrm{T\text{-}1}} = arg_{\mathrm{m}} \max_{1 \leq m \leq N, 1 \leq n \leq N}[\delta_{\mathrm{T}}(m, n)]$

  Here, the expression $arg_{\mathrm{m}} \max_{1 \leq m \leq N}[expression]$ denotes a function whose value
  is the value of $m$ that maximizes the value of the $expression$.

  Step 4: Backtracking to the first state

  For $T - 2 \geq i \geq 1$,
     $s_{\mathrm{i}} = c_{\mathrm{i+2}}(s_{\mathrm{i+1}}, s_{\mathrm{i+2}})$

  Now the optimal sequence is in $s_{\mathrm{i}}$.
```

CHAPTER 4

IMPLEMENTATION AND IMPLEMENTATION ISSUES

This chapter describes implementation of the HMM model described in Chapter 3. We have used java programming language for implementing the algorithms. The first step in implementing our model is the training of our HMM. Training of an HMM requires data. In this chapter, we describe data collection methods and detail implementation of the training and decoding phases. We also describe the issues related with implementing our system and approaches used to overcome those issues.

4.1 Data Collection

We need a set of data for training and testing purposes. Since we are testing our HMM with the OCR text, the first step was to find an OCR book, manually correct it, and preprocess it to fit with our system. We selected a book with 60 pages, which was manually corrected with reference to a non-OCR version image of the book. Two text files, the OCR and the correct version of the book, were created which is fed to the HMM for training purpose. The files were further divided into training sets and testing sets. The first 55 pages of the book were used for training. Remaining pages were used for testing. The list of OCR errors present in the training data in shown in Appendix B.

The dataset chosen for testing was further divided in 5 files. The count of the OCR errors present in the testing files are tabulated in Table 4.1.

Test File	Error Type	Count
File1	1:1 Substitution	8
(1164 characters)	1:2 Substitution	1
File2	1:1 Substitution	6
(1052 characters)	1:2 Substitution	1
File3	1:1 Substitution	7
(1002 characters)	1:2 Substitution	1
	2:1 Substitution	1
File4	1:1 Substitution	3
(1225 characters)	2:1 Substitution	2
File5	1:1 Substitution	2
(1245 characters)	2:2 Substitution	2
$\sum$		34

Table 4.1: Error Classification for Testing Data

4.2 Implementation of the Modules

In this section we descibe the implementation details of the preprocessing, training and decoding modules. Implementation involves defining number of classes that are necessary for each of these modules. We have to consider the data structures necessary for defining the parameters of each modules. We have used class diagram of UML [1], which is shown in Figure 4.1, to represent the major classes of our application

