

5-2011

## Using smoothing techniques to improve the performance of Hidden Markov's Model

Sweatha Boodidhi  
University of Nevada, Las Vegas

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

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

---

### Repository Citation

Boodidhi, Sweatha, "Using smoothing techniques to improve the performance of Hidden Markov's Model" (2011). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 1007.  
<https://digitalscholarship.unlv.edu/thesesdissertations/1007>

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](mailto:digitalscholarship@unlv.edu).

USING SMOOTHING TECHNIQUES TO IMPROVE THE PERFORMANCE  
OF HIDDEN MARKOV'S MODEL

by

Sweatha Boodidhi

Bachelor of Technology  
Jawaharlal Nehru Technological University, India  
May 2007

A thesis submitted in partial fulfillment  
of the requirements for the

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

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

Copyright by Sweatha Boodidhi 2011  
All Rights Reserved



**THE GRADUATE COLLEGE**

We recommend the thesis prepared under our supervision by

**Sweatha Boodidhi**

entitled

**Using Smoothing Techniques to Improve the Performance of Hidden Markov's Model**

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

**Master of Science in Computer Science**

School of Computer Science

Kazem Taghva, Committee Chair

Ajoy K. Datta, Committee Member

Laxmi P Gewali, Committee Member

Venkatesan Muthukumar, Graduate Faculty Representative

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

**May 2011**

ABSTRACT

**Using Smoothing Techniques to Improve the Performance of  
Hidden Markov's Models**

by

Sweatha Boodidhi

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

The result of training a HMM using supervised training is estimated probabilities for emissions and transitions. There are two difficulties with this approach Firstly, sparse training data causes poor probability estimates. Secondly, unseen probabilities have emission probability of zero. In this thesis, we report on different smoothing techniques and their implementations. We further report on our experimental results using standard precision and recall for various smoothing techniques.

## ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Kazem Taghva , for all his help and support in my pursuit of a Master's degree. Throughout this process he has provided support and encouragement in my thesis and my studies. His advice and enthusiasm were critical to the success of this effort. I would also like to thank Dr. Ajoy K Datta for serving on my advisory Committee, as well as Graduate coordinator and for his time in reviewing the prospectus and his help, advice and guidance during my course work.

I would also like to thank Dr. Laxmi P. Gewali and Dr. Venkatesan Muthukumar for their time in reviewing the prospectus, participation in defense, and counseling of the thesis as the committee members.

I would also like to thank my husband Venkat Mudupu whose suggestions and advices have been great help for me.

I would also like to express my heartiest gratitude to my parents for their unconditional support, love, and affection. A special thanks to my friends for their unrelenting support and motivation throughout this research activity.

Finally, I would like to thank Mario Martin and Sharron, staff of the Computer Science Engineering department, for providing me with all resources for my defense.

## TABLE OF CONTENTS

ABSTRACT .....	iii
ACKNOWLEDGEMENTS .....	iv
TABLE OF CONTENTS.....	v
LIST OF FIGURES .....	vii
CHAPTER 1 INTRODUCTION .....	1
1.1 Thesis Overview.....	3
CHAPTER 2 HIDDEN MARKOV MODEL.....	4
2.1 Definition of HMM .....	4
2.1.1 Examples on HMM .....	5
2.2 Main Issues Using HMM.....	6
2.2.1 Evaluation Problem .....	7
2.2.1.1 Forward recursion for HMM.....	8
2.2.1.2 Backward recursion for HMM .....	9
2.2.2 Decoding Problem.....	10
2.2.2.1 Viterbi algorithm .....	10
2.2.3 Learning Problem .....	12
2.2.3.1 Maximum Likelihood Estimation .....	13
CHAPTER 3 SMOOTHING .....	14
3.1 What is Smoothing .....	14
3.2 Why Smoothing is used in HMM? .....	14
3.2.1 Where we use Smoothing in HMM?.....	14
3.2.2 Maximum Likelihood Estimation.....	15
3.3 How Smoothing works in HMM.....	16
3.3.1 Examples.....	16
3.3.1.1 Example 1 .....	16
3.3.1.2 Example 2 .....	16
3.3.1.3 Example 3 .....	17
3.4 Smoothing Techniques .....	17
3.4.1 Absolute Discounting.....	17
3.4.2 Laplace Smoothing .....	18
3.4.3 Good-Turing Estimation .....	18
3.4.4 Shrinkage .....	19
CHAPTER 4 IMPLEMENTATION.....	21
4.1 Supervised Learning.....	21
4.1.1 Maximum Likelihood Estimation (MLE).....	21
4.2 Laplace Smoothing .....	23
4.3 Absolute Discounting.....	25

CHAPTER 5 RESULTS .....	29
5.1 Using HMM Model .....	29
5.1.1 HMM Model How it Looks .....	29
5.1.2 Results on HMM and How it Works.....	30
5.2 Comparison of Two Smoothing Techniques .....	32
5.2.1 Equation on MLE.....	32
5.2.2 Equation on Laplace Smoothing.....	32
5.2.3 Equation on Absolute Discounting.....	32
5.3 Results on Laplace Smoothing .....	33
5.4 Results on Absolute Discounting .....	34
5.5 Precision is defined as .....	36
5.6 Recall is defined as .....	36
5.7 Harmonic Mean .....	36
5.8 Results on Evaluation.....	36
5.8.1 Without Using Smoothing Techniques.....	36
5.8.2 Using Smoothing Techniques.....	37
5.8.2.1 Laplace Smoothing .....	37
5.8.2.2 Absolute Discounting .....	37
CHAPTER 6 CONCLUSIONS AND FUTURE WORK.....	38
BIBLIOGRAPHY.....	39
VITA.....	40



## LIST OF FIGURES

Figure 1 Example on Hidden Markov's Model.....	5
Figure 2 Trellis Representation of HMM .....	7
Figure 3 Forward Recursion of HMM.....	8
Figure 4 Backward Recursion of HMM .....	9
Figure 5 Viterbi Algorithm.....	11
Figure 6 Algorithm Counting for in MLE .....	23
Figure 7 Screen shot on Laplace smoothing .....	24
Figure 8 Screen Shot on Absolute Discounting.....	25
Figure 9 Screen Shot on Absolute Discounting.....	26
Figure 10 Screen Shot on Absolute Discounting.....	26
Figure 11 Screen Shot on Absolute Discounting.....	27
Figure 12 Screen Shot on Absolute Discounting.....	27
Figure 13 Screen Shot on Absolute Discounting.....	28
Figure 14 HMM Model .....	29
Figure 15 HMM Model Data .....	30
Figure 16 HMM Train Data .....	31
Figure 17 Laplace Smoothing Result .....	33
Figure 18 Laplace Smoothing Result .....	34
Figure 19 Laplace Smoothing Result .....	34
Figure 20 Absolute Discounting Result .....	35

## CHAPTER 1

### INTRODUCTION

Hidden Markov's Model (HMM) is a directed graph, with probability weighted edges (representing the probability of a transition between the source and sink states) where each vertex emits an output symbol when entered. HMM can be trained using both supervised training and unsupervised training methods. The supervised training uses MLE (Maximum Likelihood Estimation) and unsupervised training uses Baum-Welch algorithm.

