5-2011

# Topic detection and tracking using hidden Markov models

Aditya S. Tatavarty
*University of Nevada, Las Vegas*

# TOPIC DETECTION AND TRACKING USING HIDDEN MARKOV MODELS

by

Aditya Sowmya Tatavarty

Bachelor of Engineering in Computer Science and Engineering
Jawaharlal Nehru State University, India
May 2009

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**

**Graduate College**
**University of Nevada, Las Vegas**
**May 2011**

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

The Graduate College

We recommend the thesis prepared under our supervision by

Aditya Sowmya Tatavarty

entitled

Topic Detection and Tracking Using Hidden 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

Venkatesan Muthukumar, Graduate Faculty Representative

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

May 2011

ABSTRACT

**Topic Detection and Tracking Using Hidden Markov Models**

by

Aditya Sowmya Tatavarty

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

There is a continuous progress in automatic recording of broadcast speech using speech recognition. With the increasing use of this technology, a new source of data is added to the pool of information available over web. This has increased the need to categorize the resulting text, based on their topic for the purpose of information retrieval.

In this thesis we present an approach to automatically assign a topic or track a change of topic in a stream of input data. Our approach is based on the use of Hidden Markov Models and language processing techniques. We consider input text as stream of words and use Hidden Markov Model to assign the most appropriate topic to the text. Then we process this output to identify the topic boundaries. The main focus of this thesis is to automatically assign a topic to specific story.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my heartfelt gratitude to my advisor Dr. Kazem Taghva for his tremedous support and invaluable guidance throughout this thesis work. I express my sincere thanks to Dr. Ajoy K. Datta for his help during my MS degree and also for being my committee member. I extend my gratitude to Dr. Laxmi P. Gewali and Dr. Venkatesan Muthukumar for agreeing to be a part of my committee. I am grateful to the staff of the School of Computer Science for being so helpful to us.

I am always obligated to God, my parents, and sisters for their love and support, and their encouragement to strive for the best. Last but not the least, I thank my friends for their support in the successful completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES

# CHAPTER 1

## INTRODUCTION

The explosive growth and dynamic environment of digital information is creating new challenges in the field of Information Retrieval. One of them is devising a scheme to track and detect dynamic information, such as, news, using Information Retrieval techniques. There are many Information Retrieval systems publicly available that aim to help users aware of the most current news on the Web. For example, Google News offers tracking services in which users receive an email when new articles about their subject of interest become available. Such services track information updates at the document level. Information professionals such as journalists often rely on tools such as Rich Site Summary (RSS) news feeds to keep track of the most current information and events. Thus there is an increasing need for automatic techniques to analyze, present, and visualize news to users in a meaningful and efficient manner.

Constant advance in science and technology makes collection of data and storage much easier and very inexpensive than ever before. Also, there is a continuous progress in automatic recording of broadcast speech using speech recognition. This results in enormous increase in the size of data available. This has increased the need to categorize the dataset available based on their topic for the purpose of information retrieval. Topic Detection and Tracking systems are mainly used to

discover the structure of topics in non-fragmented streams of news reports as they become available across multiple media.

Dynamic information is the main topic dealt in Topic Detection and Tracking (TDT). Research in this area mainly aims at effectively retrieving and organizing broadcast news (speech) and newswire stories (text) into groups of events. Different methods have been proposed for classification of text into predefined categories. In this thesis, we use Hidden Markov models for Text Detection and Tracking.

### 1.1 Thesis Overview

Hidden Markov Models are used in Text Detection and Tracking by splitting a stream of data into stories and assigning an appropriate topic to each of these stories. The process involved is as follows. Input to our model is an unlabeled sequence of words. Output is a sequence of words labeled with the most appropriate topic. This output is processed to identify the topic boundaries. In our experiments, we integrate information about text into our model, and use it to classify given text into one of four categories: news, commercials, sitcoms, or soaps.

The remainder of the report is organized as follows. Section 2 contains a brief description of Hidden Markov Models and text detection and tracking systems. Section 3 describes the application of Hidden Markov Models for Text Detection and Tracking. This section also includes an explanation of the steps involved in implementing this model in Section 4. These steps include an explanation of preprocessing steps

involved in converted the stream of input into required input format, tagging the input stream with the most appropriate boundaries, and identifying the topic boundaries. Experimental results are reported in Section 5. In Section 6, we make some concluding remarks and shed some light on possible future work.

# CHAPTER 2

# BACKGROUND

Data Mining is defined as the process of extracting unknown but useful information from databases. In recent years, data mining not only attracted business organizations, but also has been widely used in the information technology industry [1]. After the Internet and Web were made accessible for everything and to everyone, we can now reach large amounts of data. However, to make the date useful for any practical purpose, we need to be able to parse the data to extract some meaningful knowledge. This is precisely where data mining plays a huge role. There are many well-known data mining schemes, Topic Detection and Tracking (TDT) is one of them. This thesis deals with TDT using supervised machine learning technique.

Machine Learning is defined as "the ability of a machine to improve its performance based on previous results." [2] In other words, it is a system capable of learning from experience and analytical observation, which results in continuous self improvement, thereby offering increased efficiency and effectiveness. In general, there are four different types of machine learning techniques. They are:

1. Supervised learning,
2. Unsupervised learning,
3. Semi-supervised learning, and
4. Reinforcement learning. [3]

This thesis deals with topic detection and tracking using supervised learning technique. Supervised learning is a machine learning technique that learns from a training data set. A training data set consists of input objects and categories they belong to. Assigning categories to input objects is carried out manually by an expert. Given an unknown object, supervised learning technique must be able to predict an appropriate category based on prior training.

Topic Detection and Tracking is a fairly new area of research in Information Retrieval. In this thesis, we use Hidden Markov Models to implement Topic Detection and Tracking. Text features are extracted and classified with Hidden Markov Model (HMM). Hidden Markov Model (HMM) is a popular technique widely used in speech recognition. The fundamental nature of HMM is to construct a model that explains the occurrence of observations (symbols) and use it to identify other observations sequences. However, to date, its applications have been focused on crypt-analysis and speech recognition. In our work, we extended it to topic detection and tracking.

**2.1 Hidden Markov Models**

Before defining Hidden Markov Model (HMM), we introduce the concept of Markov Chain, also referred to as an observed Markov Model.

A Markov Chain is a particular case of a weighted automaton in which the input sequence uniquely determines the states through which the automaton traverses. Markov chains are sequences of random

variables in which the future variable is determined by the present variable, but is independent of the way in which the present state is reached from its predecessors [8].

Hidden Markov Models (HMMs) separate the observations from the states; the observations (outputs) are visible, but the state sequences that led to them are hidden. It is "Markov" because the next state is determined solely from the current state. It is "Hidden" because the actual state sequences are hidden [10].

Hidden Markov Models can be defined as follows:

"A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states." [12]

In this thesis, we used a discrete first order Hidden Markov Model. As the observations of our Markov Model are characterized as discrete symbols chosen from a fine alphabet, it is a discrete Markov Model. We also assumed that transition probabilities depend only on the previous state which makes it a first-order Markov model.

A set of five elements can be used to describe an HMM.

In the following section, we will define the elements of an HMM and their notations.

### 2.1.1 Elements of an HMM

We can define an HMM as a 5-tuple (S, V, $\pi$, A, B).

We define Q to be a fixed state sequence of length T, and the corresponding observations O:

$Q = q_1, q_2, \cdots, q_T$

$O = o_1, o_2, \cdots, o_T$

T - Number of observations in the sequence

HMM Notation: $\lambda = (A, B, \pi)$

- N: Number of states in the Model.

  There is a finite set of states in a model. The states in an HMM are hidden, but there is a lot of significance to these states in defining an HMM. We denote the individual states as S1, S2, S3, …, Sn.

