COMPUTATIONAL INTELLIGENCE

Hidden Markov Model

Winter Semester 2018/19

Master's in IT

Submitted by: **Suraj Sanath kumar Bharadwaj**
Matriculation Number: **1224681**

Under Esteemed Guidance of
Prof Dr. Andreas Pech

Frankfurt University of Applied Sciences

Table of Contents

1. Introduction 1
1.1. HMM Background 1
2. Markov Chains 2
2.1 What is a Markov process? 2
2.2 Property 2
2.3 Terminology 2
2.4 Markov Model: Scenario 3
2.5 Conclusion of Markov Chain 4
3. Hidden Markov Model (HMM) 5
3.1 Define HMM 6
3.2 HMM example revisited (urn and ball example) 7
3.3 HMM Assumptions 7
3.4 Number of Hidden States 8
3.5 The Three basic Problems For HMM 9
3.6 Solutions to Problem 10
3.6.1 The Forward Backward Algorithm-Solutions to problem 1 10
3.6.2 Viterbi Algorithm – Solution to problem 2 12
3.6.3 The Baum – Welch Algorithm –Solution to Problem 3 14
4. HMM's For Classification 15
5. Applications 16
5.1 Human identification using Gait 16
5.2 Handwriting Recognition 18
5.3 Computational Biology- Gene finding and prediction 20
6. Conclusion 21
7. References 22

Abstract

How to apply machine learning to data which is represented as a sequence of observations over time? For example, we want to discover the sequence of words spoken based on an audio recording of their speech. Or on the other hand we may be keen on clarifying a grouping of words with their grammatical feature labels. This paper gives an exhaustive scientific prologue to the idea of Markov Models a formalism for thinking about states after some time and Hidden Markov Models where from a series of observations we wish to recover a series of states. The final section includes some applications of Hidden Markov Model in real time.

1. Introduction

In this paper we present an introduction to Hidden Markov Models (HMM) and their applications. HMMs are used for modeling time series data. They are used in almost all speech recognition systems, in areas of artificial intelligence, in applications in computational molecular biology, pattern recognition and in data compression. Recently HMMs been used in computer vision applications such as object tracking and image sequence modelling.

1.1. HMM Background

Hidden Markov Model theory was first introduced in the late 1960s and early 1970s by Baker at Carnegie-Mellon University and Jeninek and colleagues at IBM for speech recognition [1]. Outputs of real-world processes are generally observable and can be characterized as signals. These signals are either discrete (e.g., characters from a finite alphabet etc.), or continuous (e.g., speech samples, handwriting movements, music, etc.). The signals can be stationary or non-stationary, also signals can be pure or corrupted by noise. The main interest is to model the signals.

With a good signal model, simulation of source can be done and we can learn from these simulations. Using these signal models we can realize important practical systems—e.g., prediction systems, recognition systems, identification systems, etc., in a very efficient and robust manner. We can classify the signal models into the class of deterministic models, and the class of statistical models.

The region of interest is class of statistical models in which one tries to characterize only the statistical properties of the signal. Examples of statistical models include Gaussian processes, Poisson processes, Markov processes, and hidden Markov processes. The assumption of the statistical model is that the signal can be well characterized as a parametric random process, and that the parameters of the stochastic process can be estimated in a very efficient manner.

In this paper we will concentrate with one type of stochastic signal model, called the hidden Markov model (HMM). (These models are referred to as Markov sources or probabilistic functions of Markov chains in the communications literature.) [2].

First let's understand the theory of Markov chains and then make transition to the class of hidden Markov models using several simple examples. Then we will focus our attention on the three fundamental problems' for HMM design, solutions for these problems

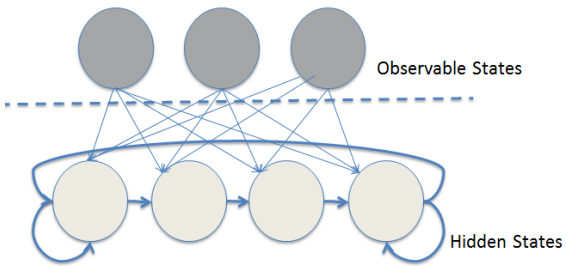


Figure 1. HMM

[1] Rabiner, L.R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. IEEE 1989, 77, 257–286

[2] D. H. Kil and F. B. Shin, Pattern Recognition and Prediction with Applications to Signal CharacterizationA, merican Institute of Physics, Woodbury, NY, 1996·

[1] https://www.researchgate.net/figure/Hidden-Markov-Model_fig2_283103282

2. Markov Chains

The underlying structure of a HMM consists of a Markov chain describing the probabilistic status between the states.