Supervised training is a training method which estimates both output symbols and states sequences. While doing supervised training using MLE we face some difficulties. Problems that occur are, Maximum Likelihood Estimates (MLE) will sometimes assign a zero probability to unseen emission-state combinations. Also, when the training data is sparse we cannot obtain good probably estimates. To avoid such situations we use Smoothing techniques.

Take an example of flipping a coin (Heads (H), Tails (T)). The probability of heads (H) is  $p$ , where  $p$  is an unknown and our goal is to estimate  $p$ .

The obvious approach is to count how many times the coin came up heads (H) and divide by the total no. of coin flips.

$$p=H/N$$

$$H=Heads$$

$$N=Total\ number\ of\ coin\ flips$$

If we flip the coin 1000 times and it comes up heads (H) 367 times and tails (T) 633 times, it is very reasonable to estimate  $p$  as approximately 0.367.

$$p=367/1000=0.367$$

Suppose we flip the coin only twice and we get heads (H) both times.

So  $H=2$  and  $T=0$  then it is reasonable to estimate  $p$  as 1.0.

$$p=2/2=1.$$

The  $p$  above is not a good probability estimates. According to the above estimate the probability of Tail showing up when a coin is tossed is zero.

To solve this problem we use different smoothing techniques

Here in this thesis we will see the different smoothing techniques and their effect on the performance on HMM.

The uses of smoothing techniques in HMM are when we train a HMM using sparse training data, there is no abundant training data and have some limited probability estimates for hidden words that have emission probabilities of zero. Smoothing techniques in HMM will be used to deal these issues. Smoothing is used to deal with the problem of zero probabilities that occur due to sparse training data. The term smoothing describes techniques for adjusting the maximum likelihood estimate of probabilities to produce more accurate probabilities. The name smoothing comes from the fact that these techniques tend to make distributions more uniform, by adjusting low probabilities such as zero probabilities upward, and high probabilities downward. Not only do

smoothing methods generally prevent zero probabilities, but they also attempt to improve the accuracy of the model as a whole. Whenever a probability is estimated from few counts, smoothing has the potential to significantly improve estimation [1].

Smoothing is the process of flattening probability distribution so that all word sequences can occur with some probability. This often involves redistributing weight from high probability regions to zero probability regions.

### **1.1 Thesis Overview**

This thesis is organized as follows to present the details of HMM, Smoothing Techniques, which algorithm work well in which situations, and why and conclusion on current work. Chapter 2 provides the background of HMM and algorithms used in this work. Chapter 3 presents about the Smoothing Techniques, different techniques used in this work and clear explanation about the Smoothing techniques. Chapter 4 discuss about the implementation and usage of algorithms in appropriate situations. Chapter 5 presents the results on different smoothing techniques. Chapter 6 provides conclusions about the present work and recommendations on future work are discussed.

## CHAPTER 2

### HIDDEN MARKOV MODEL

Hidden Markov Models (HMM) are powerful statistical models for modeling sequential or time-series data, and have been successfully used in many tasks such as speech recognition, protein/DNA sequence analysis, robot control, and information extraction from text data [2].

The Hidden Markov's Model (HMM) in abbreviation are called 3D three dimensional.

#### 2.1 Definition of HMM

“The structure of an HMM model contains states and observations. We define HMM as a 5-tuple ( S, V,  $\Pi$ , A, B ), where  $S=\{s_1, \dots, s_N\}$  is a finite set of N states,  $V=\{v_1, \dots, v_M\}$  is a set of M possible symbols in a vocabulary,  $\Pi=\{\Pi_i\}$  are the initial state probabilities,  $A=\{a_{ij}\}$  are the state transition probabilities,  $B=\{ b_{ik}(v_k) \}$  are the output or emission probabilities. We use  $\lambda=(\Pi, A, B)$  to denote all the parameters”[2].

$\Pi_i$  the probability that the system starts at state i at the beginning

$a_{ij}$  the probability of going to state j from state i

$b_i(v_k)$  the probability of “generating” symbol  $v_k$  at state i

clearly, we have the following constraints

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

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

$$\sum_{k=1}^M b_i(v_k) = 1 \text{ for } i = 1, 2, \dots, N$$

### 2.1.1 Examples on HMM

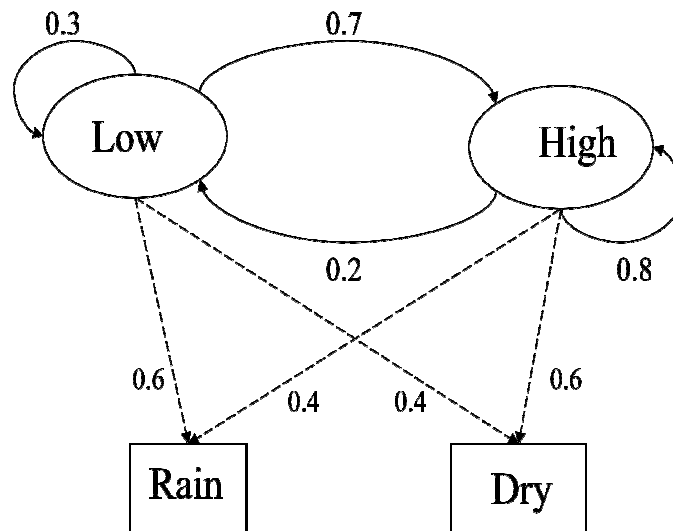


Figure 1 Example on Hidden Markov's Model

The above example model has 2 states, Low and High atmosphere weather and 2 observations Rain and Dry.

Transition probabilities are

$$P(\text{'Low' / 'Low'})=0.3$$

$$P(\text{'High' / 'Low'})=0.7$$

$$P(\text{'Low' / 'High'})=0.2$$

$$P(\text{'High' / 'High'})=0.8$$

Observation Probabilities are

$$P(\text{'Rain' / 'Low'})=0.6$$

$$P(\text{'Dry' / 'Low'})=0.4$$

$$P(\text{'Rain' / 'High'})=0.4$$

$$P(\text{'Dry' / 'High'})=0.3$$

Initial Probabilities are

$$P(\text{'Low'})=0.4$$

$$P(\text{'High'})=0.6$$

Calculation of observation sequence probability

Suppose we want to calculate a probability of a sequence Observations in our example, {'Dry','Rain'}

Consider all possible hidden state sequences

$$P(\{\text{'Dry' , 'Rain'}\})=P(\{\text{'Dry' , 'Rain'}\},\{\text{'Low' , 'Low'}\})+P(\{\text{'Dry' , 'Rain'}\},\{\text{'Low' , 'High'}\})+P(\{\text{'Dry' , 'Rain'}\},\{\text{'High' , 'Low'}\})+P(\{\text{'Dry' , 'Rain'}\},\{\text{'High' , 'High'}\})$$

Where first term is :

$$P(\{\text{'Dry' , 'Rain'}\},\{\text{'Low' , 'Low'}\})=P(\{\text{'Dry' , 'Rain'}\} | \{\text{'Low' , 'Low'}\})$$

$$P(\{\text{'Low' , 'Low'}\})=P(\text{'Dry' | 'Low'})$$

$$P(\text{'Low'})P(\text{'Low' | 'Low'})=0.4*0.4*0.6*0.4*0.3$$

$$=0.01152$$

## 2.2 Main Issues Using HMM

There are three main problems

1. Evaluation Problem
2. Decoding
3. Training

### 2.2.1 Evaluation Problem

The HMM  $\lambda = (\Pi, A, B)$  and the observation sequence  $O = o_1 o_2 \dots o_K$ , calculate the probability that model  $\lambda$  has generated sequence  $O$  [2].

