

5-2011

Implementation of hidden semi-Markov models

Nagendra Abhinav Dasu

Follow this and additional works at: <https://digitalscholarship.unlv.edu/thesesdissertations>



Part of the [Applied Mathematics Commons](#), and the [Theory and Algorithms Commons](#)

Repository Citation

Dasu, Nagendra Abhinav, "Implementation of hidden semi-Markov models" (2011). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 997.
<https://digitalscholarship.unlv.edu/thesesdissertations/997>

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

IMPLEMENTATION OF HIDDEN SEMI-MARKOV MODELS

by

Nagendra Abhinav Dasu

Bachelor of Technology in Information Technology
Jawaharlal Nehru Technological University, India
May 2009

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

Copyright by Nagendra Abhinav Dasu 2011
All Rights Reserved



THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Nagendra Abhinav Dasu

entitled

Implementation of Hidden Semi-Markov Models

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

Muthukumar Venkatesan, Graduate Faculty Representative

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

May 2011

ABSTRACT

Implementation of Hidden Semi-Markov Models

by

Dasu Nagendra Abhinav

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

One of the most frequently used concepts applied to a variety of engineering and scientific studies over the recent years is that of a Hidden Markov Model (HMM). The Hidden semi-Markov model (HsMM) is contrived in such a way that it does not make any premise of constant or geometric distributions of a state duration. In other words, it allows the stochastic process to be a semi-Markov chain. Each state can have a collection of observations and the duration of each state is a variable. This allows the HsMM to be used extensively over a range of applications. Some of the most prominent work is done in speech recognition, gene prediction, and character recognition.

This thesis deals with the general structure and modeling of Hidden semi-Markov models and their implementations. It will further show the details of evaluation, decoding, and training with a running example.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to each and everyone who helped me grow as a student and as a person all through this spell of my life.

It gives me great pleasure in acknowledging the support and help of my committee chair, Dr. Kazem Taghva for all his guidance through every stage of this thesis research. I attribute the level of my Masters degree to his encouragement and effort and without him this thesis would not have been completed.

I would like to thank my committee members, Dr. Ajoy K. Datta and Dr. Laxmi Gewali, whose work demonstrated to me the importance of simplicity and values. It was a pleasure taking courses offered by them which were not only interesting but offered a lot of challenges. Special thanks to Dr. Venkatesan Muthukumar for accepting my request and being a part of my committee. I would also like to thank all the professors of the CS department that I have worked with and also offer my special thanks to Mr. Mario and Mrs. Sharon of the CS office for guiding me in every step.

This thesis is dedicated to my parents who have given me the opportunity of an education from the best institutions and support throughout my life. I have always believed in my family being my biggest strength and henceforth I would like to thank all my family members especially my brother and my cousins. Finally, I would like express my

love to my friends for their constant understanding and backing me in the toughest of times. I owe a lot to each and every one of them.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	iiv
LIST OF FIGURES	vii
CHAPTER 1 INTRODUCTION	1
1.1 Theoretical Background of a HMM	1
1.2 Need of the HsMM over an HMM	1
1.3 General Structure of HsMM	2
1.4 Application Areas of HsMM	4
1.5 Common Issues with Semi-Markov Model	4
CHAPTER 2 HsMM: ELEMENTS AND RELATED PROBLEMS	6
2.1 Model Parameters	6
2.2 Three Basic Problems of HMM	7
2.3 Forward Algorithm.....	8
2.4 Backward Algorithm	10
2.5 Viterbi Algorithm	12
2.6 Maximum Likelihood Estimation.....	16
2.7 Baum – Welch Algorithm	17
CHAPTER 3 IMPLEMENTATIONAL DETAILS	22
3.1 Changes from the Traditional HMM	22
3.3 Smoothing.....	37
CHAPTER 4 RESULTS EVALUATION	41
4.1 Results	41
4.2 Advantages Over the Traditional HMM	51
CHAPTER 5 CONCLUSION AND FUTURE SCOPE.....	53
BIBLIOGRAPHY	55
VITA	59

LIST OF FIGURES

Figure 1: General Model of HsMM	3
Figure 2: Example of State Transitions in HsMM.....	3
Figure 3: Operations Required for Forward Variable Generation.....	9
Figure 4: Forward Variable Generation for N States for Time T.....	9
Figure 5: Operations Required for Backward Variable Generation	11
Figure 6: Backward and Forward Variable Generation in a System	11
Figure 7: Example Structure of a Trellis Diagram.....	15
Figure 8: Example of Backtracking Using the Trellis Diagram	15
Figure 9: Duration of Each State in a HsMM	23
Figure 10: Sample Model File for HsMM.....	25
Figure 11: Tagged Training Data Set for an HsMM	26
Figure 12: Untagged Training Data Set for an HsMM	27
Figure 13: Screenshot of the Project Directory and its Compilation.....	42
Figure 14: Screenshot Showing the Input Files for Decode Algorithm ...	43
Figure 15: Execution of Decode Command and the Resulting File	44
Figure 16: Alternate Input Files for the Decode Command.....	45
Figure 17: Execution and the Result File for Alternate Inputs	45
Figure 18: Execution of the Count Procedure	47
Figure 19: Resulting Model File After Training Using Count	48
Figure 20: Result File Using Viterbi on Trained Model Using Count.....	49
Figure 21: The Final Likelihood Using Baum Welch Algorithm	50
Figure 22: Result File on the Trained Model Using Baum Welch.....	51

CHAPTER 1

INTRODUCTION

1.1 Theoretical Background of a HMM

Hidden Markov Models (HMMs) is a widely used statistical model. It is modeled by a Markov process in which the states are hidden, meaning the state sequence is not observable. The markov chain which forms the structure of this model is discrete in time.

The definition of a HMM contains five variables namely, (S, V, π, A, B) . Here S is set of N finite states, V is the vocabulary set. π indicates the initial state probabilities, A is the state transition probabilities while B is the emission probabilities.

HMMs are being used from many decades in the domain of pattern recognition, covering speech and handwriting recognition. In spite of its range of applications, HMM has the limitation of working only with discrete outputs. Consequently, HMM has been extended into various models generating continuous outputs or Gaussian outputs or even the mixtures of Gaussian outputs.

1.2 Need of the HsMM over an HMM

Among many extensions of the traditional HMM, Hidden semi-Markov models is the most frequently used. HsMM, contrary to the simplicity of HMM, allows for general length distributions not solely geometric. This is

the main advantage. We can classify the HsMM into a class of models in which the state of the hidden Markov process influences the distribution of the observation sequence. HsMM is being extensively applied in the field of automatic speech recognition. On a whole, this theory and its applications are rapidly expanding to many other areas.

The proliferation of the theory, and its subsequent use, can be accredited to the model's [3]:

- Likelihood; that takes linear time to compute.
- Interpretability; which is shown in a range of cases
- Availability of conditional distributions
- Usability; for, even in cases missing a few observations it can be handled with minimal effort

1.3 General Structure of HsMM

As mentioned before, the HsMM's capacity extends beyond that of the HMM. It allows for every hidden state to be a semi-Markov chain while also introducing the concept of state duration. This means that, unlike in HMM where a state can emit one observation per state, a state in HsMM can emit a sequence of characters. The length of the observation sequence for each state is determined by the duration variable of each state. Consequently, in addition to the standard notation of a HMM, a state duration variable is added for the HsMM. This is an integer variable and takes the value from the set $\{1, 2, \dots, D\}$, where D is the

maximum duration allowed for a single state.

Below is a figure depicting the general HsMM structure [1]. The initial state and its duration are selected according to the initial transition probabilities. In this case, the first state produces two observations hence the duration equals two and transitions into the second state. The second state then produces an observation sequence length of four. This can be seen for the remaining states till time (T).

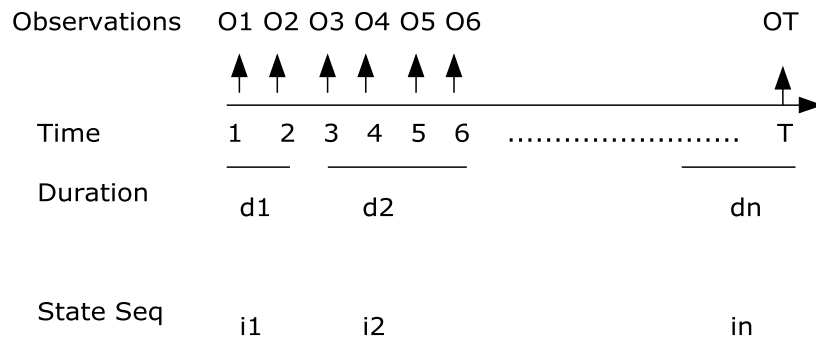


Figure 1: General Model of HsMM

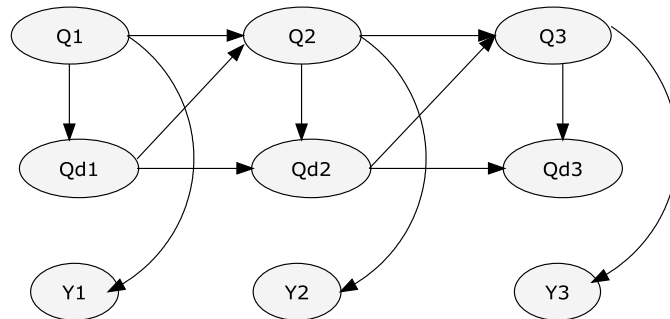


Figure 2: Example of State Transitions in HsMM

Figure 2 shows a HsMM with duration Q_t^d for each state Q_t [4]. We can clearly see from the above figure that a state generates the observation sequence (here, Y_i for a state i) and after it has completed its duration in that state, it transitions into the next state.

1.4 Application Areas of HsMM

The applications of HsMM are varied and far-reaching. Approximately thirty areas of interest are working with various tools based on HsMM. Examples include: human activity recognition, printed text recognition, recognition of human genes, MRI brain mapping, language identification, speech synthesis, remote sensing, image segmentation, handwriting recognition, Internet traffic modeling, classification of music, event recognition in videos, mobility tracking in cellular networks, symbolic plan recognition, etc. Among these, the major applications include speech synthesis, handwriting recognition, anomaly detection and MRI brain mapping. A complete list of the applications is given in [1].

1.5 Common Issues with Semi-Markov Model

HsMM has its own share of issues and disadvantages; for example:

- To reduce the complexity of computation, it makes a few significant assumptions e.g., the Markovian assumption and the Gaussian mixture assumption. The Markov assumption allows the case where in the current state is dependent only on its immediate

previous state resulting in a First order HsMM. The Gaussian mixture assumption allows the distribution to be calculated by picking a random component.

- HsMM prerequisites a large number of parameters for its efficient working.
- Consequently, the efficiency of HsMM is contingent upon large data sets to perfectly train the system.
- Lastly, implementation involves several additions to the original algorithm.

An overview of HsMMs is presented in this paper. It includes modeling, estimation and implementation. Specifically, the paper first describes the modified forward and backward algorithms used, along with the details on Viterbi and Baum-Welch algorithms. Secondly, the paper provides the implementation details and its corresponding results. Finally, we try to analyze the results obtained and draw suitable conclusions from them.

CHAPTER 2

HsMM: ELEMENTS AND RELATED PROBLEMS

2.1 Model Parameters

As we have described before, HsMM has an extra state duration variable along with the traditional HMM variables. Formally, we define an HsMM as a 6-tuple (S, V, Π, A, B, Pd) . In this paper, the model is denoted as $\lambda = (\Pi, A, B, Pd)$ to denote all the parameters.

We describe the above parameters below [5],

- The state transition probability distribution A defined as,

$$a_{ij} = \Pr\{q_j \text{ at } t+1 \mid q_i \text{ at } t\}$$

- The emission probability distribution in state i ; B

$B_{ij}(O_t) = P(y(S_t) \mid S_t = q, L_t = d)$. Here, $y(S_t)$ is the sequence of symbols emitted by the general state S_t . The length of the sequence is given as 'd'.