  S = { S1, S2, S3, … , Sn}.

- M: Number of distinct symbols observable in states.

  These symbols correspond to the observable output of the system that is being modeled. We denote the individual states as $v_1$, $v_2$, $v_3$, …, $v_M$.

  V = { $v_1$, $v_2$, $v_3$, …, $v_M$}.

- A: State transition probability distribution

  A is transition array that store the state transition probabilities. A={$a_{ij}$ }, where aij stores the probability of state j following state i.

  $a_{ij} = P(q_t = S_j / q_{t-1} = S_i)$, $i \geq 1$ and $j \geq N$

  aij, the probability of moving from state Si to Sj at time t

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & . & . & a_{1n} \\ a_{21} & a_{22} & a_{23} & . & . & a_{2n} \\ a_{31} & a_{32} & a_{33} & . & . & a_{3n} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ a_{n1} & a_{n2} & a_{n3} & . & . & a_{nn} \end{bmatrix}$$

At each time t, a new state is entered which depends on the transition probability distribution of the state at time t – 1. Transition to the same state is also possible. An important point about transition probabilities is that they are independent of time; the probability of moving from state Si to state Sj is independent of time t.

- B: Observation symbol probability distribution

  $B = \{b_j(k)\}$ is the output symbol array that stores the probability of an observation $V_k$ being produced from the state j, independent of time t. Observation symbol probability or Output Emission Probability estimates are also independent of time. The probability of a state emitting a particular output symbol does not vary with time.

  $B = \{ b_j(k) \}$ , $b_i(k) = P(x_t = v_k / q_t = S_j)$ $1 \le j \le N$ and $1 \le k \le M$

  $b_i(k)$, the probability of emitting symbol $v_k$, when state Sj is entered at time t.

  After each transition is made, a symbol is output based on the output probability distribution, which depends only on the current state.

- π: Initial state distribution

$\pi = \{\Pi_i\}$ is the initial probability array that stores the probability of the system starting at state i in an observation. It is the probability of state Si being the start state in an observation sequence.

$\pi = \{ \Pi_i \}$, $\Pi_i = P(q_1 = S_i)$, $1 \leq i \leq N$

$\Pi_i$, the probability of being in state I at time t=1

A complete specification of an HMM consists of the above five elements, {S, V, π, A, B}. We usually use a compact notation λ=(A, B, π) to represent the above complete parameter set of HMM.

The following are obvious constraints on the elements of an HMM.

$\sum_{i=1}^{n} \pi_i = 1$

$\sum_{j=1}^{n} a_{ij} = 1$ for all i

$\sum_{j=1}^{n} b_i (v_k) = 1$ for all i

As each $a_{ij}$ represents the probability $P(S_j / S_i)$, the laws of probability require that the values of the outgoing arcs from a given state must sum to one. Same laws of probability apply to Initial Probabilities and Output Emission Probabilities.

The operations of an HMM are characterized by state sequence Q and the observation sequence O.

$Q = q_1, q_2, \cdots, q_T$

$O = o_1, o_2, \cdots, o_T$

Q is a fixed state sequence of length T, and the corresponding observation sequence is O. T is total number of observations in the

observation sequence, and Ot is one of the symbols from V (Output variables). Using an HMM, we can generate an observation sequence O = $o_1, o_2, \ldots, o_T,$. We can also estimate the most probable state sequence Q= $(q_1, q_2, q_3, \ldots q_t)$ given the set of observations O =( $o_1, o_2, \ldots, o_T$ ). Here the observations are assumed to have statistical independence. The Model has discrete observation (output) symbols. For an HMM to perform all these, we need appropriate values of N, M, A, B, and π. These values can be obtained by a learning process.

**2.2 Problems of Hidden Markov Model**

We can use HMMs to solve real problems with real data by solving these three problems.

These problems are:

Problem 1- Evaluation: Given an observation sequence O and a model λ, what is the probability of the observation sequence, $P(O|\lambda)$?

$P(O|\lambda) = P(O_1,O_2,\ldots, O_T \mid \lambda) = ?$

Problem 2- Decoding: Given an observation sequence O and the model λ, what is the most probable state transition sequence Q for O?

$Q^* = \arg\max_{Q=(q1, q2, \ldots, qT)} P(Q, O \mid \lambda) = ?$

Problem 3- Training: Given a training sequence O, find a model λ, specified by parameters (A, B, π) to maximize $P(O|\lambda)$ (we

assume for now that Q and V are known).

$$P ( O \mid \lambda = ( A, B, \pi ) ) < P ( O \mid \lambda ' = ( A' , B ' , \pi' ) )$$

$$\lambda^* = \text{argmax}_{\lambda} P(O \mid \lambda )$$

Only the evaluation problem has a direct solution. The other problems are harder and involve optimization techniques like dynamic programming. There are specific algorithms for each problem that explain the best way to solve them. The problem of evaluation is solved using the Forward and Backward iterative algorithms. The second problem is solved using the Viterbi Algorithm, also an iterative algorithm that outputs the best path by sequentially considering each observation symbol of O. The last problem which deals with training an HMM, can be solved by using Baum-Welch or Maximum Likelihood Estimation (MLE). The choice between these two algorithms can be made using the training data available for the learning process.

We will now discuss in detail how these algorithms solve the three problems associated with HMM. In each section, we will discuss different algorithms that can be efficiently used to solve the problem. Each algorithm is explained mathematically using equations that make use of HMM notation, $\lambda = (A, B, \pi)$. Also, all the operations of a HMM are characterized by the hidden state sequence Q and the observation sequence O that have the following denotations

$$Q = q1, q2, \cdots, qT$$

$$O = o1, o2, \cdots, oT$$

where T is the number of observations in O.

### 2.2.1 Evaluation

Problem 1 is the evaluation problem. Given a model and a sequence of observations, the question is how to compute the probability such that the model produces the observation sequence? Here we are trying to see how well a particular observation sequence matches the given model. This is an extremely important point. If there is a situation where a choice has to be made among several competing HMMs, the model that best suits the observation sequence can be found using Evaluation.

The simplest way to solve the evaluation problem is by following the Brute-Force approach. In this approach, we enumerate every state sequence of length T (the number of observations), and calculate the probability of each state sequence producing the given observation sequence.

This approach, although theoretically useful, can lead to very long computations as the number of computations required becomes exponential. The computational complexity is $O(N^T T)$. Even if we have small N and T, this is not feasible. For example, for N = 5 and T = 100, $\sim 10^{72}$ computations are needed.

There is a more efficient procedure to solve Problem 1. It is called Forward-Backward Procedure.

Forward Algorithm

In this algorithm, the number of computations required is low. We perform recursive evaluation, using an auxiliary variable αt(i), called the forward variable.

$$\alpha_t(i) = P((O(1), O(2), \ldots O(t)), i(t) = q_i \mid \lambda)$$

$\alpha_t(i)$, the probability of the partial observation sequence (until time t) and internal state $q_t$ = Si given the model λ.

$\alpha_t(i)$ makes a recursive calculation possible because in a first-order HMM, the transition and emission probabilities only depend on the current state [6]. These recursive calculations reduce the number of calculations needed to obtain P (O | λ ).