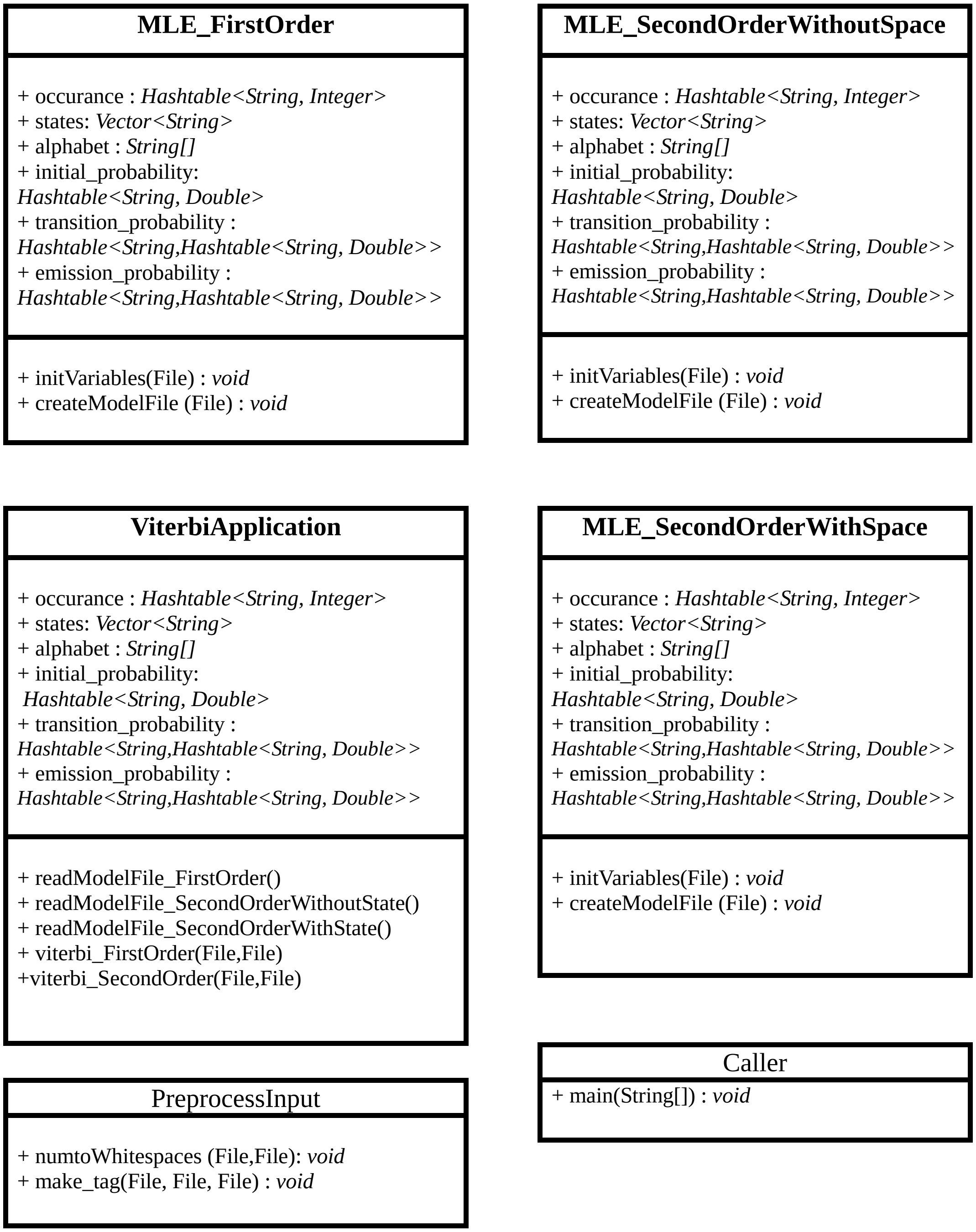

MLE_FirstOrder
+ occurance : Hashtable<String, Integer>
+ states: Vector<String>
+ alphabet : String[]
+ initial_probability:
Hashtable<String, Double>
+ transition_probability :
Hashtable<String,Hashtable<String, Double>>
+ emission_probability :
Hashtable<String,Hashtable<String, Double>>
+ initVariables(File) : void
+ createModelFile (File) : void
MLE_SecondOrderWithoutSpace
+ occurance : Hashtable<String, Integer>
+ states: Vector<String>
+ alphabet : String[]
+ initial_probability:
Hashtable<String, Double>
+ transition_probability :
Hashtable<String,Hashtable<String, Double>>
+ emission_probability :
Hashtable<String,Hashtable<String, Double>>
+ initVariables(File) : void
+ createModelFile (File) : void
ViterbiApplication
+ occurance : Hashtable<String, Integer>
+ states: Vector<String>
+ alphabet : String[]
+ initial_probability:
Hashtable<String, Double>
+ transition_probability :
Hashtable<String,Hashtable<String, Double>>
+ emission_probability :
Hashtable<String,Hashtable<String, Double>>
+ readModelFile_FirstOrder()
+ readModelFile_SecondOrderWithoutState()
+ readModelFile_SecondOrderWithState()
+ viterbi_FirstOrder(File,File)
+viterbi_SecondOrder(File,File)
MLE_SecondOrderWithSpace
+ occurance : Hashtable<String, Integer>
+ states: Vector<String>
+ alphabet : String[]
+ initial_probability:
Hashtable<String, Double>
+ transition_probability :
Hashtable<String,Hashtable<String, Double>>
+ emission_probability :
Hashtable<String,Hashtable<String, Double>>
+ initVariables(File) : void
+ createModelFile (File) : void
Caller
+ main(String[]) : void
PreprocessInput
+ numtoWhitespaces (File,File): void
+ make_tag(File, File, File) : void

Figure 4.1: Class Diagram of the HMM application

4.2.1 Preprocessing Module

This module consists to two functions, *numpunctuationstowhitespace* and *maketag*. The parameter to the first function is the file that needs to be filtered. This function reads each character of the file, and converts all the characters to lowercase. It filters all characters other than those having ASCII value between from 97 to 122, i.e., from a to z . The training sample files are fed as input to the *maketag* function. Both the files are first filterd to remove all the unwanted characters. A new text file called *tagfile.txt* is created. Then each character of the file is read one at a time and written in the *tagfile* ensuring that the topology of transitions and emission is not violated. A sample tagfile created by this module is shown in Figure 4.2.