- The initial state distribution $\Pi = \{\Pi_i\}$ where

$$\Pi_i = \Pr\{i_1 = q_i\}$$

- The duration probability Pd defined as

$$X = p(d \mid q_i)$$

$$P(d \mid q_i) = \Pr\{\text{duration}(q_i) = d\}$$

Let us now see a typical working of a HsMM. It shows the general flow of a standard model [5].

- An initial state is chosen referring to the distribution given in Π .

- An observation sequence length d_1 is chosen from P_d .
- The sequence $o_1o_2\dots o_{d_1}$ is emitted according to the value b_{i_1} in B .
- The next state i_2 is chosen from A , the transition matrix.
- The above process is repeated for i_2 and all the remaining states.

2.2 Three Basic Problems of HMM

There are three basic problems that must be addressed in order for the HsMM to be applicable in any of the research areas. These are commonly referred by the names:

- Observation Probability Problem
- Decoding Problem
- Training Problem

We describe these problems in brief below [5].

Observation Probability Problem is an evaluation problem. In simple words, given the HsMM model parameters and a set of observations, we have to evaluate the probability of the HsMM producing the observations. This is useful to pick a model which appropriately matches our observations. Forward algorithm is employed to solve this problem. This is clearly explained in the sections 2.3.

Decoding Problem as the name suggests, it tries to recognize the hidden states. It means, given an observation sequence we try to find the best possible sequence of states that optimizes the observations. This is used often in many real world applications such as, state sequences in

speech recognition or to learn the structure of the HsMM itself. The most popular recognition algorithm is the Viterbi algorithm. It is also referred as a Decoding algorithm. It is presented in section 2.5.

Training Problem is used mainly to optimize our HsMM parameters. This is done so as to best match the given sequence of emission symbols. For this, we 'train' our model with a set of observations referred as a 'training set'. The training process is an important part of development as this phase determines the efficiency of the system. Large datasets with suitable data can optimize the parameters and in turn create adept systems. Training is typically done in two ways. A supervised way which trains the model using tagged sequences specifying the states and its corresponding emissions. This is called the Maximum Likelihood Estimation algorithm. The second way is an unsupervised method where a recursive Baum – Welch algorithm is made used. We will describe both these methods in sections 2.6 and 2.7

2.3 Forward Algorithm

The Forward procedure is an easy way to solve the observation probability problem. We proceed to this solution by first defining a forward variable, $\alpha_t(i) = P(o_1, \dots, o_t, q_t = s_i | \lambda)$. To put it in simple words, it is the probability of partial observation sequence $o_1 o_2 \dots$ till o_t was generated and we reach state state s_i at time t , given our model λ . These values are computed recursively over time T .

The figure below shows the sequence of operations done in order to generate a forward variable value [7].

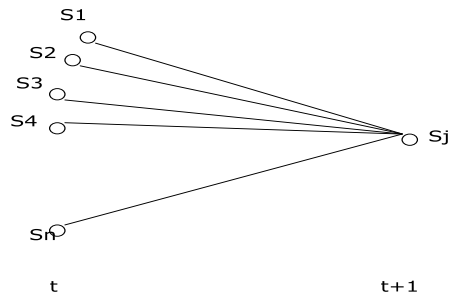


Figure 3: Operations Required for Forward Variable Generation

This is another illustration displaying the computations when given multiple states and observations. In this figure, there are 'T' observations and 'N' states. A variable at time 't' in every state 'i' is dependent on all the previous forward variables at time 't-1' [7].

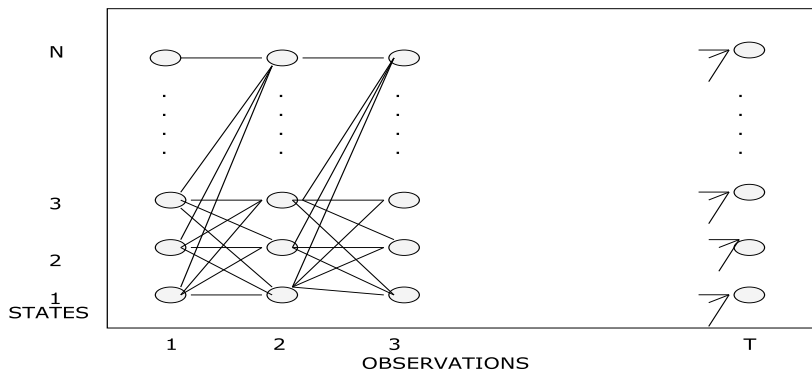


Figure 4: Forward Variable Generation for N States for Time T

The process of calculating this variable is recursive and hence has three steps.

i) Initialization:

$$\alpha_1(j) = \pi(j)b_j(O_1), \quad 1 \leq j \leq N$$

ii) Induction:

$$\alpha_t(j,d) = P(y_{t-d+1:t} | j,d)P(d | j)\sum A(i,j)(\sum \alpha_{t-d}(i,d^i))$$

iii) Termination:

$$P(O | \lambda) = \sum_{i=1}^N \alpha_T(i,d')$$

The initialization is the product of the probability of state S_j being the initial state with the emission probability. The induction takes place until the end of time sequence T . The forward variable is computed iteratively at each instant for each state. This value depends on the previous α value. Finally, the sum of terminal forward variables gives the value of $P(O | \lambda)$ [4].

2.4 Backward Algorithm

The first problem of Observation probability is solved by the forward variable since we manage to compute $P(O | \lambda)$. But, in order to solve the second problem of decoding, we need to have a second variable called the Backward variable- denoted as β .

This value is defined as, $\beta_t(i) = p(o_{t+1}, \dots, o_T | q_t = s_i, \lambda)$. It is defined as the probability of generating the partial sequence from $t+1$ to T and given the HsMM model and the state at time 't' being s_i [7].

This following figure better illustrates this concept. It gives the set of computations required for calculating a single backward variable $\beta_t(i)$ [7].

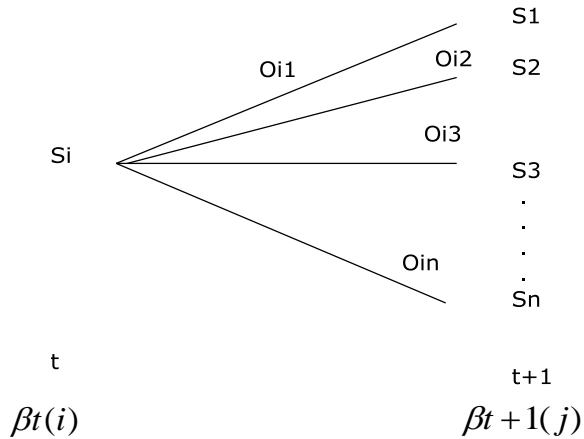


Figure 5: Operations Required for Backward Variable Generation

The following shows the set of computations that are necessary in when the system being in state 'j' at 't+d' and in the state 'i' at 't' [3].

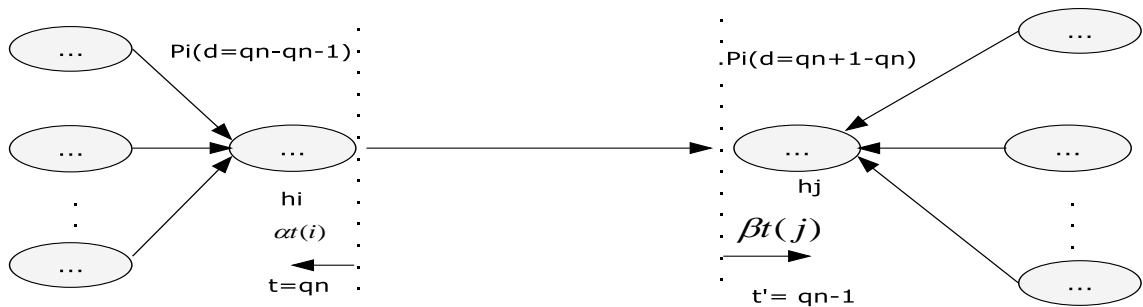


Figure 6: Backward and Forward Variable Generation in a System

Similar to the forward variable, this is also a recursive process.

i) Initialization:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

ii) Induction:

$$\beta_t(i, d') = \sum \sum \beta_{t+d}(j, d) P(y_{t+1:t+d} | j, d) P(j | i) P(d | j)$$

As shown in Figure 5, $\beta_T(i)$ is defined as 1 for all the N states. From the induction formula we can infer that we have to consider all the states S_i at time $t+d$, all the transitions from S_i to S_j and the sequence of observations in state j at $t+d$ in order to compute the probability of being in state S_i at t . All these calculations are clearly explained and derived in [4]. The total computational complexity of forward- backward algorithm is $O(N^2LT)$ where L is the total sum of durations for all the N states.

2.5 Viterbi Algorithm

The Decoding problem is solved using the Viterbi algorithm. It is one of the most frequently used methodologies in recent times. This is a dynamic algorithm that computes, for a given sequence of emissions the most likely state transition path. In essence, this is quite similar to the 'Forward algorithm'. The slight difference being that we use 'max' rather than a summation over all the paths available to arrive at a particular state.

Let the set of states be $Q = \{q_1, q_2, q_3 \dots q_t\}$ and the given observation sequence be $O = \{O_1, O_2, O_3, \dots O_t\}$.

The algorithm calculates the following quantity [7]:

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P(q_1 q_2 \dots q_t = i, O_1 O_2 \dots O_t | \lambda)$$

The above variable stores the path with the highest probability at time t of ending in state S_i . Along with this, we also need another variable $\Psi_t(i)$. It allows us to dynamically store the 'best path' to the state S_i at the instant ' t ' [8]. Hence, the probable best path is calculated for each of the states before reaching the final state. Also, the path reached is comprehensive at each and every state.

Similar to the Viterbi used in traditional HMM, HsMM also uses dynamic programming to compute the best path. At each time instant ' t ', the algorithm computes the forward variable for each state ' i '.

$$\alpha(j)^T_t = \max_{i=1}^N \max_{d=1}^D \alpha_{t-d}(i) a_{ij} P(d | j) \prod_{j=i-d+1} b_j(o_i)$$

The sequence of steps for the modified Viterbi algorithm for Hidden semi-Markov model is given below [3]:

- STEP 0 : Store the indexes:

States: $1 \leq i, j \leq N$

Time: $0 \leq t \leq T$

Duration: $0 \leq d \leq D$

- STEP 1 : Initialization

At time $t = 0$

$$\alpha_0(i) = \pi(i) b_i(O_1), \quad 1 \leq i \leq N$$

- STEP 2: Iterations

From $t > 1$ and for $1 \leq j \leq N$

Do

for t = 1 to T do

for j = 1 to N do

$$\alpha_t(j) = \max_{i=1}^N \max_{d=1}^D \alpha_{t-d}(i) a_{ij} P(d | j) \prod_{j=i-d+1} b_j(o_i)$$

end for

end for

- Step 3 : Backtracking

The state 'i' and duration 'd' which gives the best path by optimizing the above equation are stored in the variables $\delta_t(i)$ and $\Psi_t(i)$. The variable $\Psi_t(i)$ is mainly used to backtrack the best path from the final state.

The **Trellis Diagram** is a powerful tool to better comprehend the idea of backtracking and identifying various paths discovered in Viterbi [8]. Every column in this diagram corresponds to an instant of time. There are N numbers of rows, N being the number of states. Transitions among states connect a column with its adjacent. At each instant, every state is indicated with its corresponding emission probabilities. Figure 7, clearly explains a trellis diagram.