Here we try to find the probability of an observation sequence  $O = o_1 o_2 \dots o_K$  by means of considering all hidden state sequences.

For solving evaluation problem, we use Forward and Backward iterative algorithms for efficient calculations. Here we have to calculate individual algorithms like

Forward Evaluation

Backward Evaluation

Define the forward variable  $\alpha_k(i)$  as the joint probability of the partial observation sequence  $o_1 o_2 \dots o_k$  and that the hidden state at time  $k$  is  $s_i$  :

$$\alpha_k(i) = P(o_1 o_2 \dots o_k, q_k = s_i) \text{ [5].}$$

### Trellis representation of an HMM

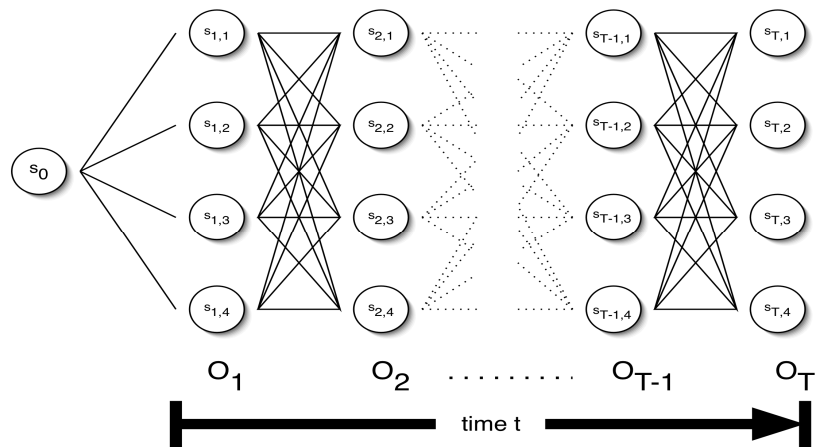


Figure 2 Trellis Representation of HMM



### 2.2.1.1 Forward recursion for HMM

Initialization:

$$\alpha_1(i) = P(o_1, q_1 = s_i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N.$$

Forward recursion:

$$\alpha_{k+1}(i) = P(o_1 o_2 \dots o_{k+1}, q_{k+1} = s_i) = \sum_i P(o_1 o_2 \dots o_{k+1}, q_k = s_i, q_{k+1} = s_j) =$$

$$\sum_i P(o_1 o_2 \dots o_k, q_k = s_i) a_{ij} b_j(o_{k+1}) = [\sum_i \alpha_k(i) a_{ij}] b_j(o_{k+1}),$$

For  $1 \leq j \leq N, 1 \leq k \leq K-1.$

Termination:

$$P(o_1 o_2 \dots o_K) = \sum_i P(o_1 o_2 \dots o_K, q_K = s_i) = \sum_i \alpha_K(i)$$
 [5]page 262-263 [2]page 2

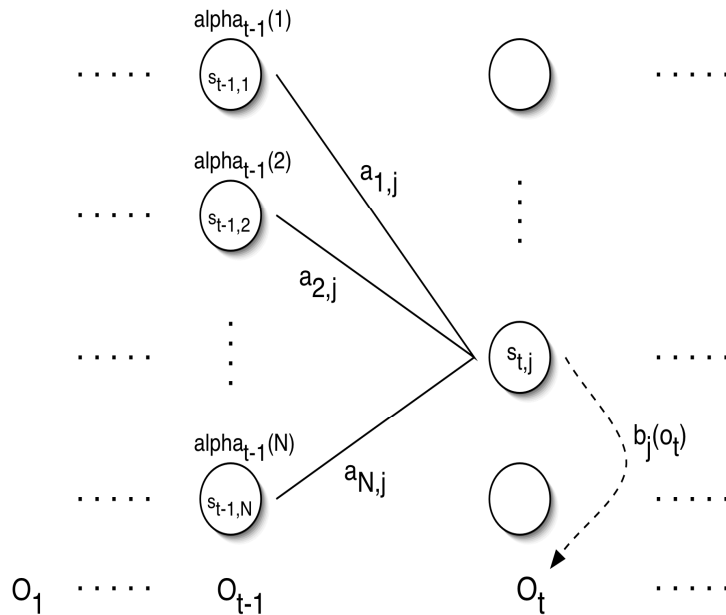


Figure 3 Forward Recursion of HMM

### 2.2.1.2 Backward recursion for HMM

Define the forward variable  $\beta_k(i)$  as the joint probability of the partial observation sequence  $o_{k+1} o_{k+2} \dots o_K$  given that the hidden state at time  $k$  is  $s_i$  :  $\beta_k(i) = P(o_{k+1} o_{k+2} \dots o_K | q_k = s_i)$

Initialization:

$$\beta_K(i) = 1, \quad 1 \leq i \leq N.$$

Backward recursion:

$$\begin{aligned} \beta_k(j) &= P(o_{k+1} o_{k+2} \dots o_K | q_k = s_j) = \sum_i P(o_{k+1} o_{k+2} \dots o_K, q_{k+1} = s_i | q_k = s_j) \\ &= \sum_i P(o_{k+2} o_{k+3} \dots o_K | q_{k+1} = s_i) a_{ji} b_i(o_{k+1}) = \sum_i \beta_{k+1}(i) a_{ji} b_i(o_{k+1}), \end{aligned}$$

For  $1 \leq j \leq N, 1 \leq k \leq K-1$

Termination:

$$\begin{aligned} P(o_1 o_2 \dots o_K) &= \sum_i P(o_1 o_2 \dots o_K, q_1 = s_i) = \sum_i P(o_1 o_2 \dots o_K | q_1 = s_i) P(q_1 = s_i) \\ &= \sum_i \beta_1(i) b_i(o_1) \pi_i \quad [5] \text{page 262-263, } [2] \text{page 3} \end{aligned}$$

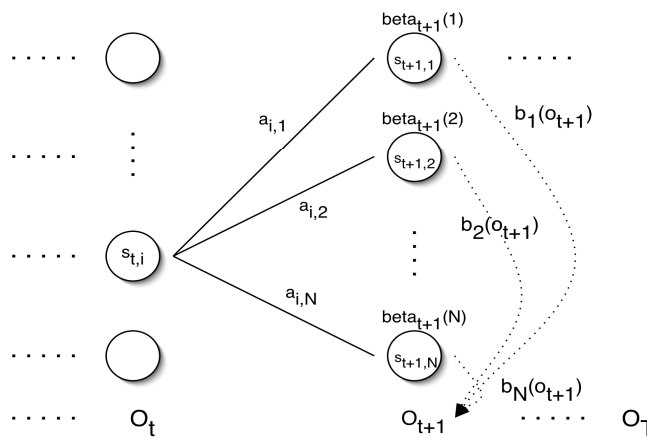


Figure 4 Backward Recursion of HMM

### 2.2.2 Decoding Problem

Decoding problem. Given the HMM  $\lambda = (\Pi, A, B)$  and the observation sequence  $O = o_1 o_2 \dots o_K$ , calculate the most likely sequence of hidden states  $s_i$  that produced this observation sequence.