2.1 What is a Markov process?

Suppose we could predict the next day's weather based on what we saw on the previous days. For example if Monday is sunny and Tuesday is sunny and Wednesday is sunny then the probability that it will be sunny on Thursday is 90 percent. We can see how this type of model is widely applicable to many fields. For instance in natural language processing we could create automatic writing systems by creating probability models for example if we start with the sequence the MacBook was created by, the next where it might be Apple or Steve Jobs or something along those lines.

The Markov property is when tomorrow's weather only depends on today's weather but not yesterday's weather. It's when the next word in the sentence depends only on the previous word in a sentence but not on any of the other words. We are throwing away all historical data except for most recent one in general.
So in general it is a random process in which the current state depends only on previous state or next state depends only on current state. Another way of saying this is that the distribution of the state at time t depends only on the state at time t-1.

2.2 Property

In general: "states"

State at time t: s(t)

$p(s(t) \mid s(t-1), s(t-2),..., s(0)) = p(s(t) \mid s(t-1))$

Why do this? The goal is to model the joint probability or in other words the probability of seeing an entire specific sequence. We can see that this becomes quite a large expression when it's fully expanded but becomes simpler when we use the Markov property.

$p(s4, s3, s2, s1) = p(s4 \mid s3, s2, s1)p(s3, s2, s1)$

$= p(s4 \mid s3, s2, s1)p(s3 \mid s2, s1)p(s2, s1)$

$= p(s4 \mid s3, s2, s1)p(s3 \mid s2, s1)p(s2 \mid s1)p(s1)$

$= p(s4 \mid s3)p(s3 \mid s2)p(s2 \mid s1)p(s1)$ (if we assume Markov property)

2.3 Terminology

If we have only $p(s(t) \mid s(t-1))$ ⟶ First-order Markov

$p(s(t) \mid s(t-1), s(t-2))$ ⟶ Second-order Markov

$p(s(t) \mid s(t-1), s(t-2), s(t-3))$ ⟶ Third-order Markov

Etc..

2.4 Markov Model: Scenario

Let's consider a weather example.

Classify a weather into three states

State 1: rain

State 2: cloudy

State 3: sunny

Figure 2: Rainy, Cloudy, Sunny

Assumptions: Tomorrow's weather depends only on today's one!

		Tomorrow		
		Rain	Cloudy	Sunny
Today	Rain	0.4	0.3	0.3
	Cloudy	0.2	0.6	0.2
	Sunny	0.1	0.1	0.8

Table 1: State transition probability

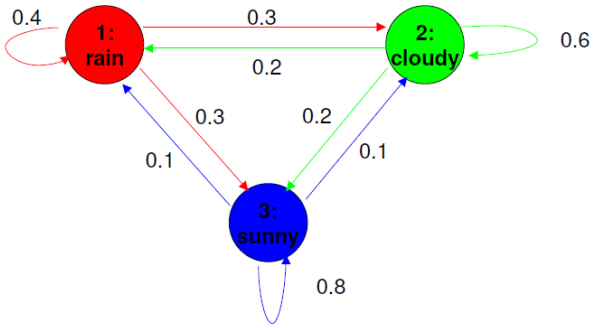


Figure 3: Markov model of weather

-Each state corresponds to one observation

-Sum of outgoing edge weights is one

-Each state can go to each state, including itself

This means we have 3 weights for each of 3 states. Therefor N*N number of weights for N states. We store this in a NxN matrix called A which is called **state Transition matrix**.

N states ⟶ N*N weights

A ⟶ NxN matrix

A(i, j)=p(s(t)=j | s(t-1)=i) $1 \leq i, j \leq N$ ……(2.1)

$$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ . & . & . & . \\ . & . & . & . \\ a_{N1} & a_{NN} & \dots & a_{NN} \end{bmatrix}$$ ……(2.2)