```
n      n
o      o
t      t
e      e
s      s

o      o
n      n

w      w
i      i
t      l
c      c
h      h
c      c
r      r
a      a
f      f
t      t

b      b
y      y

g      g
e      b
o      o
r      r
g      g
e      e

l      l
y      y
m      m
a      a
n      n

k      k
i      i
t      t
t      t
r      r
e      e
d      d
g      g
e      e

w      w
e      e

a      a
r      r
e      e
```

Figure 4.2: Sample of a *tagfile* created by the preprocessing module

4.2.2 Training Module

The input to the training module is the *tagfile* prepared by the preprocessing module. This module consists of two functions, *initVariables* and *createModelfile*. We have used the *Hashtable* data structure to store the occurance and the emission, transition and initial probabilities. The states are defined as string *Vector*. All the symbols that can be emitted at each state are stored in a string *array*. *initVariables* function maintains 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 are generated from each states. This count is used to estimate emission and transition probabilities as described in Chapter 3. The *createModelfile* function creates a new text file *hmmmodel.txt* in which it writes the model probabilities calculated in earlier function. This module created a model file of size $19,224$ bytes for the first order model and $282,366$ bytes for the second order module.

4.2.3 Decoding Module

The decoding module is the implementation of the viterbi algorithm explained in Chapter 3. The input to the decoding module is the *modelfile* prepared by the training module and the *test(observation)file*. The output of this module is the *hmm-correctedfile*, which contains the most likely sequence of states. 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 next chapter.

4.3 Implementation Issues

Although the theory of HMM is well established, the implementation involves a large number of assumptions and restrictive conditions. Therefore, the results are very much dependent on implementation issues. In this section, we deal with issues related to implementing an HMM and our approach for solving them.

4.3.1 Floating Point Underflow

When implementing a HMM, long observation sequences often result in the computation of extremely small probabilities. These values are usually smaller in magnitude than the smallest value of a floating point number in a system. This results in a significant problem called floating point underflow. When Viterbi algorithms are applied to long sequences, it results in extremely small probability values that could underflow on most machines. We solve this problem by using scaling. A simple solution is to *log* all the probability values and then add those values instead of multiplying them. One approach would be to store the logarithmic values of the probabilities in the model file. In our implementation, the model file do not have the logarithmic values. We have used scaling during the viterbi implementation.

4.3.2 Choice of Model