We want to find the state sequence  $Q = q_1 \dots q_K$  which maximizes  $P(Q | o_1 o_2 \dots o_K)$ , or equivalently  $P(Q, o_1 o_2 \dots o_K)$ .

Brute force consideration of all paths takes exponential time. To solve this issue we can use dynamic programming (DP) techniques that optimize the entire process. Viterbi is one such efficient algorithm that uses DP and reduces exponential time to linear.

Define variable  $\delta_k(i)$  as the maximum probability of producing observation sequence  $o_1 o_2 \dots o_k$  when moving along any hidden state sequence  $q_1 \dots q_{k-1}$  and getting into  $q_k = s_i$ .

$$\delta_k(i) = \max P(q_1 \dots q_{k-1}, q_k = s_i, o_1 o_2 \dots o_k)$$

Where max is taken over all possible paths  $q_1 \dots q_{k-1}$ .

#### 2.2.2.1 Viterbi algorithm

General idea if best path ending in  $q_k = s_j$  goes through  $q_{k-1} = s_i$  then it should coincide with best path ending in  $q_{k-1} = s_i$ .

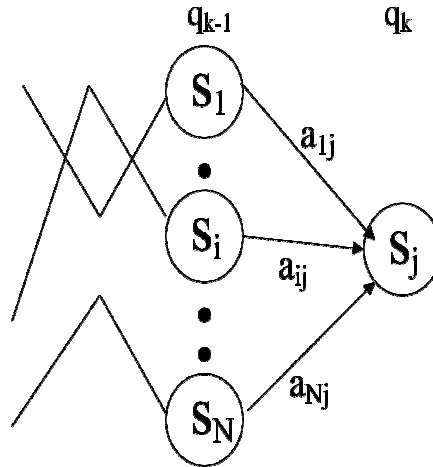


Figure 5 Viterbi Algorithm

$$\delta_k(i) = \max P(q_1 \dots q_{k-1}, q_k = s_j, o_1 o_2 \dots o_k) =$$

$$\max_i [ a_{ij} b_j(o_k) \max P(q_1 \dots q_{k-1} = s_i, o_1 o_2 \dots o_{k-1}) ]$$

For backtracking best path keep information that predecessor of  $s_j$  was  $s_i$

Initialization:

$$\delta_1(i) = \max P(q_1 = s_i, o_1) = \pi_i b_i(o_1), \quad 1 \leq i \leq N.$$

Forward recursion:

$$\delta_k(j) = \max P(q_1 \dots q_{k-1}, q_k = s_j, o_1 o_2 \dots o_k) = \max_i [ a_{ij} b_j(o_k) \max P(q_1 \dots q_{k-1} = s_i, o_1 o_2 \dots o_{k-1}) ] = \max_i [ a_{ij} b_j(o_k) \delta_{k-1}(i) ], \quad 1 \leq j \leq N, \quad 2 \leq k \leq K.$$

Termination: choose best path ending at time K

$$\max_i [ \delta_K(i) ]$$

Backtracking is the best path.

This algorithm is similar to the forward recursion of evaluation problem, with  $\Sigma$  replaced by max and additional backtracking [7]

### 2.2.3 Learning Problem

In learning problem we have both supervised training and unsupervised training. Supervised training means MLE (Maximum Likelihood Estimation), unsupervised training means Baum-Welch Algorithm.

Maximum likelihood estimation in hidden Markov models was first investigated by Baum and Petrie [BP66] for finite signal and observation states spaces.[9]

MLE is a solid tool for learning parameters of a data mining model. It is a methodology which tries to do two things. First, it is a reasonably well-principled way to work out what computation you should be doing when you want to learn some kinds of model from data. Second, it is often fairly computationally tractable. In any case, the important thing is that in order to understand things like Hidden Markov Models and many other things it's going to really help if you're happy with MLE.

Learning problem given some training observation sequences  $O = o_1 o_2 \dots o_K$  and general structure of HMM (numbers of hidden and visible states), determine HMM parameters  $\lambda = (\Pi, A, B)$  that best fit training data, that is maximizes  $P(O | \lambda)$ .

There is no algorithm producing optimal parameter values. Use iterative expectation-maximization algorithm to find local maximum of  $P(O | \lambda)$  - Baum-Welch algorithm.

### **2.2.3.1 Maximum Likelihood Estimation**

If training data has information about sequence of hidden states (as in word recognition example), then use maximum likelihood estimation of parameters.[6]

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

We use maximum likelihood in our thesis.

## CHAPTER 3

### SMOOTHING

#### **3.1 What is Smoothing**

In general Smoothing is just a mathematical technique that removes the excess data variability while maintaining a correct appraisal and smoothing is a data set  $\{X_i, Y_i\}$  when it takes the approximation  $m()$  in a growth such as  $Y_i = m(X_i) + e_i$  and estimated result on smoothing is a smooth functional estimates  $m()$ .

Smoothing is the process of flattening probability distribution so that all word sequences can occur with some probability. This often involves redistributing weight from high probability regions to zero probability regions.

#### **3.2 Why Smoothing is used in HMM?**

Smoothing is used to improve the probability estimates.

##### **3.2.1 Where we use Smoothing in HMM?**

The objective of learning is to give high probabilities in training documents and the result of learning is estimated probabilities for vocabularies and transition. Also, we face some difficulties when sparse training data causes poor probabilities estimates. Unseen words have emission probabilities of zero.

“Whenever data sparsity is an issue, smoothing can help performance, and data sparsity is almost always an issue in statistical modeling. In the extreme case where there is so much training data that all parameters

can be accurately trained without smoothing, one can almost always expand the model, such as by moving to a higher n-gram model, to achieve improved performance. With more parameters data sparsity becomes an issue again, but with proper smoothing the models are usually more accurate than the original models. Thus, no matter how much data one has, smoothing can almost always help performance, and for a relatively small effort.” Chen & Goodman (1998)[1]

Smoothing is required in maximum likelihood estimation because MLE will sometimes assign a ‘0’ probability to unseen emission state combination.

### **3.2.2 Maximum Likelihood Estimation**

Maximum Likelihood Estimation trains a data in HMM. Maximum Likelihood will estimate a transition and emission probabilities are [6]

$$P(w/s)_{ml} = (N(w, s)) / (N(s))$$

$N(w, s)$  =# of times symbols  $w$  is emitted at state  $s$

$N(s)$  =Total # of symbols emitted by state  $s$ .

Let see an example on MLE on flipping a coin Heads (H) , Tails (T) .