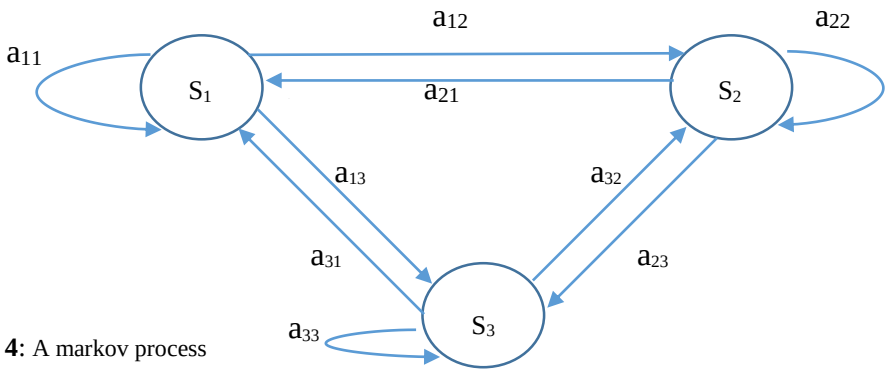


Figure 4: A markov process

[2, 3] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

[4] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

Any element in A(i, j) represents the probability of going to state j from state i. So the Constraints on state transition matrix A is that whenever we are in State i we must go to any one of the three states and there are no other possibilities. Therefore sum of row A(i, :) = 1 where i=1 …N .

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad \text{……(2.3)}$$

Another important concept in Markov Models is where we start. This is called initial state distribution function. Represented by N Dimensional vector called pi .Symbol π

$$\pi_i = (s_1 = i) \qquad 1 \leq i \leq N \qquad \text{……(2.4)}$$

The Observable States are {1,2,3,… N} and the Observerd Sequence is $S_1,S_2,..,S_T$.

In general

$$p(\, s(0),..,s(T)\,) = \pi\,(s(0))\; \Pi_{t=1..T}\; p(\, s(t) \mid s(t\text{-}1)\,) \qquad \text{……(2.5)}$$

Example: Given that weather on day 1 is sunny (state 3), question can be asked (according to figure 3) What is the probability that the weather for the next 7 days will be "sun-sun-rain-rain-sun-cloudy-sun" when today is sunny?

S_1: rain , S_2 :Cloudy, S_3 : Sunny

$P(O \mid model) = P(S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3 \mid model)$

$=P(S_3) \cdot P(S_3 \mid S_3) \cdot P(S_3 \mid S_3) \cdot P(S_1 \mid S_3) \cdot P(S_1 \mid S_1) \cdot P(S_3 \mid S_1) \cdot P(S_2 \mid S_3) \cdot P(S_3 \mid S_2)$

$= \pi_3 \cdot a_{33} \cdot a_{33} \cdot a_{31} \cdot a_{11} \cdot a_{13} \cdot a_{32} \cdot a_{23}$

$= 1 \cdot (0.8)(0.8)(0.1)(0.4)(0.3)(0.1)(0.2)$

$= 1.536*10^{-4}$

2.5 Conclusion of Markov Chain

The probability of observing observation O_t, given the model λ at a time t, is determined by the state at the time preceding states given by

$$P(O_t \mid \lambda) = P[q_t = s_j,\, q_{t-1} = s_i\,,\, q_{t-2} = s_k\,,\ldots] \qquad \text{……(2.6)}$$

For a first-order Markov chain, eq (2.6) can be simplified to:

$$P(O_t \mid \lambda) = \prod_{k=1}^{t} P(s_k \mid s_{k-1})\,.\,P(s_0) \qquad 1 \leq t \leq T \qquad \text{……(2.7)}$$

Finally, the initial conditions, the probabilities of starting (t=1) at the i^{th} state s_i :

$$\pi_i = P[q_1 = s_i\,] \qquad i=1, 2,\ldots.. N. \qquad \text{……(2.8)}$$

Now after understanding the concept of markov chain where states are observable, now let's understand the concept HMM (Hidden Markov Model).

3. Hidden Markov Model (HMM)

The basic idea is that there is something going on besides what we can see and observe and measure. What we observe is usually stochastic or random since if it were deterministic then we could predict it perfectly without doing any machine learning at all. The assumption that we make when there are hidden or latent variables is that there is some cause behind the scenes that's leading to the observations that we see. In Hidden Markov models the hidden cause is itself stochastic. It's a random process, a Markov chain.

In speech to text, a computer can't read the words you are attempting to say but it can use an internal language model i.e. a model of likely sequences of hidden states to try and match those to the sounds that it hears. So in this case what is observed is just a sound signal and the latent variables are the sentence or phrase that you are saying. So how do we go from Markov models to Hidden Markov models? The best way to do this is by means of example.