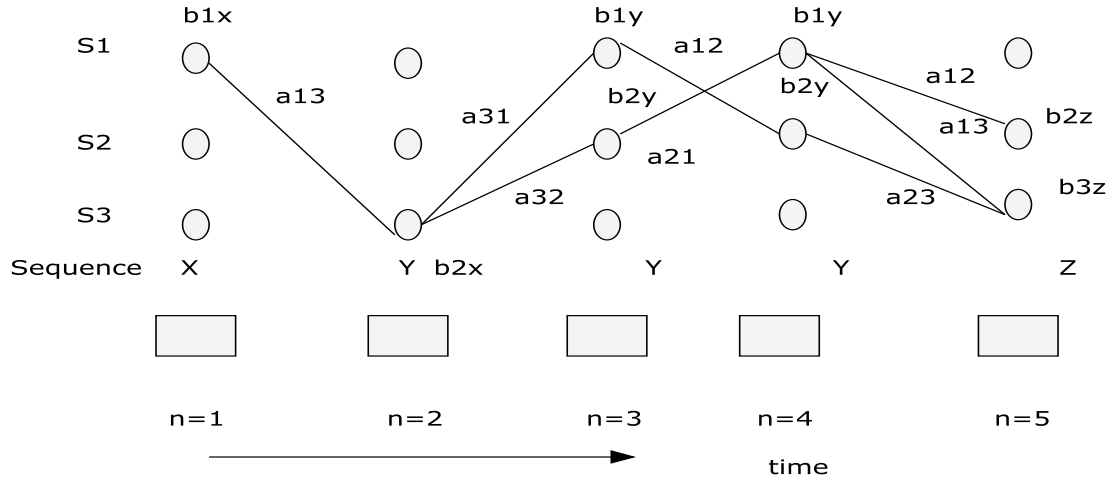


Figure 7: Example Structure of a Trellis Diagram

In the above figure, the sequence of symbols for every instant 't' is given below. There are 3 states assumed with given transition and emission probabilities. After running the algorithm we can use the trellis diagram to trace back the path using the two variables $\delta_t(i)$ and $\Psi_t(i)$ as shown in Figure 8 .

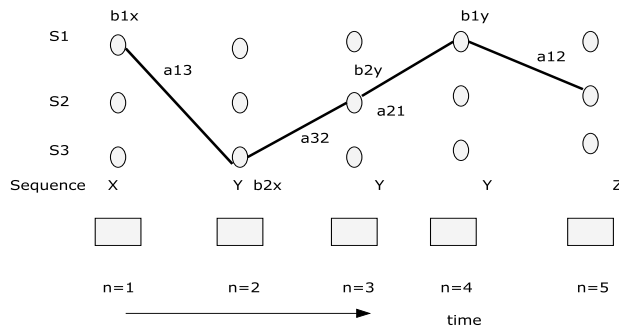


Figure 8: Example of Backtracking Using the Trellis Diagram

The above illustrates an example with 3 states and a sequence of length 5. As we can see the path with highest $\delta_t(i)$ is identified and its corresponding $\Psi_t(i)$ is used to backtrack through the nodes.

It takes $O(ND_i)$ time to compute the forward variable for a single iteration, where D_i is the duration allowed for that state. Hence, the total complexity of the algorithm is $O(N^2LT)$, where T is the total time and L is the sum of durations of N states.

2.6 Maximum Likelihood Estimation

The maximum likelihood estimation technique is a popular statistical method often used to estimate the model parameters which maximize the likelihood. In other words, this method comes up with a set of values that are most likely to produce the probability distribution derived from the observed data [15]. MLE is widely used because of the following reasons [14]:

- It is simple to compute.
- It is invariant if the parameters are changed.
- MLE implements the concept of likelihood.
- With enough data, it is mostly very efficient.

Given a tagged sequence of data, containing both the symbols emitted along with their corresponding states, the MLE comes up with the following calculations to determine the model parameters [16].

- Transition Probability: Let the two states be S_i and S_j and we want to make a transition from S_i to S_j . Then,

$$P(S_i, S_j) = \frac{\text{Number of transitions from } S_i \text{ to } S_j}{\text{Total number of transitions out of } S_i}$$

- Emission Probability: To calculate the probability of emitting symbol w from state s , we use

$$P(w | s) = \frac{\text{Number of times symbol } w \text{ is emitted at state } s}{\text{Total number of symbols emitted from state } s}$$

- Initial State Probability: The formula to calculate the probability of a state s being the starting state is ,

$$P(s, 1) = \frac{\text{Number of times } S \text{ emits the first symbol}}{\text{Total number of first symbol emissions by all the states}}$$

Hence, this is a simple counting technique that helps in improving the model parameters. But this algorithm is not really recommended for large data or in the cases where it is hard to tag the data. Hence, we next learn about an unsupervised technique which though complicated, is effective in every case.

2.7 Baum – Welch Algorithm

The most complicated and by far the most essential algorithm needed to develop an efficient HsMM is the Baum – Welch algorithm. This algorithm is an unsupervised training algorithm and solves the **Training**

problem. It was invented by Leonard E. Baum and Lloyd R Welch [9]. This comes under the class of Estimation Maximization algorithms.

The algorithm works in two stages. Estimate or initialize the parameters of HsMM and then maximize them. We will now discuss them in detail.

Estimation: We start off with random guesses for the parameters of the model. Values for the transition matrix, the emission probabilities and the initial state probabilities along with those of state duration probabilities are assigned randomly generated values.

Maximization: This process is iterative. After the initial values are assigned the algorithm enters into a loop where it uses the provided training data. Based on the tags and related probabilities, new values for the model parameters are evaluated and assigned. Hence, after every iteration we get a model which better matches the given data. The iterations continue until we reach a stage where the improvement in the parameters is smaller than an assigned threshold.

In the above manner, Baum – Welch algorithm helps to get an optimized HsMM model. We now define formally the algorithm [10].

Algorithm: Baum Welch

INPUT:

The observation sequence: $O^1, O^2, \dots O^T$.

INITIALIZATION:

Assign random values to HsMM model parameters: $\lambda = (A, B, D, \Pi)$.

MAXIMIZATION:

Repeat

{

$$\lambda = \lambda'; \Pi = \Pi';$$

For each observation sequence at every time instant 't'.

{

Calculate all the probable paths to that state.

Calculate $\alpha(t,i)$ and $\beta(t,i)$ using Forward – Backward algorithms.

Calculate the amount of change that occurred on the transition matrix.

Calculate the amount of change that occurred on the emission matrix.

}

Calculate the new model parameters.

}

Until (the change in the parameter values is less than the predefined threshold)

We use a set of formulas to compute and update the model parameters. Though we can express them using the forward and backward variables along with the current parameter values, the computation becomes far simpler with the use of two new variables.

- $\gamma_t(i)$ is defined as the probability of being in state S_i for a given set of

emission symbols at time t.

- $\xi_{t(i,j)}$ is the probability of a transition occurring between states S_i and S_j at time t for the given model and the set of observations.

The formal definitions of the above variables are given below [3].

$$\gamma_{t(i)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(i)\beta_t(i)}$$

$$\xi_{t(i,j)} = \frac{\sum_{t=1}^T \alpha_t(i) a_{ij} \sum_{d=1}^D P(d | j) b_j(O_{t+1}^t) \beta_t(j)}{\sum_{t=1}^T \sum_{i=1}^N \sum_{j=1}^N \xi_{t(i,j)}}$$

Following are the Parameter re-estimation formulas or the HsMM [3]:

- Initial State Distribution Estimation:

The probability of a state 'i' being the starting state for a given observation set O.

$$\Pi'_i = \frac{\Pi_i \left[\sum_{d=1}^D \beta_d(i) P(d | i) b_j(O_{d_1}) \right]}{P(O | \lambda)}$$

- State Transition Probabilities Estimation:

The probability of a state 'i' moving to the next state 'j' after its duration d_i .

$$a'_{ij} = \frac{\sum_{t=1}^T \xi_{t(i,j)}}{\sum_{t=1}^T \sum_{i=1}^N \sum_{j=1}^N \xi_{t(i,j)}}$$

- Emission Probabilities Estimation:

It is the probability of the segment of observations that occurred in

state S_i normalized by the probability of that set to have occurred in any of the N states.

$$b'_i(k) = \frac{\sum_{t=1}^T \sum_{s.t. O_t = v_k} \alpha_t (I) \left[\gamma_{tt'}(i,j) / \sum P(d = t' - t | i) \right] \beta_t(i)}{\sum_{t=1}^T \alpha_t (I) \left[\gamma_{tt'}(i,j) / \sum P(d = t' - t | I) \right] \beta_t(i)}$$

Using the above equations, the model parameters are calculated for every iteration. After the computations, we calculate compare the newly computed values to the previous values and record the change. If the change is below a defined threshold, we stop the training process. This is how an optimized model is designed for the given training data.

The complexity of the Baum – Welch algorithm is $O(N^2T)$. This technique is far more complex than the supervised MLE training algorithm.

CHAPTER 3

IMPLEMENTATIONAL DETAILS

3.1 Changes from the Traditional HMM

As in its theory, the implementation of a semi – Markov has a few major differences when compared to the traditional hidden markov model (HMM). The implementation of a simple HMM takes into consideration the case of discrete input; meaning that every state of the model emits only a single observation. Also, since the model is discrete the duration of being in a single state 'i' was always one. The hidden semi – Markov model takes in continuous input and thus we have to make the following critical changes:

- The model file should accommodate a sequence of characters for each state rather than a single emission symbol as before.
- We should introduce a new parameter – the state duration probabilities, which store the probability of a state's duration as a discrete random variable [13].
- There would be a significant change in the approach taken in the application of every algorithm in contrast to a regular HMM.

To explain more precisely, since the states in HsMM take a continuous input, the transition from a state 'i' to itself takes place for a certain interval before the system transfers into another state 'j'. Therefore, the code written should accommodate this condition in its

entirety. In the Figure 9, we can vividly see the concept [11].

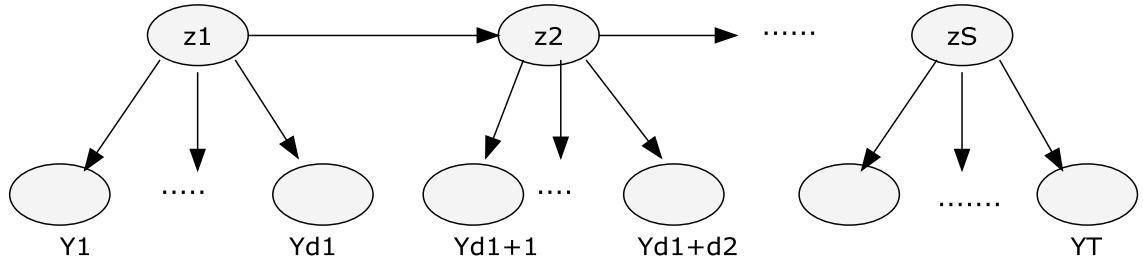


Figure 9: Duration of Each State in a HsMM

The above figure has the set of states $S = \{ Z_1, Z_2 \dots, Z_s \}$. Every state has a set of observations for certain duration, i.e. length. For example, Z_1 has the observation sequence of length D_1 and the state Z_2 has the sequence of length D_2 . A transition of state takes place only after it runs for its give duration. This paper will elaborate the complete implementation details of the structure of HsMM along with its associated algorithms in section 3.2.

3.2 Algorithm Implementation Details

The implementation code has been written entirely in C++ programming language. The implementation is divided into three files [12]. The first file being the **Hmm.hpp** is the header file where the definition of the 'Model' class is written. The second file is the **main.cpp**, which as the name says is from where the execution of the program starts and runs.