If we flip a coin twice and head show up twice.

$$P(\text{Head})_{ml} = 2 / 2 = 1.0$$

$$P(\text{Tail})_{ml} = 0 / 2 = 0$$

For reducing zero probability for unseen emission state combination we use smoothing.



### **3.3 How Smoothing works in HMM**

Smoothing will make certain estimates. An example is provided below to explain what they are and how smoothing works in HMM.

#### **3.3.1 Examples**

##### **3.3.1.1 Example 1**

Flipping a coin Heads (H), Tails (T) for which the probability of heads is  $p$ , where  $p$  is unknown, and our goal is to estimate  $p$ .

The obvious approach is to count how many times the coin came up heads and divide by the total number of coin flips. If we flip the coin 1000 times and it comes up Heads 367 times, and Tails 633 times, it is very reasonable to estimate  $p$  as approximately

$$p=H/N$$

$$H=\text{Heads}$$

$$N=\text{Total number of flip coins}$$

$$p=367/1000=0.367.$$

##### **3.3.1.2 Example 2**

Again if we flip the coin only twice and we get heads both times.

$$H=2$$

$$T=0$$

The approximate estimate value of  $p$  is

$$P=2/2=1.0.$$

$$P=0/2=0.$$

### 3.3.1.3 Example 3

Again if we flip a coin only twice it seems a bit rash to conclude that the coin will always come up Heads and for avoiding such rash we use smoothing

To solve this sparseness problem, there are many different smoothing techniques.

### 3.4 Smoothing Techniques

1. Absolute Discounting
2. Laplace Smoothing
3. Good-Turing Estimation
4. Shrinkage

#### 3.4.1 Absolute Discounting

We used absolute discounting to smooth emission probabilities. Absolute discounting consists of subtracting a small amount of probability  $p$  from all symbols assigned a non zero probability at states  $s$ . Probability  $p$  is then distributed equally over symbols given zero probability by the MLE.

If  $v$  is number of symbols assigned non zero probability at a state  $s$  and  $N$  is the total number of symbols. [6]

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

For determining the optimal value  $p$  in using  $(1 / (T_s + v))$

Where  $T_s$  is the total number of symbols emitted by a state  $s$  (i.e) the denominator of  $p(w / s)_{ml}$ .

### 3.4.2 Laplace Smoothing

It is also known as Add-One Smoothing, In Laplace Smoothing we have to add some of the probabilities for unseen events

Take an example of flipping a coin. If we flip the coin twice and count the number of Heads (H) and Tails (T), if heads come up both the times the the probability for tails is zero. To avoid such situations we use smoothing. To estimate the value p in Laplace Smoothing we have to

$$\text{estimate } p = \frac{(1+H)}{(\text{total number of flips} + |V|)}$$

$$P = (1+2) / (2+2) = 0.75$$

This rule is equivalent to starting each of our counts at one rather than 0 this is known as Laplace smoothing.

To avoid estimating any probabilities to be zero for events never observed in the data we do the following in Laplace smoothing

$$P(w / s)_{lap} = (N(w, s) + 1) / (N(s) + |V|)$$

where  $|V|$  is the vocabulary size.

$N(w,s)$ =number of times symbols w is emitted at state s

$N(s)$  =Total number of symbols emitted by state s.

### 3.4.3 Good-Turing Estimation

The Good-Turing estimate (Good, 1953) is central to many smoothing techniques. The general idea of the good turing is reallocate the probability mass of n-grams that occurs c times.

For each count c, we should pretend that it occurs  $c^*$  times

$$C^* = (C + 1) \frac{N_c + 1}{N_c}$$

Where  $N_c$  is the number of n-grams that occurs exactly  $c$  times in the training data.

$$P_{GT}(w_1, \dots, w_n) = \frac{C^*(w_1, \dots, w_n)}{N}$$

$N$  = the original number of counts in the distribution.

$$N = \sum_{c=0}^{\infty} N_c C^* = \sum_{c=0}^{\infty} (c + 1) n_{c+1} = \sum_{c=1}^{\infty} c n_c \quad [1] \text{ page 8-9}$$

The Good-Turing estimate cannot be used when  $n_c = 0$ ; it is generally necessary to "smooth" the  $n_c$ .

Example, to adjust the  $n_c$  so that they are all above zero. Recently, Gale and Sampson (1995) have proposed a simple and effective algorithm for smoothing these values. In practice, the Good-Turing estimate is not used by itself for n-gram smoothing, because it does not include the combination of higher-order models with lower-order models necessary for good performance. However, it is used as a tool in several smoothing techniques.

#### 3.4.4 Shrinkage

The Shrinkage is the distribution of a state data towards more rich data and it is used for a linear combination of probabilities

$$p(W / S_j) = \sum_{i=1}^k \lambda_j^i P(W / S_j^i)$$

$$p(W / S) = \lambda_1 p(W / S_1) + \lambda_2 p(W / S_2) + \dots$$

Where  $S_i$  is the original state.

$j$  is the state and  $i$  is the shrinkage ancestor

$S_2$  is the larger context.

$\lambda$ =shrinkage prior

In smoothing techniques the range of the shrinkage influence is when it is used for context distributions not only towards those states but also towards similar states with more data. They are three variants of shrinkage used in smoothing techniques.

## CHAPTER 4

### IMPLEMENTATION

In thesis implementation we first discuss about the MLE. Before telling about the MLE we will learn supervised learning.

#### **4.1 Supervised Learning**

The easiest solution for creating a model  $\lambda$  is to have a large corpus of training examples, each annotated with the correct classification. If we having 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  $\lambda$ .

##### **4.1.1 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.

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

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

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 a HMM from counts it is normally 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.

In the thesis we implemented MLE using the function:

```
void CountSequence(char *seqFile); and
```

Parameters : tagged sequence file

Implementation of MLE involves accumulating the following counts

- count how many times it starts with state  $s_i$
- count how many times a particular transition happens
- count how many times a particular symbol would be generated from a particular state
- Implementation of MLE is shown in the following fig

```

// Supervised training: You need to finish six assignment statements corresponding to
// counting three different events.
void Model::CountSequence(char *seqFile)
{
    char c; // to store the symbol
    int s; // to store the tagged state
    int prevState= -1;
    bool first=true;
    while (ifs >> c >> s) {
        int cIndex = c-baseChar; // convert the character to an index that can be used for BCounter.
        if (first) { // this is the very first observed symbol

            ICounter[s] +=1 ;
            INorm += 1;

            first = false;
        }
        BCounter[cIndex][s] += 1;
        BNorm[s] += 1;

        if (prevState>=0) { // the current symbol is not the first one; prevState has the previous state.

            // the state transition event,
            ACounter[prevState][s] +=1;
            ANorm[prevState] +=1;
        }
        prevState =s;
    }
}

```

Figure 6 Algorithm Counting for in MLE

Using these count relative frequencies are computed to obtain parameters of an HMM.

## 4.2 Laplace Smoothing

In Laplace smoothing we avoid zero probabilities for unseen events by calculating the probability estimates using the following equations

Equation for smoothing emission probabilities

$$P(w / s)_{lap} = (N(w, s) + 1) / (N(s) + |V|)$$

where  $|V|$  is the vocabulary size.