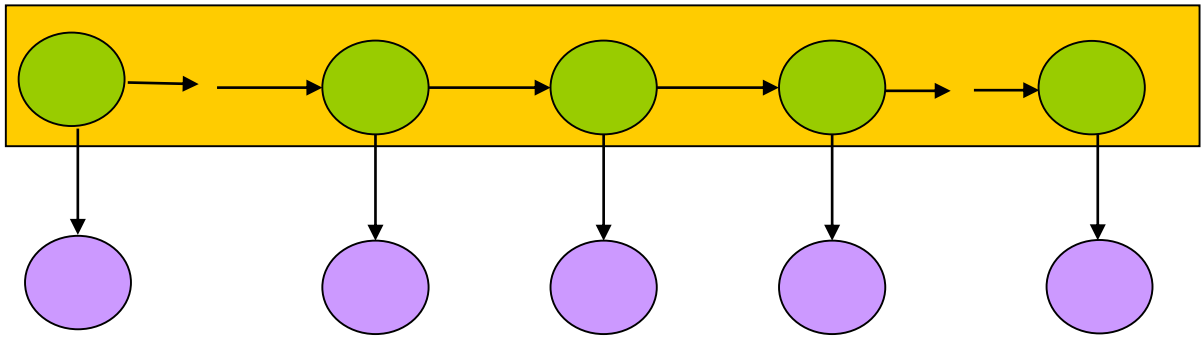


Figure 5: Green circles are hidden states, purple circles are observed states depend only on corresponding hidden states

Suppose you're at a carnival and a magician has two biased coin that he's hiding behind his back. He will choose to flip one of the coins at random and all you get to see is the result of the coin toss either Heads or Tails. So what are the hidden states and what are the observed variables? Since we can see the result of the coin toss that means heads or tails are observed variables. We can think of this as a vocabulary or space of possible observed values. The hidden states are of course which coin the magician chose to flip. You can't see that, so it's hidden. This is called a stochastic or random process, since it's a sequence of random variables.

To understand HMM more deeply let's consider an urn and ball model which was introduced by Jack Ferguson.

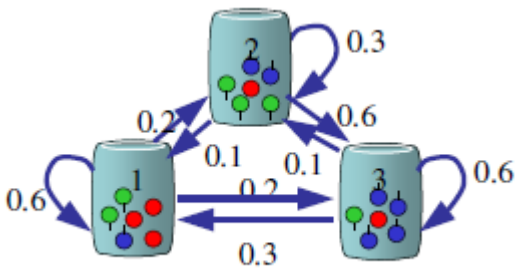


Figure 6 : Model of Process

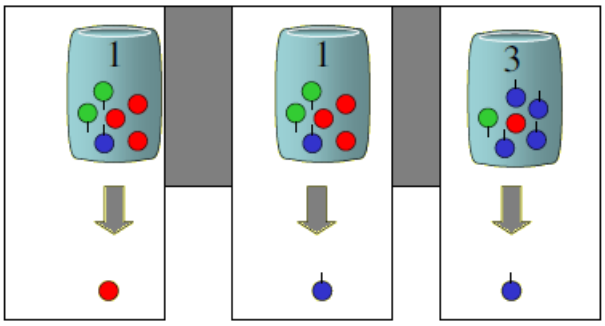


Figure 7: Random ball chosen from urns

Let's say there are three urns (N) in a room and each urn contains. Inside each urn there are number of colored balls (M).Each urn has different number of colored balls

[6, 7] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

So the algorithm for generating sequence is as follows:

A genie picks up a random ball from a chosen initial urn according to some random process. The resulting color ball is noted as an observation. The ball is kept back into the urn from which it was picked. The genie selects a new urn according to the random selection process which is connected to the first urn and a ball is picked and observation is noted. This process is repeated and entire process generates a sequence of colors which is observable which is the observable model of HMM (output). So each state corresponds to a specific urn and ball color probability is specified for each state. The state transition matrix guides the choice of urn. So the hidden part is the urn and observable is just the chosen color balls which is depicted in figure 8 below. So urn selection (state transition matrix) is hidden.

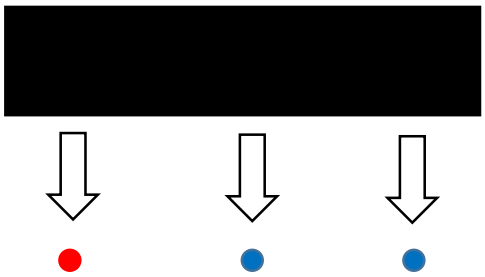


Figure 8: Black strip is hidden states.