Hmm.cpp is the program file containing the implementations of all the algorithms and procedures required for the complete functionality of the HsMM model. The project folder needs various other supporting files for its execution. These are:

- **Model Files:** Model files are the input files given to the HsMM system that have the complete details of all the HsMM parameters. In our implementation, the model file has four parameter fields to read; the initial probabilities (denoted as InitPr), the output probabilities (denoted as OutputPr), the transitional probabilities (denoted as TransPr) and the state duration probabilities (denoted as StateDr). A sample model file is given below.

```
SAMPLE MODEL FILE FOR HsMM

2
InitPr 2
0 0.5
1 0.5
OutputPr 12
0 ab 0.3
0 bc 0.3
0 cac 0.3
0 dee 0.033
0 ef e0.033
0 fdf 0.033
1 ab 0.033
1 bcb 0.033
1 ca 0.033
1 de 0.3
```

```

1 eff 0.3
1 fd 0.3
TransPr 4
0 0 0.5
0 1 0.5
1 0 0.5
1 1 0.5
StateDr 2
0 0.2 0.2 0.2 0.2 0.2
1 0.2 0.2 0.2 0.2 0.2

```

Figure 10: Sample Model File for HsMM

The model file first reads the number of states in the model (here it is 2). Then the model file contains the fields for all the model parameters in an order. On careful observation, we will see that, the output probabilities for a state (either 0 or 1) is a sequence of characters and also the state duration probabilities are presented in the model file.

- **Training Files:** The training files provide the data sets to train the model in order to optimize its parameters. We have implemented both the supervised training technique of Maximum Likelihood Estimation (MLE) and the unsupervised training mechanism of Baum – Welch. Consequently, we use a 'tagged training data' to train the model using MLE and an 'untagged data' when using the Baum – Welch. In Figure 11 we give an example tagged training sample. When using the Baum – Welch algorithm, we only give the training data without the transition

between states. Figure 12 shows a sample of training data when we are using the unsupervised training.

TAGGED TRAINING DATA FOR HsMM	
abcd	0001
abedd	00111
ab	00
bd	01
aee	011
aedf	0111
faba	1000
cabf	0110
ffbc	11100
bace	0001
abaeb	00010
eb	10
ede	111
ddfa	1110
cache	00001
fb	10

Figure 11: Tagged Training Data Set for an HsMM

```
UNTAGGED TRAINING DATA FOR HsMM

abcd 0001
abedd 00111
ab 00
bd 01
aee 011
aedf 0111
faba 1000
cabf 0110
ffbc 11100
bace 0001
abaeb 00010
eb 10
ede 111
```

Figure 12: Untagged Training Data Set for an HsMM

- **Result Files:** These are the files that we write the result of a decode function (Viterbi) into. Hence, this file will contain each emitted symbol with its corresponding most probable state.

Now we will proceed to give the full details of all the source code files, i.e `hmm.hpp`, `main.cpp` and `hmm.cpp`. We will first start off with the header file.

- **Hmm.hpp:** This is the header file which is included in all other source

code files. In the header file we define the class for the HsMM model.

This class of HsMM has the declarations of various methods which are used and also of the model parameters. The model parameters are declared as dynamic array vectors. Along with these, we also define various variables that are used in different algorithms as dynamic vectors. Below we show the declaration of the model parameters.

```
double *I; // initial state probability  
double **A; // state transition probability  
double **B; // output probability  
double **Pd; // state duration probability
```

➤ **main.cpp:** As in any project module, main.cpp is the file where the execution of the program is handled. Our main function first declares an object for our Model, the model file and the sequence file. It determines the mode of execution depending on the options given by user. Our code gives the option for the user to work in three modes [12]. Those are:

i) Decode option using **-d**: This enables the Viterbi algorithm. It is used by the user to find the best possible state sequence for a given set of observations. This mode requires a model file and a sequence file as inputs and generates a result file as its output.

Example: % `hmm -d -m modeldemo -s sampleseqdemo > sampletag`

In the above example, in addition to `-d`, `-m` is used to specify the compiler that a model file (modeldemo) will be given next. Similarly, `-s` indicates that the next input is a sequence file (sampleseqdemo). The

output file (sampletag) is created using redirection.

ii) Supervised training using **-c**: This makes use of the MLE algorithm and trains the HsMM model file with the given tagged sequence file.

Example: `hmm -c -n 4 -m modeldemo -s taggedseqdemo`

The above example makes use of `taggedseqdemo` to train the `modeldemo` file. The `-n` option specifies the number to states to be used in the model. In this case, it is four.

iii) Unsupervised training using **-t**: The Baum – Welch algorithm is used to train the given input model file with untagged training data sets.

Examples: `hmm -t -n 2 -m modeldemounsup1 -s seqdemo2`

In the above example, we give the `seqdemo2` as the untagged data to train the `modeldemounsup1` by making use of Baum – Welch procedure.

We present a part of the `main.cpp` file where it can be seen how the program identifies which operation to undertake by comparing the options given.

```
if (!strcmp("-train",argv[i],2)) {  
    action = 1;          // Calls the Baum – Welch function  
} else if (!strcmp("-decode",argv[i],2)) {  
    action = 2;          // Calls the Decode (Viterbi) function  
} else if (!strcmp("-count",argv[i],2)) {  
    action = 3;          // Calls the Count (MLE) function  
}
```

- **HMM.cpp:** This is the file where all the source code related to the implementation of various algorithms is written. Depending on the function calls made from the main.cpp, the program is executed. This program first has the duty of allocating the memory space for all the dynamic variables being used. Then it assigns the values given in the model file to its respective model parameters. It makes use of two constructors for the above operations. We use the concept of smoothing while loading the input values. This technique will be spoken about clearly in section 3.3. Once the model has been established, we can proceed on running algorithms on it.

Different methods are called upon for different modes. Let us see in detail the execution of every mode.

- **DECODE MODE:** The decode mode when called up, first takes up the model file and creates all the required variables. Then it calls up the decode function. As explained before we use the concept of a trellis diagram to make the code simpler. For this, we create a structure called 'TrellisNode' which contains all the information related to a single Viterbi iteration [12]. The function first takes in the sequence file and reads its first symbol. It calculates the probability of generating the symbol 'c' from state 'i'. We make use of a 'while' loop for the remaining symbols. The code in this loop is used to grow the best path for each state. Let us assume that the sequence already processed is $\langle c_1, c_2, \dots, c_k \rangle$. For every newly processed symbol c , we will update each of the trellis node i.e.,

trellis[i], and it will hold the state transition path ending at state i that most likely forms the sequence $\langle c_1, \dots, c_k, c \rangle$. Similarly, trellis[i].pr has the updated log probability of generating the data, given the path. Below is a small part of the code that stores the probability of generating a sequence in 'thisPr' when the system follows the path followed in 'trellis[j].path.

```

thisPr = trellis[j].pr + log10(A[j][i]) + log10(B[cIndex][i]) ;
for ( k=0; k<D; k++){
    thisPr += log10(Pd[i][k]);
    if (j==0 || thisPr > bestPr) {
        bestFromState = j;
        bestPr = thisPr;
    }
}

```

After generating all the possible paths for a given sequence of symbols, the next task is to pick the best path among them. Thus, we have to do a comparison between the generated state sequences to find out the most probable path. Below we give the relevant code that performs this operation.

```

vector<int> bestPath;
bestPath.clear();
int bestTrellis;
double bestTrellisPr = 0;
for (int i = 0; i<N; i++){

```



```

if (i==0 || (trellis[i].pr > bestTrellisPr)) {
    bestTrellis =i;
    bestTrellisPr = trellis[i].pr;
}

```

Finally, the best path is stored in the variable 'bestTrellis' and is written into the result file.

- COUNT MODE: The count mode is used for supervised training. Similar to any mode, the model file is read in and the parameters are initialized. We use counters for each parameter in order to keep a count and then use the MLE formulas to calculate the new distributions. We first use the 'ResetCounters()' function to set everything to zeros. Then, the method 'count()' takes the given sequence file and proceeds to compute the model values. After reading the first symbol and the corresponding state, we update the starting state counter, i.e Icounter to reflect that the program read the state s as a starting state and appropriately make changes in the emission symbol counter; Bcounter for that symbol.

```

if (first) {
    ICounter[s] +=1;
    INorm += 1;
    first = false;
}

```

The above code checks if the symbol read is the first in the sequence. If so, it updates the counters Icounter and INorm. The transition probabilities are calculated from the second symbol. We check if a

previous state exists for the current state 's' and if it does we update the transition counters (Acounter and ANorm) accordingly. The following code implements this functionality.

```
if (prevState>=0)
{
    previous state.
    ACounter[prevState][s] +=1;
    ANorm[prevState] +=1;
}
```

The given sequence file is read and the counts are calculated for every emitted symbol. 'UpdateParameter()' method is used to perform the MLE calculations on the accumulated counts. Consequently, for each of the 'N' states this function calculates the model parameters. Using the 'Save()' function we write back the updated model parameters onto the file [12].

- TRAINING MODE: The user intends to use the Baum – Welch algorithm to train the model using untagged data sets. After taking in the model file and allocating the memory, we call the 'Train()' function with the given sequence file as input. Baum – Welch algorithm first randomly assigns some values for the model parameters using a random generator(RandomInit). Let us call the iteration of the training process as an epoch. After every epoch, we check if our HsMM has reached a stable

configuration. We define the history of previous epochs in a variable 'meanFit' and the variable 'currentFit' has the current parameters. If the difference between these two variables is negligible, we stop the training. The training process involves calculations of forward and backward variables among many other things. The paper clearly describes the implementation of these procedures and also identifies the changes made from the Traditional HMM.

'ComputeAlpha()' method computes the forward variable for a given sequence. Since we are dealing with a Hidden semi-Markov model with continuous input the calculations involve increased complexity. We use the normalizing array 'eta[]' to compute the normalized values for alpha. The method starts off by calculating the alpha when time $t=1$. At every step of calculation, we need to consider an extra variable, the state duration probability which was unheard of in the regular HMM. Hence, we can see in the following piece of code that we perform calculations using all the three parameters; A, B and Pd.

```

for (i= 0; i<N; i++)
{
    alpha[t][i] = 0;
    for (j=0; j<N; j++)
    {
        alpha[t][i] +=alpha[t-1][j]*A[j][i] ;
    }
}

```

```

for (k=0;k<D;k++)
    temp1[seq[t]][i]+=B[seq[t]][i]*Pd[i][k];
alpha[t][i]*=temp1[seq[t]][i];
eta[t] += alpha[t][i];
}

```

We make use of a temporary dynamic variable 'temp1' to do the computations involving the duration variable (Pd). Similarly, we proceed to calculate the backward variable using the method 'ComputeBeta()'. As in the case of a HMM, at time 'T', meaning at the end of sequence the value is set to one. As we start moving down the sequence, the calculation now has to consider the duration probabilities similar to the forward variable. We compute this by following the formulas given before in section 3.4.

```

for (i= 0; i<N; i++)
{
    beta[t][i] = 0;
    for (j=0; j<N; j++) {
        temp2[t+1][j]=0;
        for(k=0;k<D;k++)
            temp2[t+1][j]+= Pd[j][k]*B[seq[t+1]][j]*beta[t+1][j];
        beta[t][i] +=A[i][j]*temp2[t+1][j] ;
    }
}

```

A variable 'temp2' is used to make the computation easy. These temporary values are de-allocated after their use. The next step is to calculate the value of gamma variable which as described in section 2.7 helps in the calculations of the model parameters. The 'AccumulateCounts()' method ciphers these variables. The calculation of the gamma variable is similar to a traditional HMM. The code computes for every state and instances the product of forward and backward variables and normalizes them. While calculating the modified HsMM parameters, we make use of the variable 'countInc' which is the variable $\xi_{t(i,j)}$ described in section 2.7. This involves intricate computation as we show in the snippet below.

```

for (i=0; i<N; i++)
{
    for (j=0; j<N; j++) {
        temp3[t+1][j] =0;
        for(k=0;k<D;k++)
            temp3[t+1][j]+=Pd[j][k]*B[seq[t+1]][j]*beta[t+1][j];
        countInc =(gamma[t][i]*A[i][j]*eta[t+1]*temp3[t+1][j] )/ beta[t][i];
        ACounter[i][j] += countInc;
        ANorm[i] += countInc;
    }
}

```

The countInc in the above code is used to compute the transition

probability $A[i][j]$ for a particular instant. Similarly we follow the formulas listed before to compute the remaining model parameters. We pass on these calculations to the 'UpdateParameter()' method similar to the MLE method. Finally, the improved model parameters are written back onto a model file using the 'Save()' method.