$N(w, s)$ =number of times symbols  $w$  is emitted at state  $s$

$N(s)$  =Total number of symbols emitted by state  $s$ .



Example:

If two times you toss a coin and head shows up twice

$$P(\text{Head})_{\text{lap}} = (2+1) / (2+2) = 0.75$$

$$P(\text{Tail})_{\text{lap}} = (0+1)/(2+2) = 0.25$$

Equation for transition probabilities

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

$N(s_i)$ : Number of transitions from state  $s_i$

$V$ : entire vocabulary (all output symbols)

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

In this thesis we implement Laplace Smoothing using function.

`void Model::UpdateParameter()`. Implementation of this function shown

below

Implementation on Laplace smoothing is show in figure.

```
129 // the HMM parameters
130 void Model::UpdateParameter()
131 {
132
133     int i, j;
134     double sumB=0;
135     for (i=0; i< N; i++) {
136         I[i] = (1+ICounter[i])/(N+INorm);
137         cout <<"\n I of "<< i << " " <<I[i];
138         for (j=0; j<N; j++) {
139             A[i][j] = (1+ACounter[i][j])/(N+ANorm[i]);
140             cout <<"\n A of "<< i << j << " " <<A[i][j];
141         }
142         sumB=0;
143         for (j=0; j<SYMMNUM; j++) {
144             B[j][i] = (1+BCounter[j][i])/(SYMMNUM+BNorm[i]);
145             sumB = sumB + B[j][i];
146             cout <<"\n B of "<< j << i << " " <<B[j][i];
147         }
148         cout <<"\n sum B of state"<< i << " " << sumB;
149     }
150 }
```

Figure 7 Screen shot on Laplace smoothing

### 4.3 Absolute Discounting

We used absolute discounting to smooth emission probabilities. Absolute discounting consists of subtracting a small amount of probability  $p$  from all symbols assigned a non zero probability at states  $s$ . Probability  $p$  is then distributed equally over symbols given zero probability by the MLE.

If  $v$  is number of symbols assigned non zero probability at a state  $s$  and  $N$  is the total number of symbols. [6]

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

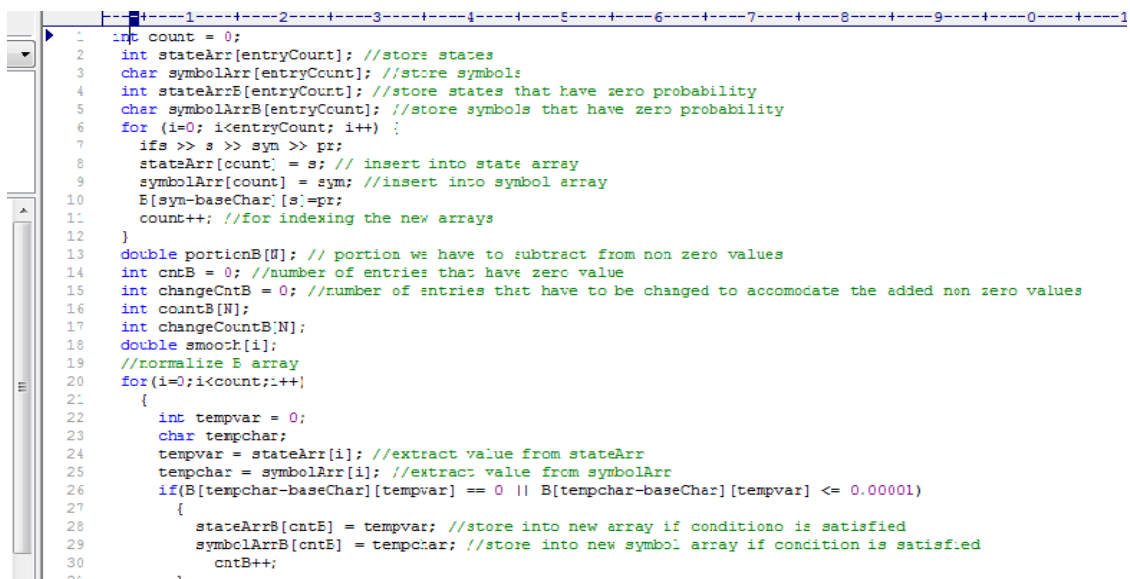
$p(w / s)_{ml}$  is emission probability.

$V$  is the number of symbols assigned non zero probability at state  $s$ .

$$P = (1 / (T_s + v))$$

$T_s$  is the total number of symbols emitted by state  $s$ .

In the thesis we implemented Absolute Discounting using the function



```
1  int count = 0;
2  int stateArr[entryCount]; //store states
3  char symbolArr[entryCount]; //store symbols
4  int stateArrE[entryCount]; //store states that have zero probability
5  char symbolArrB[entryCount]; //store symbols that have zero probability
6  for (i=0; i<entryCount; i++) {
7      ifs >> s >> sym >> pr;
8      stateArr[count] = s; // insert into state array
9      symbolArr[count] = sym; //insert into symbol array
10     E[sym-baseChar][s]=pr;
11     count++; //for indexing the new arrays
12 }
13 double portionB[M]; // portion we have to subtract from non zero values
14 int cntB = 0; //number of entries that have zero value
15 int changeCntB = 0; //number of entries that have to be changed to accomodate the added non zero values
16 int countB[N];
17 int changeCountB[N];
18 double smooth[i];
19 //normalize E array
20 for(i=0; i<count; i++)
21 {
22     int tempvar = 0;
23     char tempchar;
24     tempvar = stateArr[i]; //extract value from stateArr
25     tempchar = symbolArr[i]; //extract value from symbolArr
26     if(B[tempchar-baseChar][tempvar] == 0 || B[tempchar-baseChar][tempvar] <= 0.00001)
27     {
28         stateArrB[cntB] = tempvar; //store into new array if conditiono is satisfied
29         symbolArrB[cntB] = tempchar; //store into new symbol array if condition is satisfied
30         cntB++;
31     }
```

Figure 8 Screen Shot on Absolute Discounting

```

33 changeCntB = entryCount - cntB;
34 for (i=0; i < N ;i++ )
35 {
36     countB[i] = 0;
37 }
38 for (i=0; i < N ;i++ )
39 {
40     changeCountB[i] = 0;
41 }
42 for (i=0; i < N ;i++ )
43 {
44     for(j=0;j<cntB;j++)
45     {
46         if ( stateArrB[j] == i )
47         {
48             countB[i]++;
49         }
50     }
51 }
52 for (i=0; i < N ;i++ )
53 {
54     changeCountB[i] = SYMNUM - countB[i];
55 }
56 //
57 for (i=0; i < N ;i++ )
58 {
59     // cout<<"\n Count and Change count for  "<<i<< "are" <<countB[i] << " " << changeCountB[i] <<endl;
60 }
61 //
62 double dm;