3.1 Define HMM

Let us now formally define elements of HMM based on above examples. A HMM has three parts π, A, B. This is opposed to the regular Markov model which just has π. Pi is the initial state distribution or the probability of being in a state when the sequence begins. In our coin example suppose our magician really likes coin 1 so the probability that he starts with this coin is 0.9.

$$\pi_i = \textbf{probability of starting at state i} \quad \ldots\ldots (3.1)$$

A is the state transition matrix which tells us the probability of going from one state to another. In hidden market models the states themselves are hidden. So A corresponds to transitioning from one hidden state to another hidden state.

$$\textbf{A(i, j)= Probability of going from state i to state j} \quad \ldots\ldots (3.2)$$

The new variable here is of course B. This is the probability of observing some symbol given what state you are in. Notice that this is also a matrix because it has two inputs. What state you are in, which is j, in which you observe which is k.

$$\textbf{B(j, k) = probability of observing symbol k when you are in state j} \quad \ldots. (3.3)$$

The number of states in the model is denoted by N. In coin toss example the each state corresponds to a distinct biased coin. In urn and ball example states corresponds to urns.

The number of symbols observable in a state is denoted by M. For coin toss example observation symbols was just heads or tails, for urn and ball example observation symbols were colors of balls chosen from urn. Individual symbols are denoted as:

$$\mathbf{V = \{v_1, v_2, v_3, v_4, \ldots\ldots\ldots\ v_M\}} \quad \ldots\ldots (3.4)$$

So given values of N, M, π, A and B, HMM can be used as generator to obtain observation sequence

$$O = O_1, O_2, O_3, \ldots\ldots O_T \qquad \ldots\ldots (3.5)$$

where number of observations is T and each O_t is one of symbols from V.

Steps:

1. An initial state $q_1 = S_i$ is chosen according to π which is initial state distribution.
2. Set time t=1.
3. Choose $O_t = v_k$ according to symbol probability distribution in state i.e. B(j, k).
4. Go to new state $q_{t+!} = S_j$ according to state transition probability distribution for state S_i i.e. A(i, j).
5. Now make t=t+1 and return to step 3 if t<T, else terminate the process.

Therefore the model for HMM is defined in compact notation as:

$$\boldsymbol{\lambda = (A, B, \pi)} \qquad \ldots\ldots (3.6)$$

3.2 HMM example revisited (urn and ball example)

From Figure 7:

No of states: N =3

No of observations: M=3

V= {R, G, B}

Initial state distribution $\pi = \{ P(q_1 =i) \} = [1, 0, 0]$

State Transition probability distribution matrix:

$$A = \{a_{ij}\} = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.1 & 0.3 & 0.6 \\ 0.3 & 0.1 & 0.6 \end{bmatrix}$$

Observation symbol probability distribution matrix:

$$B = \{ b_i (v_k) \} = \begin{bmatrix} 3/6 & 2/6 & 1/6 \\ 1/6 & 3/6 & 2/6 \\ 1/6 & 1/6 & 4/6 \end{bmatrix}$$

3.3 HMM Assumptions

- In HMM we are making more independent assumptions than the Markov assumption.
- Remember that the markov assumption is that the current state depends only on the previous state but is independent of any state before the previous state.

$$\boldsymbol{P(q_t \,|q1,q_2,\ldots.q_{t-1}) = P(q_t \,|q_{t-1})} \qquad \ldots\ldots (3.7)$$

- Now that we have both observed and hidden variables in our model we have another independence assumption and that's what we observe, is only dependent on the current state. So the observation at time t depends only on the state at time t but not on any other time or any other state or any other observation.

$$\boldsymbol{P(X_t \,|q_t,X_1,\ldots X_{t-1},q_1,\ldots.,q_{t-1}) = P(X_t \,|q_t)} \qquad \ldots\ldots (3.8)$$

3.4 Number of Hidden States

So how do we choose the number of hidden states? In machine learning the number of hidden states is called hyperparameter. We can imagine that if we have n training samples, then we could train a model with n parameters and can get a perfect score with no error. But this doesn't say anything about how well our model will perform on data that we haven't seen yet. With stocks for example. Sure we want the model to have good accuracy for last year's trends but what we really want is to be able to predict next year's trends with good accuracy. In linear regression, if we fit too tightly with the training data we will increase our error on the test data which is the opposite of what we want to do. Another way of thinking is that all data is noisy. When we overfit to our training data we are fitting to its trend and its noise. Our goal should be to fit to the trend while ignoring the noise. If we can capture the real underlying trend then we should be able to accurately predict non training data.