We can solve $\alpha_t(i)$ inductively as follows

Step 1: Initialization

$\alpha_1(i) = \pi_i \ . \ b_i(O_1)$ , $1 \le i \le N$

Step 2 : Induction

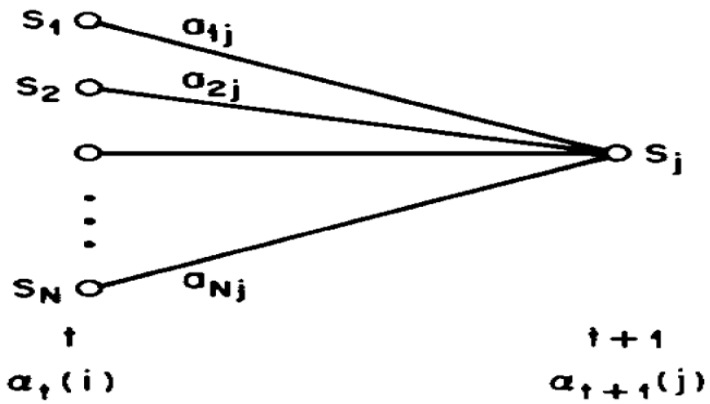


Figure 1: Operations for computing the forward variable $\alpha_j$ (t + 1)

Here we calculate $\alpha_{t+1}(j)$, the next step, using the previous one $\alpha_t(i)$. $\alpha_{t+1}(j)$ represents the probability of the observation sequence up to time t + 1 and being in state $S_j$ at time t + 1.

$$\alpha_{t+1}(i) = b_{j\,O_{t+1}} \sum_{i=1}^{N} a_{ij} \alpha_t(i)$$

$$1 \le t \le T\text{-}1,\ 1 \le j \le N$$

Note that we have to keep track of $a_t(i)$ for all N possible internal states. These values are used in the termination step.

Step 3: Termination:

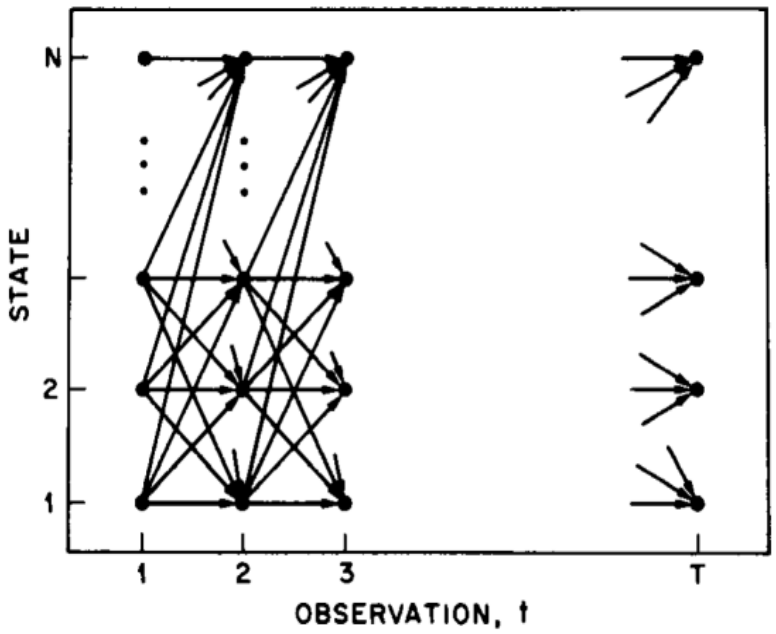


Figure 2: αj(t) in terms of a lattice of observations and states

If we know $\alpha_T(i)$ for all the possible states, we can calculate the overall probability of the sequence given the model, $P(O \mid \lambda)$

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

The forward algorithm allows us to calculate $P(O|\lambda)$. As it can be seen from the above algorithm, it has a computational complexity $O(N^2T)$. This is linear in T, rather than exponential. This means that it is feasible to implement. Apart from calculating $P(O|\lambda)$, this algorithm is also used in Baum Welch Algorithm for unsupervised learning.

Backward Algorithm:

The α values computed using the forward algorithms is sufficient to solve the first problem, $(O \mid \lambda)$. However, in order to solve the third problem, we will need another set of probabilities β values. Similar to forward variable, we define a backward variable, $\beta_t(i)$. We denote the backward variable $\beta_t(i)$ as the probability of the partial observation sequence after time t, given state Si at time t

$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T \mid qt = Si, \lambda)$, $1 \le t \le T$, $1 \le i \le N$

Just like α's, β's can also be computed using the following backward recursive procedure:

Step 1: Initialization

The initialization step arbitrarily define $\beta_T(i)$ to be 1 for all i.

$\beta_T(i) = 1$, $1 \le i \le N$

This algorithm is backward in the sense that the time interval t is from T to one.

Step 2: Induction

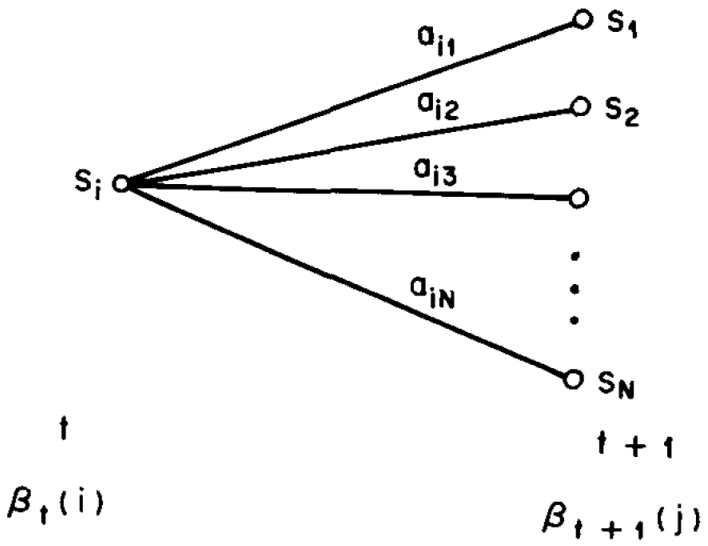


Figure 3: Operations for computing the backward variable $\beta_t(i)$

Here we calculate $\beta_t(i)$, the next step, that makes use of the previous step, $\beta_{t+1}(j)$.

$\beta_t(i) = \sum_{j=i}^{N} a_{ij} \cdot b_j(O_{t+1}) \beta_{t+1}(j)$ , t=T-1, T-2, . . . , 1 $1 \le i \le N$

In order to be in state Si at time t, and to account for the observation sequence from time t+1 onwards, we have to consider all possible states Sj at time t+1, accounting for the transition from Si to Sj (aij), as well as observation Ot+1 in state j ( $b_j(0t+1)$), and then account for the remaining partial observation sequence from state j( $\beta_{t+1}(j)$ ) [1].

Step 3: Termination

$$p(O \mid \lambda) = \sum_{i=1}^{N} \beta_1(i)$$

Again the computation of $\beta_t(i)$ , $1 \le t \le T$, $1 \le i \le N$ require $N^2T$ calculations.

We will see later how the backward as well as the forward algorithms are used to solve problems 2 and 3 of HMM.

### 2.2.2 Decoding

Problem 2 deals with decoding. In decoding, we attempt to uncover the hidden part of the HMM. In other words, we try to find the optimal state sequence for a given observation sequence. Unlike evaluation, in decoding, there is no single optimal sequence. To get a correct solution, we choose states that are individually most likely and then find the single best state sequence that guarantees that the uncovered observation sequence is valid. The most common solution to the decoding problem is the Viterbi algorithm which also uses partial sequences and recursion.

Viterbi Algorithm:

The Viterbi algorithm is a dynamic programming algorithm that computes the most likely state transition path given an observed sequence of symbols. It is actually very similar to the forward algorithm, except that we will be taking a "max", rather than a " $\sum$ ", over all the possible ways to arrive at the current state under consideration. However, the formal description of the algorithm involves some cumbersome notations [11].

We need to define the following quantity for solving the problem using Viterbi Algorithm.

$$\delta_t(i) = \max_q P(q_1, q_2, \ldots q_p = i, o_1, o_2, \ldots o_t \mid \lambda)$$

$\delta_t(i)$ is the probability of the most probable path ending in state Si at time t.