This section clearly described the implementation changes of every algorithm and in the process gave out the differences in application of HsMM concepts with those of an HMM. The next section describes the concept of scaling and the various scaling mechanisms used in this implementation.

3.3 Smoothing

Smoothing is a technique used to flatten a given probability distribution function, so that every sequence could occur with some probability. The essence of any smoothing procedure involves redistribution of weights from high probability regions to the zero probability regions [17]. Smoothing is often required, especially when dealing with language models. For example, in a speech recognition problem, due to lack of a perfect data set this may represent certain words almost inconceivably. In such cases, we encounter zero probabilities which hamper the working of the model [18].

In our Hidden semi-Markov model, we might have a few cases of zero

probabilities if we are given a sparse training data. This can cause a set of poor probability estimates, meaning there might be a set of unseen sequences accounting for zero emission probabilities. The simplest smoothing technique (also called, Laplace Smoothing) works by just assuming that each bigram appears exactly one more than it actually did [17]. Hence, the new probability for the bigram is calculated as,

$$P(w_i | w_{i-1}) = \frac{1 + C(w_{i-1}, w_i)}{\sum (1 + C(w_{i-1}, w_i))} = \frac{1 + C(w_{i-1}, w_i)}{V + \sum C(w_{i-1}, w_i)}$$

The implementation of HsMM makes use of two smoothing techniques. These are relatively simple when compared to the other complicated smoothing mechanisms. These are: Absolute Discounting and the usage of Logs. We will now describe them in detail.

- **Absolute Discounting:** The idea of this is to reduce a fixed discount D from non-zero probabilities, and redistributing them among unobserved events [19]. This technique is simple and quite effective. After the model allocates memory for the model parameters, we use this technique while taking in the input values and initializing them to their respective variables.

From the non-zero occurrences, we deduct a fixed amount D :

$$P_{\text{new}}(x) = (C(x) - D) / C, \text{ if } C(x) > 0$$

Let there be M zero occurrences, hence we have a total deducted

amount of $D(N-M)$, where N is the total number of event occurrences. We distribute this amount to the M zero instances as below,

Each zero occurrence will get a value: $D(N-M)/M$.

$$P_{\text{new}}(x) = D(N-M) / MC, \text{ if } C(x) = 0$$

The source code implements this technique after reading the parameter values from the input file. Thus, making sure the algorithms always work with non-zero probabilities.

- **Using Logarithms:** We make sure that none of the algorithms processes zero input probabilities but we also have to make sure that, the procedures do not produce zero probabilities from their calculations. Since every algorithm involves multiple iterations, the program runs into cases where diminutive values of probabilities are multiplied more than once resulting in infinitesimally small values. To avoid this, we use the concept of logarithms which replaces the concept of multiplication with arithmetic summation.

For example, in the 'Decode()' function we use logarithms to the base 10 while calculating the probability of a trellis path occurring for the given emission sequence.

$$trellis[i].pr = \log_{10}(I[i]) + \log_{10}(B[cIndex][i]) + \log_{10}(Pd[i][1]);$$

This is employed in all the other algorithms, ensuring that we eliminate any case of zero probabilities occurring in the hidden semi-Markov model.

This chapter clearly described the implementation techniques involved in modeling a Hidden semi-Markov model and its primary dissimilarity to the common HMM. The paper will now describe the working results of this model over a given set of input values.

CHAPTER 4

RESULTS EVALUATION

This chapter mainly discusses and illustrates the results obtained for the various algorithms after the implementation of the same in chapter 3. We will also evaluate the results obtained, compare them with the results of a traditional HMM and try to list out a few properties based on the observations.

4.1 Results

As described before, our implementation of Hidden semi-Markov model works in three modes. The mode for decoding the state sequence for a given set of observation symbols, a mode which allows the program to train the model file based on supervised data and the final mode which works using the Baum – Welch algorithm to train the system without tagged sequences.

We will test our implementation by giving the specified inputs needed for every mode and comparing the proximity of the results obtained. The implementation is written in C++ and hence runs on a simple g++ compiler with standard set of libraries. We use the command “gmake” to compile the code before running it.

The below screen shot illustrates the directory of files that comprise our implementation and then proceeds to show the compilation of our

make file. This step creates an object file (named hmm) which will be used for running the program. Let us now proceed to describing the execution of every mode and its corresponding results.

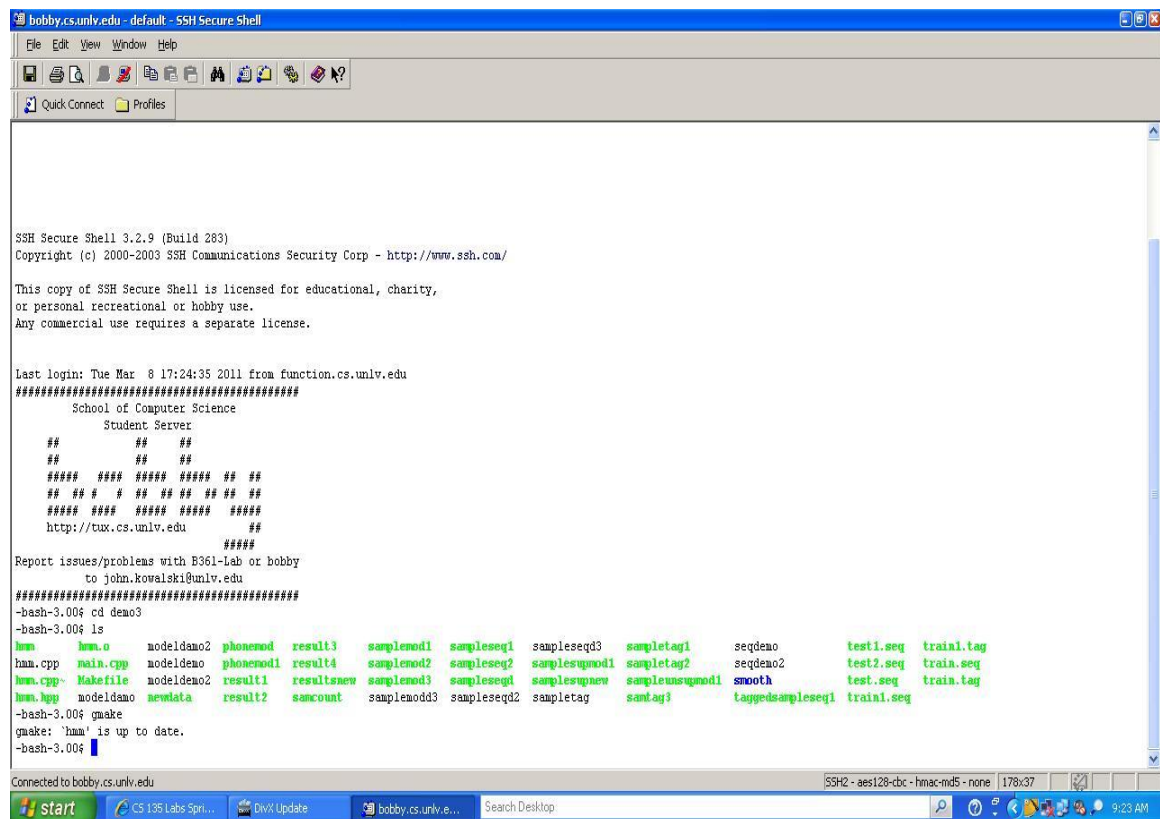


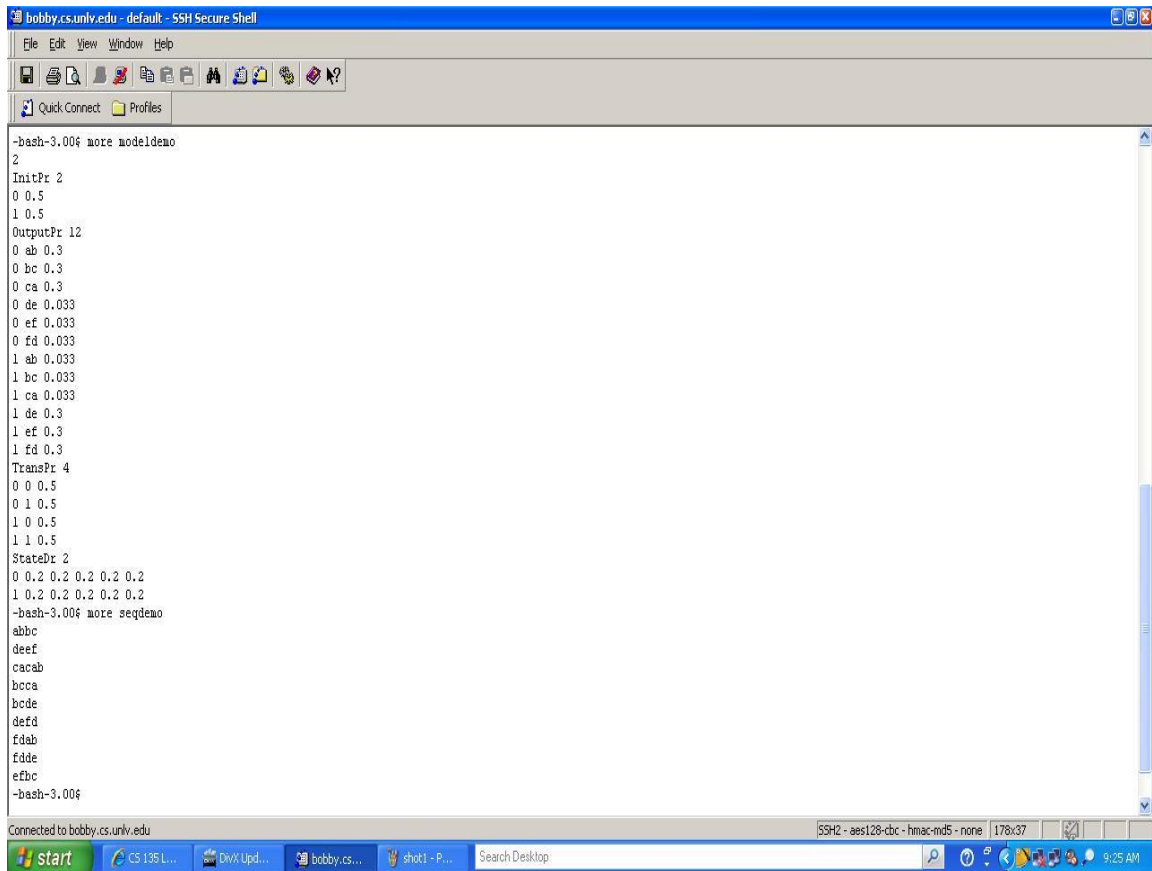
Figure 13: Screenshot of the Project Directory and its Compilation

- **Decode – Viterbi Algorithm:**

The function ' decode ()' is called upon with the inputs being the model file and a sequence file. After running the program we will generate a result file which tags every symbol to its state [12]. The command issued for execution is:

```
“./hmm -d -m /home/nagendra/demo3/modeldemo -s  
/home/nagendra/demo3/seqdemo>/home/nagendra/demo3/sampletag”
```

The above command takes in the model file (modeldemo) and the sequence file (seqdemo) and generates a result file named 'sampletag'. Below we give a screenshot showing the above mentioned input files.