```

Figure 9 Screen Shot on Absolute Discounting

```

63 for(i=0;i<N;i++)
64 {
65     if(SYMNUM==changeCountB[i])
66     {
67         portionB[i]=0;
68     }
69     else
70     {
71         dm = (SYMNUM + changeCountB[i]);
72         //cout<< "\n denominator of portionB"<< i<< dm;
73         portionB[i]= 1/dm;
74     }
75 }
76 for(i=0;i<N;i++)
77 {
78     cout<<"\n portion" <<i<< "=" << portionB[i];
79 }
80 for(i=0; i<N; i++)
81 {
82     double nms,dms;
83     if(SYMNUM == changeCountB[i])
84     {
85         smooth[i]=0;
86     }
87     else
88     {
89         nms=(portionB[i] * changeCountB[i]);

```

Figure 10 Screen Shot on Absolute Discounting

```

93     dms=( SYMNUM - changeCountB[i]);
94     smooth[i]= nms/dms;
95     }
96
97 }
98 ////
99 for(i=0; i<N; i++)
100 {
101     cout << "\n smooth of state " << i << " = "<<smooth[i] ;
102 }
103 ///////////////
104 cout<<"\nprinting B" <<endl;
105 for(i=0;i<count;i++)
106 {
107     cout<<stateArr[i]<<" " <<symbolArr[i] << " " << B[symbolArr[i] - baseChar][stateArr[i]]<<endl;
108 }
109
110 int c=0;
111 for(i=0;i<count;i++)
112 {
113     int tempvar = 0;
114     char tempchar;
115     char chl;
116     tempvar = stateArr[i];
117     tempchar = symbolArr[i];
118     chl=tempchar-baseChar;
119     for(j=0;j<cntB;j++)
120     {
121         if(tempvar == stateArrB[j] && tempchar == symbolArrB[j]) //if the value of index i and index j are the same
122         {

```

Figure 11 Screen Shot on Absolute Discounting

---

```

124     B[tempchar-baseChar][tempvar] = B[tempchar-baseChar][tempvar] + smooth[tempvar];
125     c++;
126 }
127 }
128 }
129 int flag=0;
130 for(i=0;i<count;i++)
131 {
132     int tempvar = 0;
133     char tempchar;
134     flag=0;
135     tempvar = stateArr[i];
136     tempchar = symbolArr[i];
137
138     for(j=0;j<cntB;j++)
139     {
140         if(tempvar == stateArrB[j] && tempchar == symbolArrB[j])
141         {
142             flag=1;
143         }
144     }
145     if( flag==0)
146     {
147
148         B[tempchar-baseChar][tempvar] = B[tempchar-baseChar][tempvar] - portionB[tempvar]; //subtracting
149     }
150 }
151 }
152 cout<<"printing normalized B" <<endl;
153 for(i=0;i<count;i++)

```

Figure 12 Screen Shot on Absolute Discounting

```

154     {
155     cout<<stateArr[i]<<" " <<symbolArr[i] << " " << B[symbolArr[i] - baseChar][stateArr[i]]<<endl;
156     }
157     double sum1,sum2;
158     sum1=0;
159     sum2=0;
160     int c1=0,c2=0;
161     for(i=0;i<count;i++)
162     {
163     if(stateArr[i]==0)
164     {
165     c1++;
166     sum1 = sum1 + B[symbolArr[i] - baseChar][stateArr[i]];
167     }
168     }

```

Figure 13 Screen Shot on Absolute Discounting

We evaluated the performance of Laplace smoothing and Absolute discounting by calculating precision and recall on the test data. The results obtained are presented in chapter 5.

## CHAPTER 5

### RESULTS

#### 5.1 Using HMM Model

An example will be provided in this chapter while training HMM model to explain how it works on example data. In this HMM model, we have a number of states, initial probabilities and output probability as shown in figure 1.

##### 5.1.1 HMM Model How it Looks

N is the number of states, InitPr is Initial probability, Output Pr is Output Probability, TransPr is Transition Probability

```
⌈
InitPr 2
0 0.5
1 0.5
OutputPr 4
0 a 0.99999
0 b 0.00001
1 b 0.99999
1 a 0.00001
TransPr 4
0 0 0.5
0 1 0.5
1 0 0.5
1 1 0.5
```

Figure 14 HMM Model

### 5.1.2 Results on HMM and How it Works

To run this program we are taking a data of telephone numbers and names.

The figure below shows the training data used in our example, a list of phone numbers and names.

```
BALAGUER-FEDERICO-- (217) 333-5219
BALIGA-GIRISH (217) -BANTWAL-- (217) 333-3202
BANDH (217) AKAVI-SINDH (217) URA-- (217) 333-5827
BAUGH (217) -LEE-W-- (217) 333-6740
BECKMAN-JR-ALAN-MATTOX-- (217) 244-5619
BELCH (217) ER-MATTH (217) EW-OLIVER-- (217) 265-9233
BH (217) ALLA-ARUN-- (217) 244-9682
BIEH (217) L-JACOB-T-- (217) 333-3077
BIKSH (217) ANDI-GANESH (217) -- (217) 244-5979
BOND-MICH (217) AEL-D-- (217) 265-9212
BOOTH (217) -JONATH (217) AN-A-- (217) 333-5827
BRADLEY-TERRY-J-- (217) 244-5973
BRANDYBERRY-MARK-D-- (217) 265-5301
BRAZ-RODRIGO-DE-SALVO-- (217) 333-2584
BREEMS-NICH (217) OLAS-SH (217) ANE-- (217) 244-8886
BRETH (217) OUR-TANYA-M-- (217) 333-3019
BULLOK-DANIEL-W-- (217) 333-6172
BUNDE-DAVID-PATTISON-- (217) 244-6433
```

Figure 15 HMM Model Data

From this given data we have to find the state sequence made of 0 and 1 where 1 indicates phone numbers and 0 indicates characters other than phone numbers. A continuous sequence of ten numbers is characterized as a phone number.

Below figure shows about the training data and result of HMM on telephone numbers and names. The output of HMM is a tagged sequence file which looks like the one shown below.

```
Best start state between S0 and S1 is 0
B 0
A 0
L 0
A 0
G 0
U 0
E 0
R 0
- 0
F 0
E 0
D 0
E 0
R 0
I 0
C 0
O 0
- 0
- 0
( 1
2 1
1 1
7 1
) 1
3 1
3 1
3 1
```

---

Figure 16 HMM Train Data

After completing the execution on HMM we face some problem for unseen events on MLE on state transition probabilities with given sequence for observed symbols. For avoiding such situations we are using Smoothing concept and different smoothing techniques.



## 5.2 Comparison of Two Smoothing Techniques

Laplace Smoothing and Absolute Discounting Smoothing are implemented in this work. The definition and equation of MLE is provided as below.

MLE is a Maximum Likelihood Estimation and while training HMM we face some difficulties in Supervised Training

### 5.2.1 Equation on MLE

$$P(w/s)_{ml} = (N(w, s) / N(s))$$

$N(w, s)$  = number of times symbols  $w$  is emitted at state  $s$

$N(s)$  = Total number of symbols emitted by state  $s$ .

### 5.2.2 Equation on Laplace Smoothing

$$P(w/s)_{lap} = (N(w, s) + 1) / (N(s) + |V|)$$

$|V|$  is the vocabulary size.