By induction, we have

$$\delta_{t+1}(j) = \max_i(\delta_t(i)a_{ij})b_j(o_{t+1})$$

To retrieve the state sequence, we need to keep track of the argument that maximized $\delta_{t+1}(i)$, for each *t* and *j*. We use an array $\psi_t(j)$ for backtracking the state sequence.

The Viterbi Algorithm is as follows:

Step 1: Initialization

$$\delta_1(i) = \pi_i b_i(o_1) \qquad 1 \le i \le N$$

$$\psi_1(i) = 0$$

Step 2: Recursion

$$\delta_t(j) = \max_{1 \le i \le N}(\delta_{t-1}(i)a_{ij})b_j(o_t)$$

$$\psi_t(j) = \arg\max_{1 \le i \le N}(\delta_{t-1}(i)a_{ij})$$

$$2 \le t \le T, 1 \le j \le N$$

Step 3: Termination

$$P^* = \max_{1 \le i \le N} \delta_T(i)$$

P* gives the state-optimised probability

$$q_T^* = \arg\max_{1 \le i \le N} \delta_T(i)$$

Q* is the optimal state sequence ( Q* = {q1*,q2*,...,qT*} )

Step 4: Backtrack State Sequence

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad t + T - 1, T - 2, .. 1,$$

Viterbi algorithm is similar to Forward algorithm except for the backtracking step and the maximization over previous states instead of summation. So, the time complexity is $O(N^2T)$. We can use a trellis structure to properly explain the Viterbi Algorithm.

### 2.2.3 Training

The problem 3 in HMM is training to obtain the most likely parameters that best models a system, given a set of sequences that originated from this system.

There is no known way to analytically solve for the model that maximizes the probability of the observation sequence(s). So, we come up with models $\lambda = (A, B, \pi)$ which locally maximizes $P(O)$.

HMM = Topology + Statistical parameters

During the training process, we compute the statistical parameters of the HMM. The topology is already designed. So, the input to a training algorithm would be a database of sample HMM behaviour, and the output is the transition, emission, and initial probability distribution of HMM. Thus, we can conclude that given a set of examples from a process, we should be able to estimate the model parameters $\lambda = (A,B, \pi)$ that best describe that process.

There are two standard approaches to the learning task based on the form of the examples (database available for learning process), supervised and unsupervised training. If the training examples contain both the inputs and outputs of a process, we can perform supervised

training. It is done by equating inputs to observations and outputs to states. 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 supervised training.

Supervised Learning:

The easiest solution for creating a model λ is to have a large corpus of training examples, each annotated with the correct classification. If we have such tagged training data, we use the approach of supervised training. In supervised learning, we count frequencies of transmissions and emissions to estimate the transmission and emission probabilities of the model λ.

Maximum Likelihood Estimation (MLE):

MLE is a supervised learning algorithm. In MLE, we estimate the parameters of the model by counting the events in the training data. This is possible because the training examples for a MLE contain both the inputs and outputs of a process. So, we can equate inputs to observations and outputs to states, and we easily obtain the counts of emissions and transitions. These counts can be used to estimate the model parameters that represent the process.

$$a_{ij} = \frac{\#\ of\ transitions\ from\ i\ to\ j\ in\ the\ sample\ data}{total\ \#\ of\ transition\ from\ the\ state\ i\ in\ sample\ data}$$

$$b_i(v_k) = \frac{\#\ of\ emissions\ of\ the\ symbol\ v_k\ from\ i\ in\ the\ sample\ data}{total\ \#\ of\ emissions\ from\ the\ state\ i\ in\ sample\ data}$$

There is a possibility of $a_{ij}$ or $b_i(v_k)$ being zero. For example, consider the case where state $s_i$ is not visited by the sample training data. Then $a_{ij}=0$. In practice, when estimating an HMM from counts, it is usually necessary to apply smoothing in order to avoid zero counts and improve the performance of the model on data not appearing in the training set.

Unsupervised learning:

Key idea of unsupervised learning is iterative improvement of model parameters. We can use iterative expectation-maximization algorithm, Baum-Welch, to find local maximum of P (O | λ). Baum-Welch algorithm uses the forward and backward algorithms to calculate the auxiliary variables α and β.

B-W algorithm is a special case of the EM algorithm:

- E-step: calculation of ξ and γ
- M-step: iterative calculation of

E-step *:*

In order to describe the procedure for solving the problem, we need to first define $\xi_t(i,j)$

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda)$$

$\xi_t(i,j)$ is the probability of being in state Sj at time t, and state Sj at time t+1, given λ, O.

$\gamma_t(i)$ = P ( $q_t = S_i$ | O, λ )

$\gamma_t(i)$ is the probability of being in state Si at time t for a given observation sequence O and model λ.

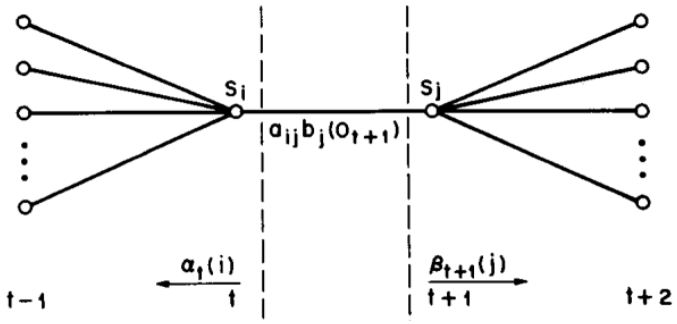


Figure 4: Operations for computing $\xi_t(i,j)$

We can relate $\gamma_t(i)$ and $\xi_t(i,j)$ by summing over j

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

M-step:

$\sum_{t=1}^{T-1} \gamma_t(i)$ = expected number of transitions from state Si to Sj

$\sum_{t=1}^{T-1} \xi_t(i,j)$ = expected number of transitions from Si to Sj.

Using these formulas, we can re-estimate the parameters of an HMM.

$\hat{\pi} = \gamma_1(i)$, the expected frequency of state i at time t=1

$\hat{a}_{ij} = \frac{\sum \xi_t(i,j)}{\sum \gamma_t(i)}$ , ratio of expected number of transitions from state $i$ to $j$ over expected number of transitions from state $i$

$\hat{b}_j(k) = \frac{\sum_{t,o_t=k} \gamma_t(j)}{\sum \gamma_t(j)}$ , ratio of expected number of times in state $j$ observing symbol $k$ over expected number of times in state $j$.

We may face some numerical problems while dealing with long observation sequences that have to be solved using scaling.

### 2.3 Example of Hidden Markov Model

Example 1: Coin Tossing Experiment

Consider the coin tossing experiment of Markov Models. But here the person on the other side of the curtain has several coins both biased and un-biased with him. He selects one of his several coins, and tosses it. Then the person tells us the outcome (H, T), but not the coin selected. He does this several times, and the outcome obtained after each trial is recorded as an observation. Here the coins will be the hidden states, and H, T are the observations.

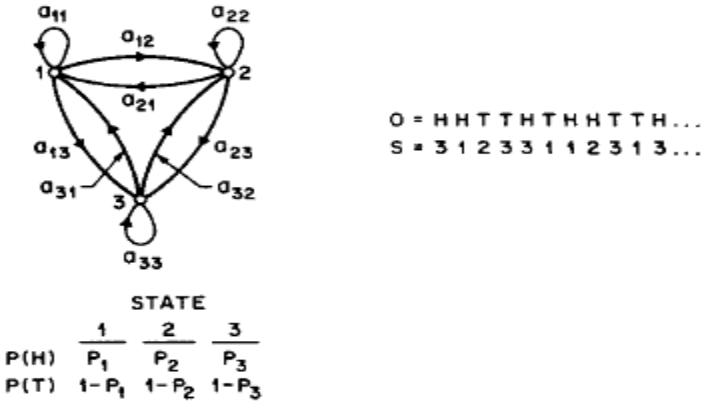


Figure 5: HMM-Coin Tossing Experiment

We make an assumption that the person has three coins, and he chooses one among these three, based on some probabilistic event [4].