There is no theoretically correct way of making choice of the type of model, model states and observation symbols. These are made based on trial and error depending on the process being modeled. In our second-order model, we initially designed the HMM with 26 states. When running the model with test files, we observed that more errors were introduced in stop words and the first character of a word. All the state transitions for stop words and first character of a word is ignored if whitespace is not considered as a state. This resulted in wrong modeling of the model parameters and thus more errors were introduced. To minimize such errors, the model was redesigned with whitespace added as a state. The results of an HMM model is hugely dependent on proper choice of model. Table 4.2 illustrates the results of the test files for the two different models. The word accuracy is calculated as described in Section 5.1.

File	Original Word Accuracy	HMM Model	New Word Accuracy
File1	95.5	26 states	89.04
		27 states	96.5
File2	95.9	26 states	92.5
		27 states	98.8
File3	94.15	26 states	85.06
		27 states	94.8
File4	97.5	26 states	91.2
		27 states	98.4
File5	97.4	26 states	91.4
		27 states	97.4

Table 4.2: Illustration of HMM output for different Model choice

Table 4.2 shows that the word accuracy of the datasets decreased when the HMM was modeled with 26 states.

4.3.3 Lack of training Data

Lack of training data results in parameters receiving poor values and certain characteristics of the source being modeled incorrectly. Some case of emission and transitions may not occur in the training data. The model will never recognize such outcomes and will state those as impossible observations by assigning *zero* probabilities values. One appraoch to solve this problem is the smoothing technique. Another remedy is to reduce the size of the model, thus reducing the number of parameters that need to be estimated. However, such change is undesirable when the model is chosen to accomodate some inherent characteristics of a source. In our experiments, a minimum value for all probability values was enforced.

CHAPTER 5

EXPERIMENTS AND RESULTS

In this chapter we provide the definition of the performance measures for the system and the values obtained. These values give insights about the performance of the post-processing system. The system is tested for the dataset described in Section 4.1. The value of the performance measures for the test dataset are shown in Section 5.2.

5.1 Performance Measures

To evaluate the performance of our HMM, we need to determine the evaluation measures. There are four possible outcomes that can occur when applying an HMM to an incorrect text.

1. correct → correct: A correct character is still correct at HMM output. This is a true negative (TN).

2. correct → wrong: A correct character is *corrected* to a wrong character at HMM output. This is a false positive (FP).

3. wrong → correct: A character is corrected by the HMM. This is a true positive (TP).

4. wrong → wrong: A wrong character is still wrong at HMM output. This is a false negative (FN).

Recall and Precision values are calculated using TP, FP and FN as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$$

The definition of these measures has been proposed and discussed by Reynaert [21]. Recall values show the ability of the system to correct errors and the precision value indicates the accuracy of the system. Other performance measure are word accuracy and error reduction rate. Word accuracy is determined as defined by Taghva in his study OCRSpell [9].

$$\text{Word Accuracy} = \frac{\text{number of words recognized correctly}}{\text{total number of words}}$$