$N(w, s)$  = number of times symbols  $w$  is emitted at state  $s$

$N(s)$  = Total number of symbols emitted by state  $s$

There is a minute difference exist between equations of MLE and Laplace Smoothing.

### 5.2.3 Equation on Absolute Discounting

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

There is no optimal value of  $p$  but we can determine  $p$  using  $(1 / (T_s + v))$  which often gives good results.

Where  $T_s$  is the total number of symbols emitted by a state  $s$  (i.e) the

denominator of  $p(w / s)_{ml}$ .

### 5.3 Results on Laplace Smoothing

After including Laplace Smoothing in HMM, we see the count sequences values for given. In Figure 4 we see the improvements on output

```
|
I of 0 0.666667
A of 00 0.935961
A of 01 0.0640394
B of 00 0.00338983
B of 10 0.00338983
B of 20 0.00338983
B of 30 0.00338983
B of 40 0.00338983
B of 50 0.00338983
B of 60 0.00338983
B of 70 0.00338983
B of 80 0.00338983
B of 90 0.00338983
B of 100 0.00338983
B of 110 0.00338983
```

Figure 17 Laplace Smoothing Result

Here in this Figure 4(a) we see the initial state probabilities with states 0 is 1. In Figure 4(b) we see the Laplace Smoothing with two states 0 and 1 we get the sum B of state 0 as 1. In Figure 4(c) we can see the sum B of state 1 is 1

```

B of 810 0.00338983
B of 820 0.00338983
B of 830 0.00338983
B of 840 0.00338983
B of 850 0.00338983
B of 860 0.00338983
B of 870 0.00338983
B of 880 0.00338983
B of 890 0.00338983
B of 900 0.00338983
B of 910 0.00338983
B of 920 0.00338983
B of 930 0.00338983
sum B of state0 1
I of 1 0.333333
A of 10 0.0764331
A of 11 0.923567
B of 01 0.004
B of 11 0.004
B of 21 0.004
B of 31 0.004
B of 41 0.004

```

Figure 18 Laplace Smoothing Result

```

| B of 791 0.004
  B of 801 0.004
  B of 811 0.004
  B of 821 0.004
  B of 831 0.004
  B of 841 0.004
  B of 851 0.004
  B of 861 0.004
  B of 871 0.004
  B of 881 0.004
  B of 891 0.004
  B of 901 0.004
  B of 911 0.004
  B of 921 0.004
  B of 931 0.004
sum B of state1 1

```

Figure 19 Laplace Smoothing Result

#### 5.4 Results on Absolute Discounting

After including the absolute discounting in HMM we see the initial probabilities and state transition with states 0 and 1

```
portion0=0.00840336
portion1=0.00934579
smooth of state 0= 0.0030447
smooth of state 1= 0.00149994
printing B
0 ! 4.9751e-08
0 " 4.9751e-08
0 # 4.9751e-08
0 $ 4.9751e-08
0 % 4.9751e-08
0 & 4.9751e-08
0 ' 4.9751e-08
0 ( 4.9751e-08
0 ) 4.9751e-08
0 * 4.9751e-08
0 + 4.9751e-08
0 , 4.9751e-08
0 - 0.208954
0 . 4.9751e-08
0 / 4.9751e-08
0 0 4.9751e-08
0 1 4.9751e-08
0 2 4.9751e-08
0 3 4.9751e-08
```

Figure 20 Absolute Discounting Result

For reducing unseen events in HMM we include Smoothing Techniques in HMM training as shown in above Figures. Now we have to calculate the precision, recall and harmonic average accuracy for individual smoothing techniques to see their effect on HMM.

The performance of the smoothing techniques is evaluated based on standard precision, recall and harmonic average accuracy values [11].

Let TP be the number of true positives i.e. the number of documents which both experts and HMM agreed as belonging to the phone category.

Let FP be the number of false positives i.e. the number of documents that are wrongly tagged by the HMM as belonging to the tagged sequence.

### **5.5 Precision is defined as**

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

Let FP be the number of False Positive.

### **5.6 Recall is defined as**

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

Let FN be the number of False Negative.

### **5.7 Harmonic Mean**

The harmonic mean of precision and recall is called the F1 measure is defined

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

Here In this work, we have to calculate the precision, recall and F1 values are calculated. The ideal values of precision and recall is something which is greater than 0.8 and harmonic mean should be close to 1.

### **5.8 Results on Evaluation**

After careful calculations on HMM, without using Smoothing techniques and including smoothing techniques we have got the following results of the testing parameters:

#### **5.8.1 Without Using Smoothing Techniques**

Precision: 81.05

Recall: 98.54

F1: 89.68%

## **5.8.2 Using Smoothing Techniques**

### **5.8.2.1 Laplace Smoothing**

Precision: 86.05

Recall: 98.03

F1:91.7%

### **5.8.2.2 Absolute Discounting**

Precision: 90.16

Recall: 99.8

F1: 95.2%

## CHAPTER 6

### CONCLUSIONS AND FUTURE WORK

In this work, we have measured the performance of two different Smoothing Techniques in HMM for a given training data of phone numbers. We also compared to performance without being used the Smoothing Techniques. The accuracy of the HMM without using any Smoothing was found to be 89.68 %. Laplace Smoothing in HMM had an accuracy of 91.7% where as Absolute Discounting had 95.21 %. The absolute discounting technique of HMM showed better accuracy compared to Laplace Smoothing.

In future work, it might be interesting to implement other smoothing techniques and compare their effect on the performance of the HMM. Smoothing techniques that gave the best results may be used in our HMM to improve the performance of HMM (Hidden Markov's Model).

## BIBLIOGRAPHY

1. Stanley F. Chen, Joshua Goodman, “An Empirical Study of Smoothing Techniques for Language Modeling”, July 24,1998
2. ChengXiang Zhai, “A Brief Note on the Hidden Markov Models (HMM),” March 16,2003
3. Chuan Liu, “cu HMM: a CUDA Implementation of Hidden Markov Model Training and Classification”, May 6, 2009
4. Phil Blunsom, “Hidden Markov Models”, August 19, 2004
5. Lawrence R. Rabiner, “A Tutorial on hidden Markov Models and Selected Applications in Speech Recognition”
6. Kazem Taghva, Jeffrey Coombs, Ray Pereda, Thomas Nartker, “Address Extraction Using Hidden Markov Models”
7. Barbara Resch, “Hidden Markov Models”
8. L.E.Baum and T. Petrie, “Statistical Inference for Probabilities function of Finite State Markov Chains”. Annals of Mathematical Statistics, 37:1559-1563, [BP66], 1966
9. Gabor Molnar-Saska, “Analysis of the Maximum-Likelihood Estimation of Hidden Markov Models”
10. Michael Buckland, Fredric Gey, “The relationship between Recall and Precision”, University of California, Berkeley, Berkeley



VITA

Graduate College  
University of Nevada, Las Vegas

Sweatha Boodidhi

Degrees:

Bachelor of Technology in Information Technology, 2007  
Jawaharlal Nehru Technological University, India

Thesis Title: Using Smoothing Techniques to Improve the Performance of  
Hidden Markov's Model

Thesis Examination Committee:

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