# CHAPTER 3

# TOPIC DETECTION AND TRACKING USING HIDDEN MARKOV MODEL

Text Detection and Tracking is used to develop algorithms for discovering and grouping together topically related material in streams of data such as newswire and broadcast news [5]. In this thesis we use Hidden Markov Models (HMMs) to solve the problem of tracking a changing topic.

In this model hidden states are the topics and the observations are words. Our HMM is trained using some sample stories on several events of interest (topics) and is used to automatically find the topic for other stories. The input to the program will be a stream of words, first on one topic, then on another topic, and so on. The goal is to identify the topic of each of these words and segment the stream of text into blocks based on the topic identified. In a document, for the given sentence sequence, the HMM aims to find the most likely topic sequence.

The main components of this program are the following:

- Building an HMM from training data;
- Implementing the Viterbi algorithm for finding the most likely sequence of states using the HMM built from training data; and
- Running the code on several test datasets and exploring its performance.

## 3.1 Building an HMM from data

The first step in implementing our model is building an HMM from training data. This training data is a stream of words collected from

sources which cannot be directly used in our model. In order to use it in our model we have to pre process it. Several preprocessing techniques like Tokenizing, Parsing, and Stemming are applied on training data before using it in our Model. This preprocessed data is now fed as input to our model.

Using the preprocessed data as input we train our model. The output of this learning process are the five parameters of the HMM, (S, V, π, A, B)

S: States in the Hidden Markov Model

We denote the individual states as $S_1$, $S_2$, $S_3$, …, $S_n$.

$S = \{S_1, S_2, S_3, \ldots, S_n\}$

The data set we used in the experiment is categorized into six groups or topics which form the states of our HMM.

V: Distinct symbols observable in states.

We denote the individual states as $v_1$, $v_2$, $v_3$, …, $v_M$

$V = \{ v_1, v_2, v_3, \ldots, v_M \}$

List of output symbols is list of significant terms obtained after preprocessing the training data.

We estimate the other parameters of our model using Maximum Likelihood Estimate (MLE) described in section 2.

Three sets of probabilities calculated using MLE are:

- A: State transition probability distribution

  $A=\{ a_{ij} \}$. aij stores the probability of state j following state i.

$a_{ij} = P(q_t = S_j / q_{t-1} = S_i)$, $i \geq 1$ and $j \geq N$

$a_{ij}$, the probability of moving from state $S_i$ to $S_j$ at time t

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & . & . & a_{1n} \\ a_{21} & a_{22} & a_{23} & . & . & a_{2n} \\ a_{31} & a_{32} & a_{33} & . & . & a_{3n} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ a_{n1} & a_{n2} & a_{n3} & . & . & a_{nn} \end{bmatrix}$$

- B: Observation symbol probability distribution

  B = { $b_j(k)$ } is the output symbol array that stores the probability of an observation $v_k$ being produced from the state j, independent of time t.

  $B = \{ b_j(k) \}$ , $b_j(k) = P(x_t = v_k / q_t = S_j)$ $1 \leq j \leq N$ and $1 \leq k \leq M$

  $B_j(k)$, the probability of emitting symbol $v_k$ when state Sj is entered at time t

- π: Initial state distribution

  π = { Πi } is the initial probability array that stores the probability of the system starting at state i in an observation.

  $\pi = \{ \Pi_i \}$, $\Pi_i = P(q_1 = S_i)$, $1 \leq i \leq N$

We will compute probability estimates of these three parameters from the training data…Training data consists of one or more sequences of state-output pairs, During this training phase, we assume that the state variables are visible. Given these sequences, we estimate the probabilities that define the HMM using Maximum Likelihood Estimation.

We estimate the observation symbol probabilities (bi(k)), the probability of emitting symbol $v_k$ in state $s_i$ in the following manner. We count the number of times state s emits output symbol $v_k$ in the given data, and divide it by a normalization constant so that the sum of all the output probabilities from state $s_i$ adds up to one. In our case, we used the normalization constant to be the number of times state s appears in the data.

N(k,i): Number of times state $s_i$ has seen output symbol k

N(s): Number of occurrences of state $s_i$

V: entire vocabulary (all output symbols)

$$b_i(k) = P(k|i) = N(k,i) \ / \ N(i)$$

When we are making estimates of this sort, there is a need to use smoothing techniques to smooth the probability estimates. The reason is sparse training data causes poor probability estimates that is, unseen probabilities have emission probabilities of zero.

To see what this means and understand the need to use smoothing let us consider an example of flipping a coin. Assume the probability of heads is p, p is unknown, and our goal is to estimate the value of p. This can be calculated by counting the number of times the coin came up heads and divide it by the total number of coin flips. If the coin is flipped 1000 times and it shows heads 367 times then the value of p is 0.367. However, if we flip the coin 2 times and we get tails both the times the value of p is 0. It is not reasonable to assume that the coin will always

come up with heads. To avoid such situations arising from sparse training data we applied Laplace smoothing to our model.

Laplace smoothing is a simple smoothing technique in which we add pseudo counts to all word frequencies to move them towards uniform distribution. The result is all the unseen words have equal non-zero probabilities.

Mathematically Laplace smoothing can be described as follows:

N(k,i): Number of times state i has seen output symbol k

N(s): Number of occurrences of state i

V: entire vocabulary (all output symbols)

$$b_i(k) = P(k|i) = (N(k,i) + 1) / (N(i) + |V|)$$

In the above example p would be

p = (1+ Number of heads) / (2 + Number of flips)= (1+0) / (2+2) = 0.25

Another advantage of Laplace smoothing is that it avoids estimating any probabilities to be zero, even for events never observed in the data. For HMMs, this is important since zero probabilities can be problematic for some algorithms.

We estimate the probability of transition from one state to another in the following manner. To calculate the transition probability from state $s_i$ to state $s_j$ we count the number of transitions from $s_i$ to $s_j$ and divide it by the total number of transitions from state $s_i$. Laplace-smoothed estimates should also be applied on the transition probabilities

from one state to another. After accommodating this change the formula for transition probabilities is

$N(s_i , sj)$: Number of times we move from state $s_i$ to state $s_j$

N(si): Number of transitions from state si

V: entire vocabulary (all output symbols)

$a_{ij} = P(q_t = s_j / q_{t-1} = s_i) = (N(s_i , s_j) + 1) / (N(s_i) + |V|)$

To estimate the probability of starting from a state ($\Pi_i$) in an observation sequence we count the number of times our input started from a state say $s_i$ and divide it by total number the number of input sequences in training data.

$\Pi_i$ , the probability of being in state I at time t=1

N(s1): Number of times we start from state s

N: Number of input sequences

$\Pi_i = N(si) / N$

The HMM parameters computed from the MLE are written into a model file.

**3.2. Finding the most likely state sequence**

The second part of the thesis is to compute the most probable sequence of states for a given sequence of outputs using the HMM that we built from the training data. This is done by implementing the Viterbi algorithm (described in section 2) on the HMM model generated from training data.

Viterbi algorithm will be provided with the output part of the state – output pairs from the test sequences. This output part of the test sequence and the HMM model generated from previous step are used to calculate the most likely state sequence to have produced such an output sequence. Estimated state sequence is compared with the state part of the test sequence. A clear explanation of Viterbi algorithm is provided in section 2. The functions used and the steps involved in implementing this step is described in detail in section 3.

Viterbi algorithm involves multiplying many probabilities together. Since each of these numbers is less than one, we can end up working with numbers that are tiny enough to be indistinguishable from zero by a real computer. To avoid this we worked with log of probabilities. For instance, if we need to calculate product pq of probabilities p and q, we would instead compute log of their product simply by adding their logarithms using the rule.

$$\log(pq) = \log(p) + \log(q).$$

### 3.3. Exploring performance on actual datasets

After completion of step 2 we evaluate the performance of our model on the test data set from data collection. The performance of text detection and tracking model built is evaluated based on standard precision, recall and F1 values.