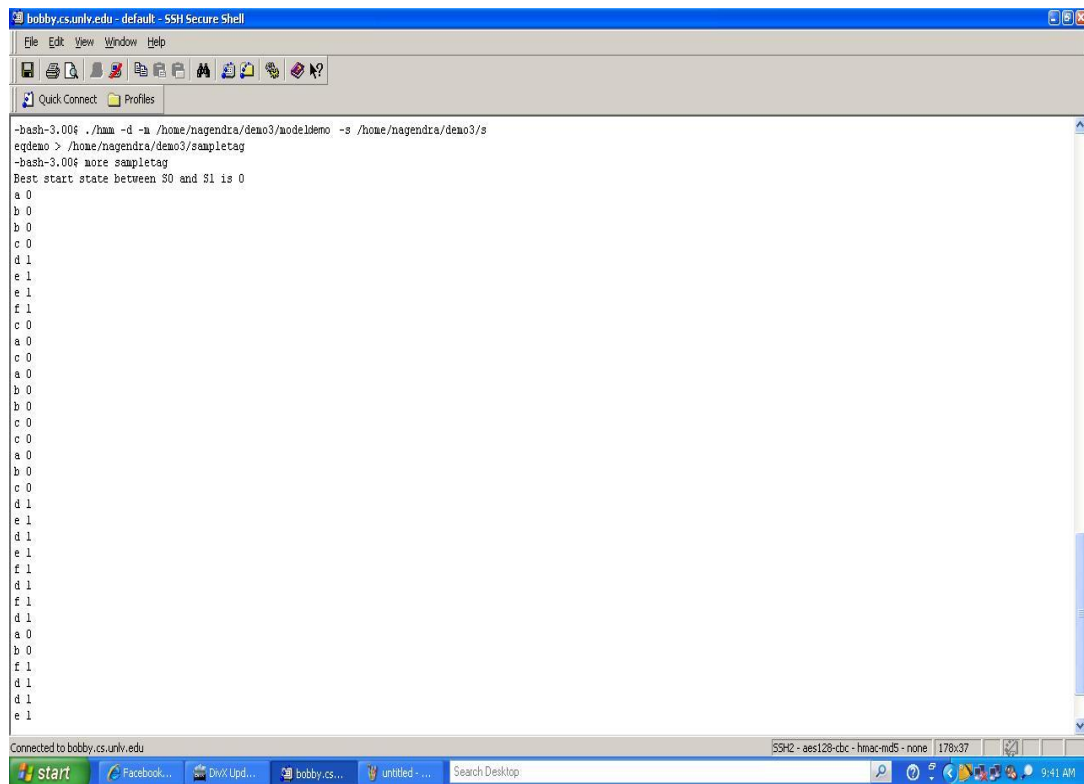


```
bobby.cs.univ.edu - default - SSH Secure Shell  
File Edit View Window Help  
Quick Connect Profiles  
-bash-3.00$ more modeldemo  
2  
InitPr 2  
0 0.5  
1 0.5  
OutputPr 12  
0 ab 0.3  
0 bc 0.3  
0 ca 0.3  
0 de 0.033  
0 ef 0.033  
0 fd 0.033  
1 ab 0.033  
1 bc 0.033  
1 ca 0.033  
1 de 0.3  
1 ef 0.3  
1 fd 0.3  
TransPr 4  
0 0 0.5  
0 1 0.5  
1 0 0.5  
1 1 0.5  
StatePr 2  
0 0.2 0.2 0.2 0.2 0.2  
1 0.2 0.2 0.2 0.2 0.2  
-bash-3.00$ more seqdemo  
abbc  
deef  
cacab  
bccca  
bode  
defd  
fdab  
fdde  
eEbc  
-bash-3.00$
```

Figure 14: Screenshot Showing the Input Files for Decode Algorithm

The model file used in this example gives high probability of occurrence to the symbols {a, b, c} for state '0' and the symbol set {d, e, f} for the state '1'. The model file also specifies the state duration and other

parameters required to complete as HsMM. On executing the decode command, we check the generated result file by using the 'more' command. Figure 15 illustrates these steps. It clearly elucidates that the 'sampletag' contains the symbols {a, b, c} tagged to state '0' and {d, e, f} tagged to state '1' respectively. To make sure the implementation is correct; we tested it using another set of inputs. Figure 16 displays the 'modeldemo2' model file and the 'seqdemo2' sequence file. We get accurate results by running our code on these inputs. The results are stated in Figure 17.



```
bobby.cs.unlv.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

--bash-3.006 ./hmm -d -a /home/nagendra/demo3/modeldemo -s /home/nagendra/demo3/s
eqdemo > /home/nagendra/demo3/sampletag
--bash-3.006 more sampletag
Best start state between S0 and S1 is 0
a 0
b 0
b 0
c 0
d 1
e 1
e 1
f 1
c 0
a 0
c 0
a 0
b 0
b 0
c 0
c 0
a 0
b 0
c 0
d 1
d 1
e 1
d 1
e 1
f 1
d 1
f 1
d 1
a 0
b 0
f 1
d 1
d 1
e 1
```

Figure 15: Execution of Decode Command and the Resulting File

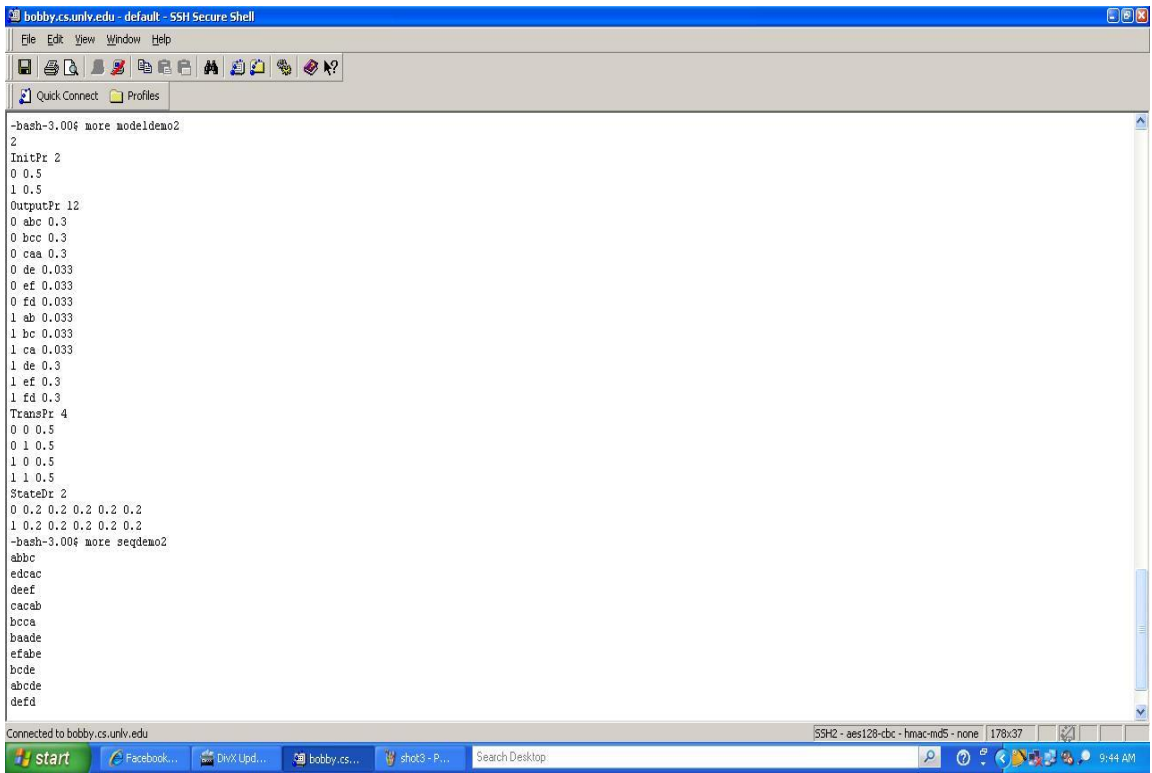


Figure 16: Alternate Input Files for the Decode Command

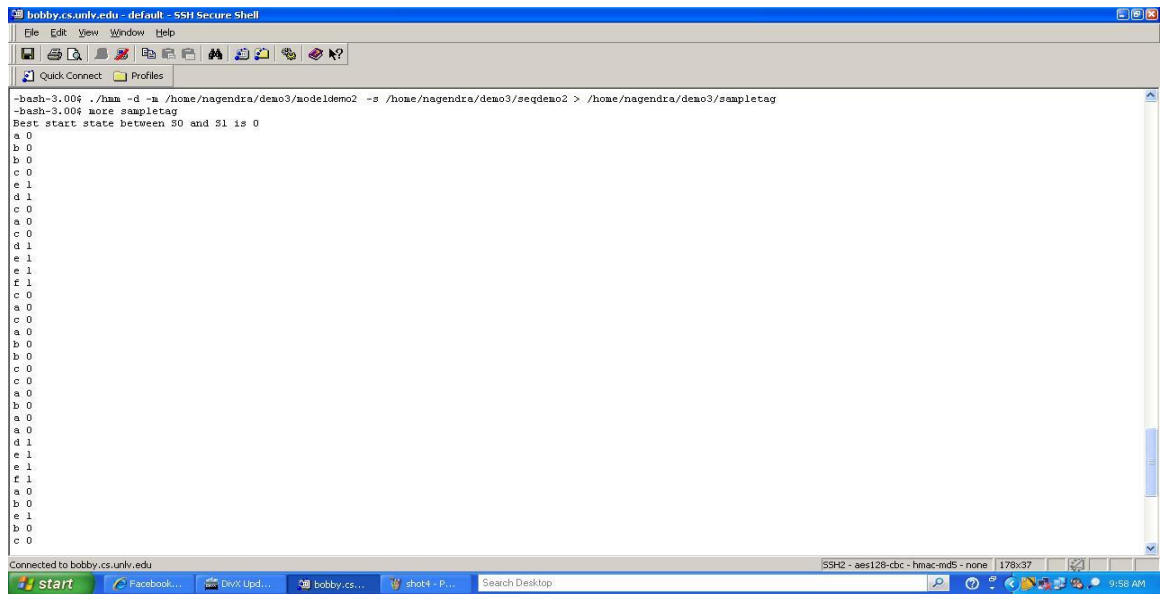


Figure 17: Execution and the Result File for Alternate Inputs

The above screen shot depicts the 'sampletag' file. On close examination, we can see that the decode function correctly assigns the state to its symbol in accordance to the given model file. The paper now proceeds to show the execution of the other two modes.