The error reduction rate is calculated as follows:

$$\text{Error Reduction Rate} = \frac{\text{Corrected errors - Introduced errors}}{\text{Total errors}}$$

5.2 Results

In this section, the statistics of the performance measures for the test datasets are tabulated.

5.2.1 Recall and Precision

Table 5.1 shows that values of TP, TN, FP and FN for the test datasets. We observe that the HMM was able to make some corrections (TP values), but it also introduced some errors (FP values). The observation on introduced errors are explained in Section 5.3. In an average, recall of 43.38% and precision of 75.34% were obtained.

Test File	TP	TN	FP	FN	Recall	Precision
File1	2	1109	0	7	22.2	100
File2	5	1002	0	2	71.4	100
File3	3	901	2	6	33.3	60.0
File4	2	1172	1	3	40.0	66.7
File5	2	1192	2	2	50.0	50.0
$\sum$	14	5376	5	20		

Table 5.1: Measures of TP, TN, FP and FN for the test datasets

5.2.2 Word Accuracy

Table 5.2 shows that the HMM was able to improve the word accuracy of the test files. In an average, the word accuracy of the test file increased from 96.3% to 97.3%.

Test File	Initial Word Accuracy	No. of Words	No. of Words Recognized Correctly	New Word Accuracy
File1	95.5	201	194	97.4
File2	95.9	174	172	98.8
File3	94.15	154	146	94.8
File4	97.5	206	202	98.4
File5	97.4	198	194	97.4

Table 5.2: Measure of word accuracy for the test datasets

5.2.3 Error Reduction Rate

The error reduction rate determines how many errors were reduced by the system. The number of corrected errors is equal to the TP values and the number of introduced errors is equal to the FP values. The error reduction rate obtained is 26.14%.

Test File	Error Count	Corrected Errors	Introduced Errors
File1	9	2	0
File2	7	5	0
File3	9	3	2
File4	5	2	1
File5	4	2	2
$\sum$	34	14	5

Table 5.3: Measure of error reduction rate for the test datasets

5.3 Observation

In the result section we found that some new errors were introduced by the HMM (FN values). This section explains the occurence of those new errors. Second order viterbi algorithm uses the transition and emission probabilities to find the optimal sequence. Since training data determines the value of these probabilities, we have to go through the training data to find the possible reason for the introduced error. We observed that *just judgment* → *just hudgment*. In the training file, we found that the count of transition *(t whitespace j)*, the transition of *whitespace* to *j* given the earlier state is *t*, is 9 whereas the count of transition *(t whitespace h)*, the transition of *whitespace* to *h* given the earlier state is *t*, is 109. Therefore, transition probability $a_{\text{t whitespace h}}$ is higher than $a_{\text{t whitespace j}}$. For the error, *quantity* → *huantith* we found count of transition *(ity)* is 76 whereas the count of transition *(ith)* is 188. For the error, *salem* → *salea* we found count of transition *(lem)* is 42 whereas the count of transition *(lea)* is 56. We observed that higher the number of occurence of a transition, higher is the likelihood for the transition to be in the optimal state sequence. Therefore, the output of HMM is dependent on transitions in the training file.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, we presented a post processing system based on second order HMM. The motivation of this study was to propose a system for correcting the errors in OCR generated text. For this purpose, we designed a HMM model, implemented the MLE and Viterbi Algorithm in Java. The proposed system achieved an error reduction rate of 26.4 % on the test data. In an average, recall of 43.38% and precision of 75.34% were obtained.