CHAPTER 4

IMPLEMENTATION

To perform Text Detection and Tracking, we collected a set of 5302 articles from six newsgroups on six different topics, namely rec.sports.baseball, rec.auto, talk.politics.guns, sci.med, talk.religion.misc, and comp.os.ms-windows.misc. They can be categorized into six topics: baseball, cars, guns, medicine, religion, and windows. These topics become the six hidden states of our model. Document in the data set should be preprocessed before giving it as input to our model. Out of the 5302 documents present in the data set, we used 1500 articles in testing and the rest in training.

**4.1 Documents Preprocessing**

All the documents are organized into folders based on their categories, which are based on the topic they belong to. They need to be converted into a stream of text tagged to their corresponding topic. Initially, all training documents are parsed to convert all letters to lower case, and to remove the punctuations by converting them to white space. All the special characters are also converted to white space. Once the documents are parsed, they should be tokenized.

Tokenization is the process of breaking parsed text into pieces, called tokens. For example, consider the sentence "Although there was inflation, at least the economy worked," from a document that belongs to category Trade tokenized as shown in Table 4.3.

| Although |
| --- |
| There |
| Was |
| Inflation |
| At |
| Least |
| The |
| Economy |
| Worked |

Table 1: List of tokens

Next step after tokenization is removing stop words. Common words such as 'are', 'the', 'with', 'from', etc. that occur in almost all documents, do not help in deciding if a document belongs to a category. Such words are referred to as stop words. So, these words can be removed by forming a list of stop words. This thesis works on a total of 416 stop words. Once stop words are removed, next step performed is stemming.

Stemming refers to the process of reducing terms to their stems or root variant. For example, “computer”, “computing”, and “compute” are reduced to “comput”, and “engineering”, “engineered”, and “engineer” are reduced to “engine”. The main advantage of using stemming is to reduce the computing time and space as different forms of words are stemmed to a single word. The most popular stemmer in English is the Martin Porter's stemming algorithm shown to be empirically effective in

many cases. It is implemented in various programming languages which are available for free. This thesis works on stemming algorithm programmed by porter in java [6]. After stemming all terms, the next step is to build a list of output variables from this list of terms.

We store the values of output variables, their term count, and document count using linked hash maps and tree set. Tree set is used to store the list of output variables. We use two hash maps to store document frequency and term frequency of output variables.

Consider a simple example with three documents, D1, D2, and D3.

D1: “it is an apple”

D2: “apple is in the basket”

D3: “it is a banana”

The tree set built is as shown in Table 2.

| Term |
|---|
| Apple |
| Basket |
| Banana |

Table 2: A record level inverted index file

In this thesis, we build document frequencies and term frequencies to figure out significant terms in the collection. Document frequency is defined as the number of documents that contain a particular term.

Term frequency is defined as the number of times that a particular term occurs in the collection. Consider the above example for which the document frequencies are shown in Table 4.5. The document frequency for the term "apple" is 2 because it occurs in two documents D1 and D2.

| Term | Document Frequency |
|---|---|
| apple | 2 |
| basket | 1 |
| banana | 1 |

Table 3: Terms with their document frequencies

A major difficulty of text detection and tracking problem is the high dimensionality of feature space, i.e., the total number of output symbols. Even for a moderate-sized text collection, there are hundreds of thousands of unique terms [7]. So, our concentration is to reduce the number of terms in the collection, known as dimensionality reduction. There are many known methods to perform dimensionality reduction. This thesis works on term selection based on document frequency thresholding. Document frequency thresholding is the simplest dimensionality reduction technique used for reducing vocabulary in the collection. This is carried out based on a predefined threshold value such that only those terms are removed from the collection that are less than the given threshold value.

This thesis concentrates only on those terms whose document frequency is greater than three and less than the number of training documents subtracted by 3, and excludes the remaining terms. Suppose, if document frequency is less than three, then those terms are considered as rare terms as they appear in fewer documents. Basically, the rare terms are considered to be non-informative for category prediction in global performance and hence can be removed [6]. If some terms have very high document frequency, it means that those terms occur in most of the document collection. So, based on those terms, one cannot distinguish between two documents, and hence can be removed.

Significant terms obtained from the above step form the output symbols of HMM. The next step is to remove other terms from the document collection. All the files in the document collection are compared with the output symbols, and the words that do not belong to this set of output symbols are removed from the collection.

The resulting documents, after preprocessing, contain only the output symbols. These output symbols are tagged according to the category to which they belong. The 5302 articles, classified into several sets of categories, were then randomly permuted and concatenated together forming a sequence of states (topics) and outputs (words).

After preprocessing the data and converting it into a format required by our program, the next step is to develop a model to solve our

problem and test its efficiency. The problem can now be divided into two phases. They are:

- Training phase and
- Test phase or Text Detection and Tracking phase.

**4.2 Training Phase**

Out of the 5306 present in the document collection, we are using 3806 documents for training. Our HMM is trained using stories on several events of interest (topics) from the test set. The goal of this phase is to build an HMM using preprocessed training data as input. The output of this learning process is the five parameters of the HMM, (S, V, π, A, B).

States in the Hidden Markov Model: S:

We denote the individual states as $S_1$, $S_2$, $S_3$, …, $S_n$.

The six topics into which the data set can be categorized become the six hidden states of our model. Each state is represented by an integer in the range 0 (inclusive) to number of states (exclusive). Hence the individual states in our HMM model are

S = {0, 1, 2, 3, 4, 5}.

Distinct symbols observable in states: V

We denote the individual states as $v_1$, $v_2$, $v_3$, …, $v_M$

V = { $v_1$, $v_2$, $v_3$, …, $v_M$ }

List of output symbols is a list of significant terms obtained after preprocessing the training data.

We estimate the other parameters of our model using Maximum Likelihood Estimate (MLE) described in Section 2.

Three sets of probabilities calculated using MLE are:

State transition probability distribution A

$A=\{a_{ij}\}$: aij stores the probability of state j following state i.

$N(s_i , sj)$: Number of times we move from state $s_i$ to state $s_j$

N(si): Number of transitions from state si.

V: entire vocabulary (all output symbols).

$a_{ij} = P(q_t = s_j / q_{t-1} = s_i) = (N(s_i , s_j) + 1) / (N(s_i) + N)$, $i \geq 1$ and $j \geq N$.

Transition probability matrix in our HMM model is a 6 by 6 array. It gives the probability of changing from one particular topic to another.

Observation symbol probability distribution: B

$B = \{b_j(k)\}$ is the output symbol array that stores the probability of an observation $v_k$ being produced from the state j, independent of time t.

$B = \{b_j(k)\}$ , $b_i(k) = P(x_t = v_k / q_t = S_j)$ $1 \leq j \leq N$ and $1 \leq k \leq M$.

N(k,i): Number of times state i has seen output symbol k.

N(s): Number of occurrences of state i.

V: entire vocabulary (all output symbols)

$bi(k) = P(k|i) = (N(k,i) + 1) / (N(i) + |V|)$

Initial state distribution: Π

$\Pi = \{\Pi_i\}$ is the initial probability array that stores the probability of the system starting at state $s_i$ in an observation.

$\pi = \{\Pi_i\}$, $\Pi_i = P(q_1 = S_i)$, $1 \leq i \leq N$

$\Pi_i$ , the probability of being in state $s_i$ at time t=1.

$N(s_i)$: Number of times we start from state $s_i$.

N: Number of input sequences.

$\Pi_i = N(s_i) / N$

Initial state probability matrix in our HMM model is an array of length 6. It gives the probability for every particular topic to be the first topic in a sequence.