- **Count – Maximum Likelihood Estimation Algorithm:**

The 'count ()' algorithm is used to estimate the parameters of the model by making use of the given tagged sequence of symbols. The input includes the number of states, a model file name and a tagged sequence file [12]. The following command is used to execute the count mode.

```
“./hmm -c -n 2 -m /home/nagendra/demo3/supmoddemo -s  
/home/nagendra/demo3/demotagseq “
```

Since we are using smoothing throughout our implementation, the generated model parameters are non-zero. Figure 18 displays the execution of the above command. We can check the model file generated using the 'more' command. This is also shown in the figure below.

```

1:bobby.cs.unlv.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/
This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.

Last login: Fri Mar 11 09:22:46 2011 from function.cs.unlv.edu
#####
      School of Computer Science
      Student Server
      ##      ##      ##
      ##      ##      ##
      #####  #####  #####  ##  ##
      ## # # # # # # # # # #
      #####  #####  #####  #####
      http://tux.cs.unlv.edu      ##
#####
Report issues/problems with B361-Lab or bobby
to john.kovalski@unlv.edu
#####
-bash-3.00$ clear
-bash-3.00$ /ham -c -n 2 -n /home/nagendra/demo3/supaoddemo -s /home/nagendra/demo3/demotagseq
-bash: ./ham: No such file or directory
-bash-3.00$ clear
-bash-3.00$ cd demo3
-bash-3.00$ make lma
make: 'ham' is up to date.
-bash-3.00$ /ham -c -n 2 -n /home/nagendra/demo3/supaoddemo -s /home/nagendra/demo3/demotagseq
-bash-3.00$ more supaoddemo
2
InitPr 2
0 0.99999
1 0.9998e-06
OutputPr 188
0 !# 1.2498e-06
0 * 1.2498e-06
0 + 1.2498e-06
0 % 1.2498e-06
0 & 1.2498e-06
0 ^ 1.2498e-06
0 | 1.2498e-06
0 ~ 1.2498e-06
0 ` 1.2498e-06
0 { 1.2498e-06
0 } 1.2498e-06
0 ~ 1.2498e-06
0 ! 1.2498e-06
#####
Connected to bobby.cs.unlv.edu
#####
start 1:bobby.cs.unlv.edu ... 2:bobby.cs.unlv.edu ...
SSH2 - aes128-cbc - hmac-md5 - none 178x42 NUM
12:03 PM

```

Figure 18: Execution of the Count Procedure

The given sequence file is structured such that the maximum output probabilities are to be assigned for the string 'ab' in state '0' and for the string 'cd' in the HsMM state '1'. Figure 19 shows the state '0' with the output 'ab' having a probability of 0.86 and the remaining output sequences with a negligible probability.

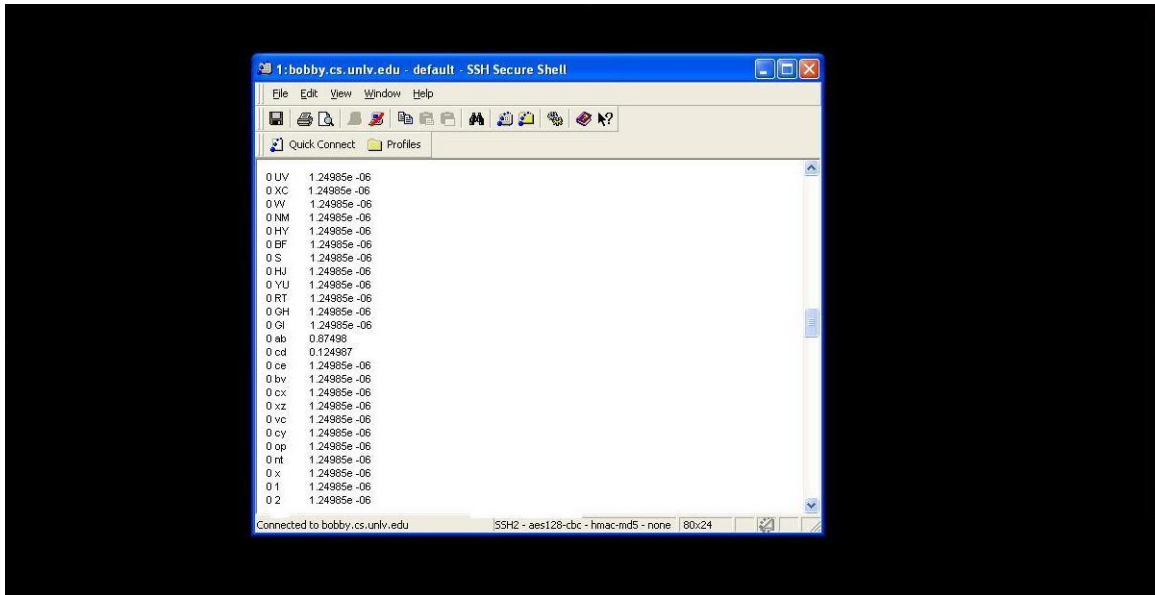
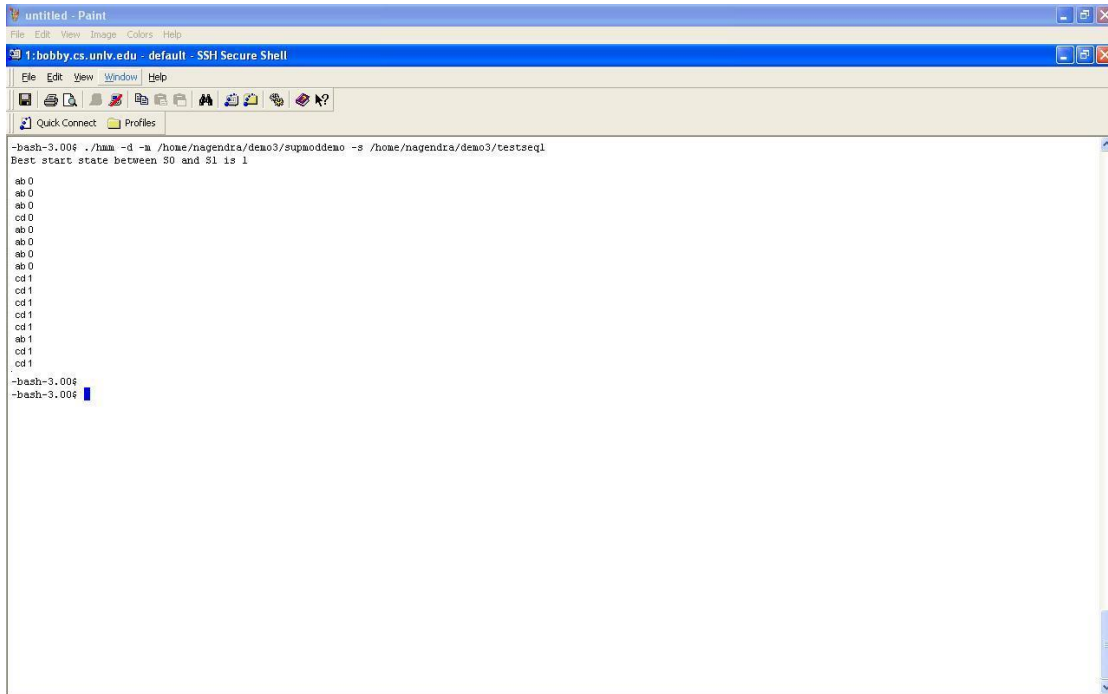


Figure 19: Resulting Model File After Training Using Count

To test the estimated model, we use a sequence file named 'testseq1' which has eight occurrences each of 'ab' and 'cd'. If the implementation is right and the model generated using the MLE algorithm is accurate, the Viterbi algorithm will generate a result file mostly tagging 'ab' to state '0' and 'cd' to state '1'. Figure 20 illustrates this fact. The command given below is used to run the Viterbi algorithm on the 'testseq1' file.

`“./hmm -d -m /home/nagendra/demo3/supmoddemo -s /home/nagendra/demo3/testseq1”`



```
1:bobby.cs.unlv.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
-bash-3.00$ ./ham -d -a /home/nagendra/demo3/supmoddemo -s /home/nagendra/demo3/testseq1
Best start state between S0 and S1 is 1
ab 0
ab 0
ab 0
cd 0
ab 0
ab 0
ab 0
ab 0
cd 1
cd 1
cd 1
cd 1
ab 1
cd 1
cd 1
-bash-3.00$
-bash-3.00$
```

Figure 20: Result File on Using Viterbi on the Trained Model Using Count

Apart from a couple of exceptions, the model file perfectly tags the symbols to its states accordingly. The paper now proceeds to test the unsupervised training technique.

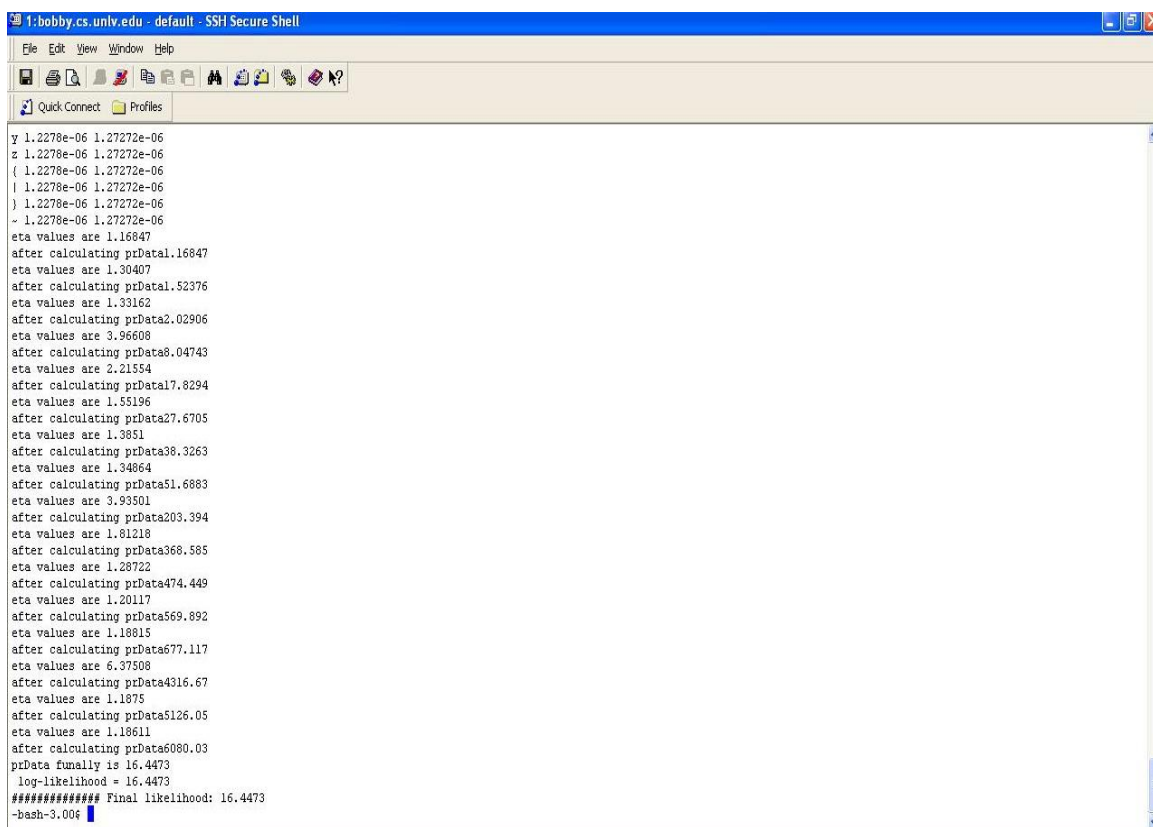
- **Train – Baum-Welch Algorithm:**

The third and the most complex of the three is the Baum – Welch algorithm. The 'train()' function is called upon in this mode. Unlike any of the other algorithms, this mode runs the training routine for an unspecified number of times. Until the model file has reached a state where no further improvement can be made, the training continues. The input includes the number of states, the model file and a raw sequence

file [12]. The following command is used to train the model file:

```
“./hmm -t -n 2 -m /home/nagendra/demo3/unsupmoddemo -s  
/home/nagendra/demo3/testseq1”
```

In the above case, the function went on training for hundred and four times before reaching a state of a stable likelihood. Figure 21 shows the screen shot of the final epoch with its likelihood.