A study based on first order HMM for correcting OCR errors achieved recall of 11.76% and precision of 100%. The high recall value indicates that the second order model corrected more error compared to the first order model. Of the three different types of OCR errors that we have classified, the system was able to correct most of the 1:1 errors.

Adding more states and observation symbols to the HMM and design of advanced preprocessing module to detect the OCR errors can potentially improve the performance of the system. Further studies on improvement of the system output are the prospect as future work.

APPENDIX A

The Standard OCR Procedure

Figure A.1 shows the typical OCR process. The procedure involves four standard steps.

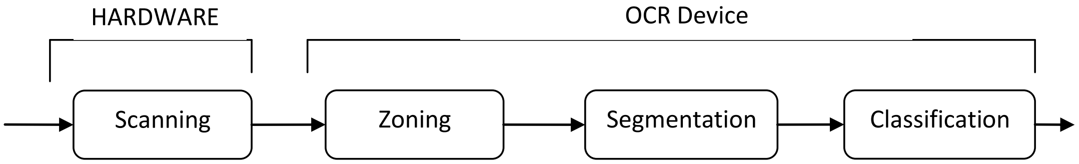
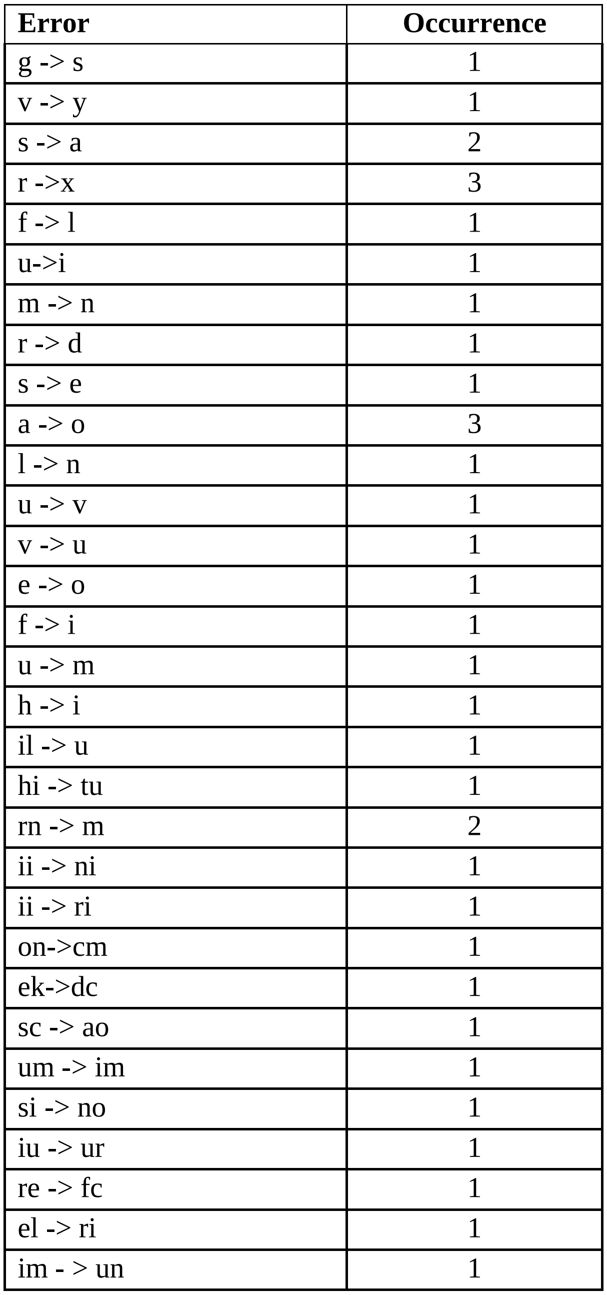


Figure A.1: Standard OCR Procedure

1. scanning the paper documents to procedure an electronic image
2. zoning which automatically orders the various regions of the text in the documents
3. the segmenation process breaks the various zones into their respective components (zones are decomposed into words and words are decomposed into characters)
4. the characters are classified into their respective ASCII characters

APPENDIX B

List of OCR errors in training data



Error	Occurrence
g -> s	1
v -> y	1
s -> a	2
r ->x	3
f -> l	1
u->i	1
m -> n	1
r -> d	1
s -> e	1
a -> o	3
l -> n	1
u -> v	1
v -> u	1
e -> o	1
f -> i	1
u -> m	1
h -> i	1
il -> u	1
hi -> tu	1
rn -> m	2
ii -> ni	1
ii -> ri	1
on->cm	1
ek->dc	1
sc -> ao	1
um -> im	1
si -> no	1
iu -> ur	1
re -> fc	1
el -> ri	1
im - > un	1

Figure B.1: List of OCR errors in training data

Error	Occurrence
v -> y	1
in ->m	1
l -> j	2
e -> t	2
f ->i	5
a-> o	10
l-> i	8
n -> m	3
e -> u	1
l -> i	1
m -> x	1
y -> jr	1
u -> y	1
g -> s	1
e -> r	1
t -> i	2
y -> s	1
si -> m	1
a -> f	1
r -> i	1
nr -> m	1
s -> i	1
c -> o	2
l -> i	2
o -> c	1
in -> m	1
m -> n	2
x -> i	3
v -> u	2
j -> i	1
e -> o	1
f -> i	1
in -> m	2
im -> un	1
ir -> n	1
i -> l	1
a -> c	1
a ->e	1
c -> o	1
n -> ii	3
ii -> n	2

Error	Occurrence
b -> e	1
r->s	2
h->ii	1
m->ni	5
al->ri	1
um->im	2
rm->nn	4
g->s	3
el->d	2
is->d	2
ll->u	2
ri->ii	3
r->ir	2
y->rs	1
s->nd	1
f->t	4
a->u	3
ci->d	2
rn->m	2
l->f	2
t->l	1
r->x	5
z->s	2
a->z	4
in->m	2
c->e	7
r->l	3
f->t	2
f->l	1
iu ->ur	1
h -> b	2
g ->c	3
rc ->n	1
y -> j	1
a -> u	2
v -> r	1
r -> e	2
e-> r	2
el -> ri	1
x -> z	2

[1] L.E. Baum and T. Petrie, “Statistical inference for probabilistic functions of finite state Markov Chains,” *The Annals of Mathematical Statistics*, vol. 37, pp. 1554–1563, 1966.

[2] L.E. Baum , “An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes,” *Inequalities*, vol. 3, pp. 1–8, 1970.

[3] L.R Rabiner and B.H Juang, “An Introduction to Hidden Markov Model,” *ASSP*, vol. 3, no. 1, pp. 4–16, 1986.

[4] S.E. Levinson, L.R. Rabiner and M.M. Sondhi , “An Introduction to the application of the theory of probabilistic functions of Markov process to Automatic Speech Recognition,” *Bell System Technical Journal*, vol. 62, no. 4, pp. 1035–1074, 1983.