In our program, we used CountSequence and UpdateParameter functions to compute the initial state probabilities, transition probabilities, and observation symbol probabilities, given tagged sequences of pre-processed training data.

*CountSequence:*

In CountSequence following counts are accumulated:

- number of times it starts with state i
- number of times a particular transition happens
- number of times a particular symbol would be generated from a particular state

For each value in the training file, we do the following:

Step 1: Read the output variable into *sym* variable and the state to which it is tagged into *s* variable.

```
while (stt.hasMoreTokens()) {

    token = stt.nextToken();
     sym=token;

     while(!GlobalVars.Vocab.containsKey(sym))
      {
          sym=stt.nextToken();
      }
      token = stt.nextToken();
    s=Integer.parseInt(token);
```

Figure 6: Reading Training data into variables

Step 2:

After reading into sym and s variables, make corresponding changes in the counts.

Update Parameters:

Using the counts obtained from CountSequence function, compute initial state probabilities, emission probabilities, and transition probabilities.

```
public void UpdateParameter() {

      int i, j;

      for (i=0; i< N; i++) {
        I[i] = (1+ICounter[i])/(N+INorm);

        for (j=0; j<N; j++ ) {
          A[i][j] = (ACounter[i][j])/(N+ANorm[i]);
         System.err.println( i + " " + j + " " + A[i][j]+" \n");
        }

        for (j=0; j<VocabCount; j++) {
          B[j][i] = (BCounter[j][i])/(VocabCount+BNorm[i]);
        }
      }
}
```

Figure 7: Calculating Probabilities.

The output of this phase is the parameters of HMM which are written into a model file. A model file obtained for the training data used in our case looks like the following:

```
Start probabilities:
0 : .250
1 : .200
2 : .150
3 : .150
4 : .100
5 : .150

Transition probabilities:
      0     1     2     3     4     5
0  : .230  .250  .020  .180  .200  .120
1  : .200 .200   .100  .110  .020  .370
… and so on

Output probabilities:
     0    1    2 . . .
0 : .333 .667
1 : .600 .400
```

Figure 8: Model file

The first table in the model file shows the probability of each of the states being a start state. The second table shows the probability of transitioning from each state to every other state. The third table shows the probability of each output symbol seen in each state.

We use the HMM model generated from the training data to compute the most probable sequence of states for a given test sequence, which is a sequence of outputs.

**4.3. Testing phase**

In the testing phase, we evaluate the performance of our model using the test data set from data collection. The goal of this phase is to compute the most probable sequence of states for a given sequence of outputs using the HMM that we built from the training data. This is done by implementing the Viterbi algorithm on the HMM model generated from training data.

Viterbi algorithm will be provided with the output part of the state-output pairs from the test sequences and the HMM model generated from previous step is used to calculate the most likely state sequence to have produced such an output sequence. Estimated state sequence is compared with the state part of the test sequence to evaluate the performance of our model.

In our program, we used the functions ComputeStep and Decode to compute the most probable state sequence matching given observation sequence (given an HMM). A trackback is used to detect the maximum

probability path travelled by the algorithm. The probability of travelling such sequence is also computed in the process.

Implementation:

Step 1:

```
for (int i = 0; i <N; i++) {
    delta[0][i] = -Math.log(I[i]) - Math.log(B[cIndex][i]);
    psy[0][i] = 0;
}
```

Figure 9: Viterbi Algorithm-Initialization-code snippet.

In this step, we initialize variables δ and Ψ stored in delta[T][N] and psy[T][N], respectively.

Step 2:

Compute Step

```
/*
 * Computes delta and psy[t][j] (t > 0)
 */

public void computeStep(String obs, int t, int j) {
    // TODO Auto-generated method stub
    double minDelta = Double.MAX_VALUE;
    int min_psy = 0;

    for (int i = 0; i < N; i++) {
        double thisDelta = delta[t-1][i] - Math.log(A[i][j]);

        if (minDelta > thisDelta) {
            minDelta = thisDelta;
            min_psy = i;
        }
    }
    //char c=obs.charAt(0);
    int cIndex=GlobalVars.Vocab.get(obs);
    delta[t][j] = minDelta - Math.log(B[cIndex][j]);
    System.err.println("delta of ["+t+" ]"+"["+j+"] is....." +delta[t][j] );
    psy[t][j] = min_psy;
}
```

Figure 10: Viterbi Algorithm Compute Step

In this function, we compute δ and Ψ values. Using these values, we compute the most probable state sequence.

Step 3: Termination

```
lnProbability = Double.MAX_VALUE;
for (int i = 0; i < N; i++) {
    double thisProbability = delta[size-1][i];

    if (lnProbability > thisProbability) {
        lnProbability = thisProbability;
        stateSequence[size - 1] = i;
    }
}
lnProbability = -lnProbability;

for (int t2 = size - 2; t2 >= 0; t2--)
    stateSequence[t2] = psy[t2+1][stateSequence[t2+1]];
```

Figure 11: Viterbi Algorithm-termination code snippet

StateSequence array stores the most probable state sequence.

The output of a HMM is a tagged sequence file which looks like this

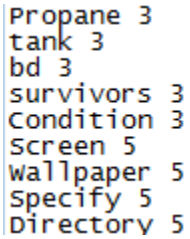


Figure 12: screen shot of tagged sequence file

We conclude this chapter by describing our model in terms of HMM Parameters.

**4.4 Representation of TDT HMM Model**

In HMM for Text Detection and Tracking, the vocabulary is the output variables obtained from the training data after pre-processing. These output variables are written into "OutputVariables.txt" file. So, the vocabulary consists of all the tokens in the *OutputVariables.txt* file.

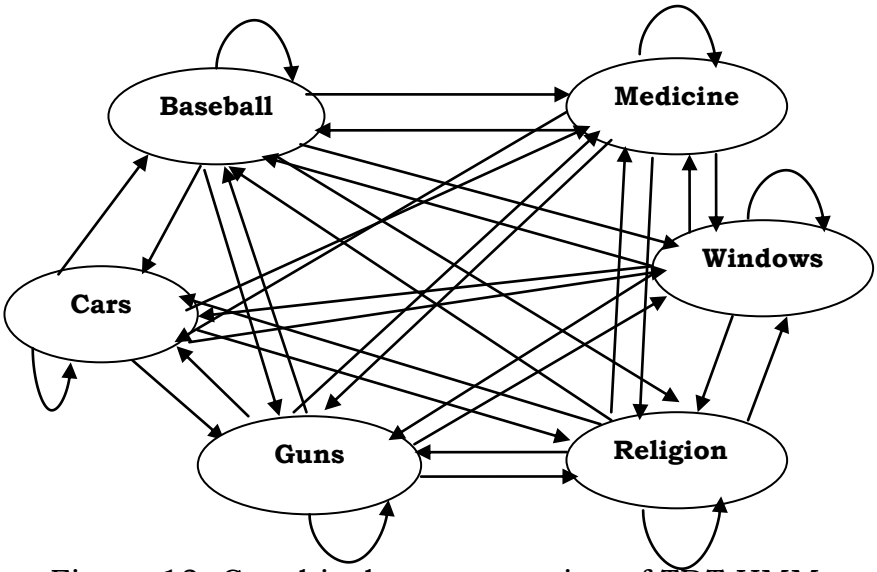

Figure 13: Graphical representation of TDT HMM

The number of states is stored in a variable N. This HMM has 6 states baseball, religion, cars, medicine, guns, and Windows. Each state is represented with a number.

S= { 0, 1, 2, 3, 4, 5, 6 } are the states,

*Baseball is state 0*

Religion is state 1

Cars is state 2

Medicine is state 3

Guns is state 4

Windows is state 5

The 6-state HMM for TDT is represented by the following arrays:

1. Initial state probability (I)

I is an array of length 6 (0-indexed), with I[s] representing the initial probability of the starting the observation sequence with a particular topic (state).