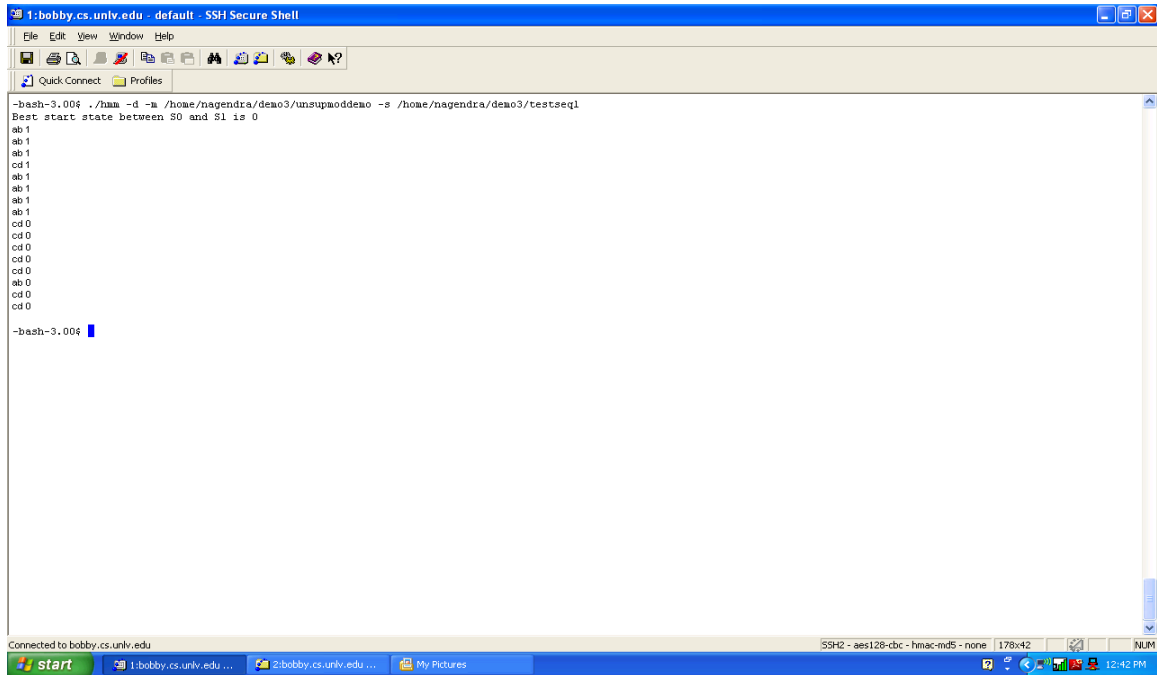


```
1: bobby.cs.unlv.edu - default - SSH Secure Shell  
File Edit View Window Help  
Quick Connect Profiles  
y 1.2278e-06 1.27272e-06  
z 1.2278e-06 1.27272e-06  
{ 1.2278e-06 1.27272e-06  
| 1.2278e-06 1.27272e-06  
} 1.2278e-06 1.27272e-06  
~ 1.2278e-06 1.27272e-06  
eta values are 1.16847  
after calculating prData1.16847  
eta values are 1.30407  
after calculating prData1.52376  
eta values are 1.33162  
after calculating prData2.02906  
eta values are 3.96608  
after calculating prData8.04743  
eta values are 2.21554  
after calculating prData17.8294  
eta values are 1.55196  
after calculating prData27.6705  
eta values are 1.3851  
after calculating prData38.3263  
eta values are 1.34864  
after calculating prData51.6883  
eta values are 3.93501  
after calculating prData203.394  
eta values are 1.81218  
after calculating prData368.585  
eta values are 1.28722  
after calculating prData474.449  
eta values are 1.20117  
after calculating prData569.892  
eta values are 1.18815  
after calculating prData777.117  
eta values are 6.37508  
after calculating prData4316.67  
eta values are 1.1875  
after calculating prData5126.05  
eta values are 1.18611  
after calculating prData6080.03  
prData finally is 16.4473  
log-likelihood = 16.4473  
##### Final likelihood: 16.4473  
-bash-3.006
```

Figure 21: The Final Likelihood Using Baum Welch Algorithm

Since we are using an unsupervised mechanism there is a possibility that the states might have their most likely emission symbols interchanged [12]. Similar to the Count mode, this model file is also tested on the same sequence file; 'testseq1'. The results of this operation

are shown in Figure 22.



```
1:bobby.cs.unlv.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
-bash-3.00$ ./ham -d -m /home/nagendra/demo3/unsupmoddemo -s /home/nagendra/demo3/testseq1
Best start state between S0 and S1 is 0
ab 1
ab 1
ab 1
cd 1
ab 1
ab 1
ab 1
ab 1
cd 0
cd 0
cd 0
cd 0
cd 0
ab 0
cd 0
cd 0
-bash-3.00$
```

Figure 22: Result File on the Trained Model Using Baum Welch

As suspected before, the states do change their most favored output sequence of symbols. Nevertheless, the model is consistent and correctly assigns the states most of the times.

We have described acutely the execution of the different modes of the implementation along with their corresponding outputs.

4.2 Advantages Over the Traditional HMM

All the modes of operation have one common distinct feature compared to a regular HMM and that being the fact that, not only the Hidden semi-Markov model work with a discrete output but it works

equally good with any combination of a continuous output sequence. This is a clear improvement over a traditional HMM. This results in the following advantages:

- The concept of continuous chain of symbols extends the usability to many other domains.
- The scalability of the model is increased and the dynamic processing of data is easier [20].
- The recognition phase implemented for an HsMM when implemented at a segment level shows comparatively better results [5].
- The use of Gaussian distribution for the symbol probability distribution at the state level is a defining aspect of feature selection [5].
- In domains such as handwriting recognition, it is always a good idea to implement a variable duration model for the Hidden markov mode [5]. There is a given built-in dubiousness involved with the handwritten characters.

Hence, because of the above benefits, HsMM is being exploited extensively over the past decade.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

This thesis “Implementation of Hidden Semi-Markov Models” studies the technique of continuous markov chain. We have implemented the comprehensive HsMM model comprising of all the algorithms needed to solve its three basic problems.

We discussed the background of the semi-Markov model where the advantages over the traditional HMM were given and the general structure along with its issues was explained in Chapter 1. Chapter 2 put forth the details of the modified model parameters along with the theoretical changes made to the commonly used procedures used for problem solving. All the implementations on the algorithms along with the smoothing mechanisms employed were explained in Chapter 3. All the results were presented clearly and compared to the common HMM in Chapter 4.

The thesis mainly dealt with comparing the semi-Markov hidden model with the regular hidden Markov model. The theoretical differences were laid out and analyzed. The paper also describes the advantages of using the continuous model against the discrete model and to support it gives the list of active applications in this domain.

From the results obtained in Chapter 4, we can evaluate that this model of working is advantageous in many ways. The additional features

offered by the HsMM go a long way in real time applications providing a certain edge over other methodologies. Though HsMM is expedient in many cases, it does have a few disadvantages of its own. Those are mainly related to the computational complexities.

Continuous Markov Models is an active field of study in the area of information extraction and machine learning. In future, this study can be extended by implementing this model on larger and suitable data sets. A large variety of tools can be modeled keeping this as the basic structure. Some of the easier applications that can be developed are an acronym finder or a handwriting recognition system among many others.

BIBLIOGRAPHY

1. Shun-Zheng Yu, 'Hidden semi-Markov models', Department of Electronics and Communication Engineering, Sun Yat-Sen University, Guangzhou 510275, PR China, 2009
2. Jan Bulla, 'Application of Hidden Markov Models and Hidden Semi-Markov Models to Financial Time Series', Dissertation Presented for the Degree of Doctor of Philosophy at the Georg-August-University of Göttingen, 2006
<http://mpra.ub.uni-muenchen.de/7675>
3. Ming Dong, David He, 'Hidden semi-Markov model-based methodology for multi-sensor equipment health diagnosis and prognosis', European Journal of Operational Research, 2006.
4. Kevin P. Murphy, 'Hidden semi-Markov models (HSMMs)', 2002
www.ai.mit.edu/~murphyk
5. Mou-Yen Chen, Amlan Kundu, Sargur N. Srihari, 'Variable Duration Hidden Markov Model and Morphological Segmentation for Handwritten Word Recognition', IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 4, NO. 12, DECEMBER , 1995
ieeexplore.ieee.org/iel2/916/7982/00341066.pdf
6. ChengXiang Zhai, 'A Brief Note on the Hidden Markov Models (HMMs)', Department of Computer Science University of Illinois at Urbana-Champaign, 2003
<http://sifaka.cs.uiuc.edu/course/498cxz05f/hmm.pdf>

7. Lawrence R. Rabiner, 'A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition', Pages: 257-286, Proceedings of the IEEE, 1989
<http://www.cs.ubc.ca/~murphyk/Bayes/rabiner.pdf>
8. Barbara Resch, 'Hidden Markov Models, A Tutorial for the Course Computational Intelligence', Signal Processing and Speech Communication Laboratory, Institute for Theoretical Computer Science, Institute for Theoretical Computer Science, 2009
<http://www.igi.tugraz.at/lehre/CI>
9. Robin, 'BAUM WELCH ALGORITHM', NATURAL LANGUAGE PROCESSING ARTICLES ON NATURAL LANGUAGE PROCESSING, 2009
<http://language.worldofcomputing.net/pos-tagging/baum-welch-algorithm.html#>
10. Rose Hoberman, Dannie Durand, 'HMM Lecture Notes', Computational Genomics and Molecular Biology, Carnegie Mellon University, 2006
<http://www.cs.cmu.edu/~durand/03-711/2006/Lectures/hmm-bw.pdf>
11. Matthew J. Johnson, Alan S. Willsky, 'The Hierarchical Dirichlet Process Hidden Semi-Markov Model', 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010), Avalon, California, 2010

<http://www.mit.edu/~mattjj/uai2010>

12. ChengXiang Zhai, 'Assignment #5: Hidden Markov Models for Information Extraction', Department of Computer Science University of Illinois at Urbana-Champaign, 2003
<http://sifaka.cs.uiuc.edu/course/498cxz04f/assign5.html>
13. Shun-Zheng, YuHisashi Kobayashi, 'An Efficient Forward-Backward Algorithm for an Explicit-Duration Hidden Markov Model', Pages: 11-14, IEEE SIGNAL PROCESSING LETTERS, VOL. 10, NO. 1, 2003.
http://sist.sysu.edu.cn/~syu/Publications/IEEE_SPL03.pdf
14. Clayton Scott, Robert Nowak, 'Maximum Likelihood Estimation', 2004
<http://cnx.org/content/m11446/latest/>
15. Jae Myung, 'Tutorial on maximum likelihood estimation', Journal of Mathematical Psychology, 2002
citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.74.671
16. Kazem Taghva, Jeffrey Coombs, Ray Pereda, Thomas Nartker, 'Address Extraction Using Hidden Markov Models', Information Science Research Institute University of Nevada, Las Vegas, 2005
<http://cat.inist.fr/?aModele=afficheN&cpsidt=17027918>
17. Joseph Picone, 'LECTURE 33: SMOOTHING N-GRAM LANGUAGE MODELS', Insitute of Signal and Image Processing, Temple

University, 2009

http://www.isip.piconepress.com/publications/courses/msstate/ce_8463/lectures.

18. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, 'Introduction to Information Retrieval' Cambridge University Press, 2008
<http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf>
19. Mengqiu Wang, 'A Presentation on Smoothing', 2010
<http://www.stanford.edu/class/cs224n/.../cs224n-section1-smoothing-040910.ppt>
20. Mohammed Waleed Kadous, 'Auslan Sign Recognition' PhD dissertation, School of Computer Science and Engineering, University of New South Wales, 1998
<http://www.cse.unsw.edu.au/~waleed/phd/html/node35.html>

VITA

Graduate College
University of Nevada, Las Vegas

Nagendra Abhinav Dasu

Degrees:

Bachelor of Technology in Computer Science, 2009
Jawaharlal Nehru Technological University, India

Thesis Title: Implementation of Hidden Semi-Markov Models

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. Muthukumar Venkatesan, Ph.D.