[5] L. 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, 1989.

[6] F. J. L. R. Bahl and R. L. Mercer, “A Maximum Likelihood Approach to Continuous Speech Recognition,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 179–190, 1983.

[7] A. Kundu, Y. He, and P. Bahl, “Recognition of handwritten word: First and second order hidden markov model based approach,” *Pattern Recognition*, vol. 22, no. 3, pp. 283 – 297, 1989.

[8] S. R. Veltman and R. Prasad, “Hidden markov models applied to on-line handwritten isolated character recognition,” *IEEE Transactions on Image Processing*, vol. 3, no. 3, pp. 314–318, 1994.

[9] K. Taghva and E. Stofsky, “OCRSpell: an interactive spelling correction system for OCR errors in text,” *IJDAR*, vol. 3, no. 3, pp. 125–137, 2001.

[10] J. F. Mari, J. P. Haton, and A. Kriouile, “Automatic Word Recognition Based on Second-Order Hidden Markov Models,” *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 1, pp. 22–25, 1997.

[11] K. Taghva, J. Borsack, B. Bullard, and A. Condit, “Post-Editing Through Approximation and Global Correction,” *IJPRAI*, vol. 9, no. 6, pp. 911–923, 1995.

[12] K. Taghva, R. Beckley, and J. Coombs, “The Effects of OCR Error on the Extraction of Private Information,” in *Document Analysis Systems VII*, vol. 3872, pp. 348–357, 2006.

[13] Kazem Taghva, Allen Condit, Julie Borsack, John Kilburg, Changshi Wu, and Jeff Gilbreth, "The MANICURE Document Processing System," tech. rep., Information Science Research Institute, University of Nevada, Las Vegas, March 1995.

[14] Kazem Taghva, Allen Condit, and Julie Borsack, "An expert system for automatically correcting OCR output," in *In Proceedings of the 1994 International Symposium on Electronic Imaging Science and Technology*, pp. 270–278, 1994.

[15] K. Kukich, "Techniques for automatically correcting words in text," *ACM Comput. Surv.*, vol. 24, pp. 377–439, Dec. 1992.

[16] S. M. Thede and M. P. Harper, "A second-order Hidden Markov Model for part-of-speech tagging," in *Proceedings of the 37th Annual Meeting of the ACL*, pp. 175–182, 1999.

[17] L. Vyas, "Finding Acronyms and their definitions using HMM," Master's thesis, May 2011.

[18] C. Zhai, "A Brief Note on the Hidden Markov Models(HMMs)," March 2003.

[19] J. Esakov, D. P. Lopresti, and J. S. Sandberg, "Classification and distribution of optical character recognition errors," in *Proceedings of the IS&T/SPIE International Symposium on Electronic Imaging*, (San Jose, CA), February 1994.

[20] Y. H. Y. He, "Extended Viterbi algorithm for second order Hidden Markov process," *1988 Proceedings 9th International Conference on Pattern Recognition*, vol. 718, no. 1, pp. 718–720, 1988.

[21] Martin Reynaert, "All, and only, the Errors: more Complete and Consistent Spelling and OCR-Error Correction Evaluation," in *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco*, European Language Resources Association, 2008.

VITA

Graduate College
University of Nevada, Las Vegas

Srijana Poudel

Degrees:
Bachelor of Computer Enginnering 2007
Institute of Engineering, Pulchowk Campus, Tribhuvan University

Thesis Title: Post Processing of Optically Recognized Text via Second Order Hidden Markov Model

Thesis Examination Committee:
Chairperson, 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.