2. State transition probability matrix (A)

   A is a 0-indexed two-dimensional array (6x6), with A[i][j] representing the probability of going from one topic (state j) to the other topic (state i).

3. Output probability matrix (B)

   B is a 0-indexed two-dimensional array (Mx6), with B[o][s] representing the probability of a particular topic(state s) generating output variable "o". M is the number of tokens in the *OutputVariables.txt* file.

Note that the observed variable “o” cannot be used directly as an index to access the entries of matrix B, each output variable is indexed to a number, and the number is used to access the corresponding entries.

### 4.4.1 HMM Representation on the disk file

An HMM can be encoded as a text file. The syntax is very simple.

Below is the model file for the HMM implemented in our thesis.

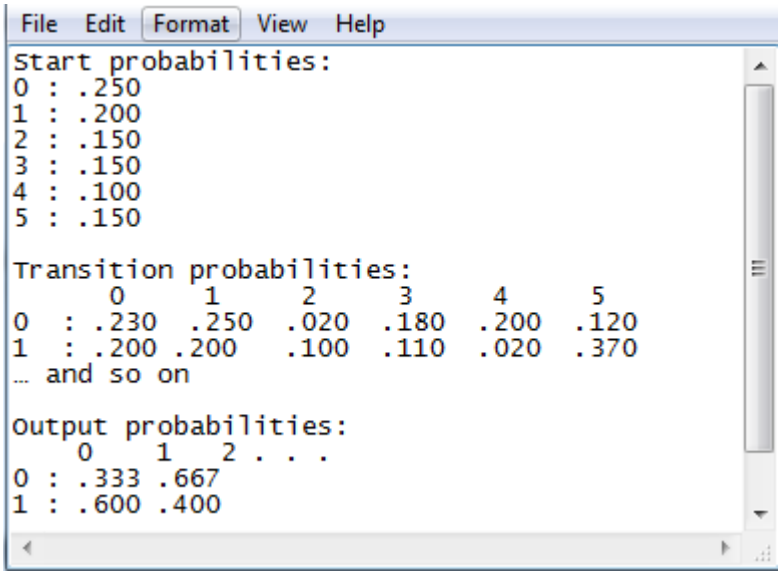


Figure 14: Model file for TDT HMM

In the Model file, the first number is the number of states. The keyword "InitPr" represents Initial Probabilities, "TransPr" represents Transition Probabilites, and "OutputPr" represents Output Emission Probabilities. The number after each keyword is the number of entries following it that should be associated with this keyword.

Model file and sequence files are the inputs and outputs of HMM, respectively.

Model file is generated as the output of the training phase. We use the HMM model generated from the training data to compute the most probable sequence of states for a given test sequence.

# CHAPTER 5

## DATA AND EXPERIMENTAL RESULTS

To perform Text Detection and Tracking, we collected a set of 5302 articles from six newsgroups: rec.sports.baseball, rec.auto, talk.politics.guns, sci.med, talk.religion.misc, and comp.os.ms-windows.misc. They can be categorized into six topics: baseball, cars, guns, medicine, religion, and windows. The training data is further formatted by applying several preprocessing techniques. Preprocessing techniques include the following: parsing the documents to remove all punctuation marks by converting them into white spaces, converting all letters to lower case, converting the text into tokens, removing stop words, and stemming and dimensionality reduction. The resulting documents after preprocessing contain only the output symbols. These output symbols are tagged according to the category to which they belong. The 5302 articles, classified into several sets of categories are then randomly permuted and concatenated together forming a sequence of states (topics) and outputs (words). A state in this problem is an underlying topic. An output is an actual word appearing in the text.

Out of the 5302 documents in the data set,1500 articles were used in testing, the rest in training.

During the testing phase, based on the experience from the training data, the model must track and detect the change of topic in the new unseen stream of text. A model associates terms in the test

document with the most likely states based on Viterbi algorithm described in section 2.

After detecting and tracking the change of topic in test data, we calculate precision and recall values for the outputs generated by the model. Precision and recall are calculated to evaluate the performance of our model. In the following section, we define precision, recall, and F1, then present the results obtained on our test data.

**5.1. Precision and Recall**

The performance of text detection and tracking model built is evaluated based on standard precision, recall, and F1 values. Precision, recall, and F1 values are calculated on the test data from the collection.

Let TP be the number of true positives, i.e., the number of documents that both experts and the model agreed as belonging to the same category. Let FP be the number of false positives, i.e., the number of documents that are wrongly categorized by the model as belonging to that category.

Precision is defined as:

$$precision = \frac{TP}{TP + FP}$$

Precision value obtained for our test data was 95%.

Let FN be the number of false negatives, that is, the number of documents that are not labeled as belonging to the category but should have been.

Recall is defined as:

$$recall = \frac{TP}{TP + FN}$$

Recall value obtained for our test data is 96%

The harmonic mean of precision and recall is called the F1 measure, and is defined as [24]:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

F1 value obtained for our test data was 95 %.

# CHAPTER 6

## CONCLUSION AND FUTURE WORK

In this thesis research, we have demonstrated the effectiveness of HMMs for topic tracking and segmentation. After running our model on a collection of 1500 test documents, the precision and recall values were calculated as reported in Chapter 5. From these values, it can be concluded that this is an efficient tool to perform Topic Detection and Tracking. We used smoothing to reduce the problems caused by sparse training data. We used scaling to handle multiplication of large observation sequences that helped improve the performance of HMM. So, we can conclude that our HMM can handle issues, like large observation sequences as well as sparse training data.

In this thesis, we worked on news articles from newsgroups. As a future work, one can evaluate performance of HMM using speech recognition output. We used a small training data set of 5036 documents. This work can be extended by training and testing the model built on large document collections determining their precision and recall values. Also, this model can be compared with the various Text Detection and Tracking tools that are available to determine the models that perform better in a commercial environment.

BIBLIOGRAGHY


1. Wikipedia, the free Encyclopedia, Data Mining, http://en.wikipedia.org/wiki/Data_mining
2. Machine Learning. The Free On-line Dictionary of computing http://encyclopedia.thefreedictionary.com/Machine+learning+algorithm
3. Wikipedia, the free Encyclopedia, Machine Learning, http://en.wikipedia.org/wiki/Machine_learning
4. Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition
5. Topic Detection and Tracking by. Omid Dadgar
6. Martin Porter, 'The Porter Stemming Algorithm'
7. http://tartarus.org/~martin/PorterStemmer/
8. Yiming Yang, Jan O. Pederson, 'A comparative study on feature selection in text categorization'. http://net.pku.edu.cn/~course/cs502/2003/031119/yang97comparative.pdf
9. Hidden Markov Models by Marc Sobel
10. Wikipedia, the free Encyclopedia, Precision and Recall, http://en.wikipedia.org/wiki/Precision_and_recall
11. Hidden Markov Models by John Fry, San Jose State University
12. A Brief Note on the Hidden Markov Models (HMMs) by ChengXiang Zhai

13. Wikipedia, the free Encyclopedia, Hidden Markov Models, http://en.wikipedia.org/wiki/Hidden_Markov_model

VITA

Graduate College
University of Nevada, Las Vegas

Aditya Sowmya Tatavarty

Degrees:

Bachelor of Technology, Computer Science, 2009
Jawaharlal Nehru Technological University.

Thesis Title: Topic Detection and Tracking Using Hidden Markov Models

Thesis Examination Committee:

Chairperson, Dr. Kazem Taghva, Phd.
Committee Member, Dr. Ajoy K. Datta, Phd.
Committee Member, Dr. Laxmi P. Gewali, Phd.
Graduate College Representative, Dr. Venkatesan Muthukumar, Phd.