So what is the right amount of expressiveness of a model. If it's too expressive we will overfit and if it's not expressive enough we will underfit. That is like trying to fit a line to a sine wave. A lot of the time we don't know exactly what the model is going to learn. So we just have to try different values. We'll keep some data outside the training data and call this the validation set and fit the model to the training data only and get the cost on the validation set. We will then choose the number of hidden states that gives us the highest validation accuracy. A common way of doing this is fold cross-validation.

A lot of the time the number of states in an HMM can reflect a real physical situation or what we know about the system that we are trying to model. For instance in our magician example we know the magician has only two coins. So we would use just two states. When we are using speech to text, we know the number of words in vocabulary. In addition we can separately train the hidden state transitions on pure text to give us a good initialization on the transition probabilities. Another example is in biology, a codon is a sequence of three DNA or RNA nucleotides. Furthermore, these are in charge of making explicit amino acids which at that point transform into proteins. A simple HMM might then have three hidden states so we can use our insight of a physical system to help us determine the number of hidden states.

3.5 The Three basic Problems For HMM

Given the type of HMM of the past segment, there are three essential issues of intrigue that must be tackled for the model to be valuable in true applications. These problems are the accompanying:

Problem 1: If the observation sequence $O = O_1, O_2, O_3, \ldots\ldots O_T$ is given and a model $\lambda = (A, B, \pi)$ is given, then how to compute efficiently the $P(O \mid \lambda)$ observation sequence probability, given the model?

Problem 2: If the observation sequence $O = O_1, O_2, O_3, \ldots\ldots O_T$ is given and a model $\lambda = (A, B, \pi)$ is given, then how to choose corresponding state transition sequence $Q = q_1\, q_2\, q_3 \ldots q_T$ which best explains the observations?

Problem 3: How to determine modal parameters $\lambda = (A, B, \pi)$, given a training set of observations to maximize $P(O \mid \lambda)$?

Problem 1 is called the **Evaluation problem**, given model and observation sequence how can we determine the probability that observed sequence was produced by the model, i.e. what's the likelihood that, the model is generating particular sequence. We can also view it as how well the given model is matching observation sequence. Problem 2 is called the **Decoding problem**, in which we try to uncover the hidden part of model i.e path analysis is done. For practical situations, to solve this problem we generally use optimal criterion. There are many optimal criterion that can be imposed which acts as a strong function to uncover the states.

Problem 3 is called the **Training problem,** tells how can we best optimize model parameters so that we can know how best a given observation sequence comes out. The model parameters are adjusted using the observation sequence and hence is called training sequence since it is used to "train" HMM. This is most crucial part as it allows to create best model for real world applications.

3.6 Solutions to Problem

3.6.1 The Forward Backward Algorithm-Solutions to problem 1

Suppose we have N hidden states in our sequence of observations of length Big T. The idea is we want to marginalize the joint probability over all possible values of the hidden states. In this case we get equation (3.10). Notice how this involves all of our HMM variables. π gives us the probability of the initial state, A gives us the probability of going to state j from state i and B gives us the probability of seeing symbol k from state j. So how long will this take to calculate well in the inner part.

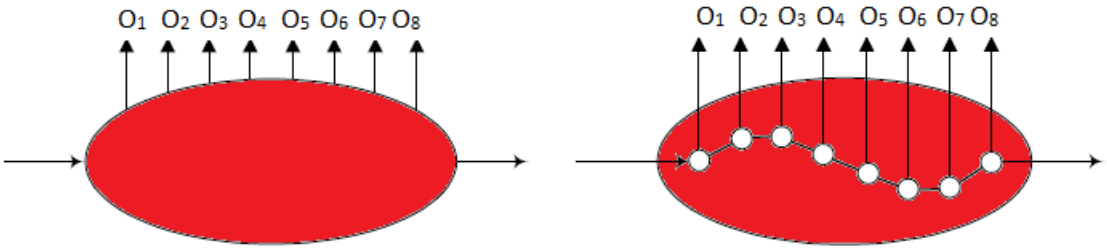


Figure 9

$$O = \{O_1, O_2, O_3, \ldots\ldots O_T\} \quad \ldots\ldots(3.9)$$

$$Q = q_1\, q_2\, q_3 \,..\, q_T \quad \ldots\ldots(3.10)$$

For some hidden state sequence, joint probability distribution is :

$$P(O,Q) = \pi(q_1)P(O_1 \mid q_1)\prod_{t=2}^{T} P\,(q_T \mid q_{T-1})P(O_T \mid Q_T) \quad \ldots\ldots(3.11)$$

Therefore to get the probability of some sequence $O = \{O_1, O_2, O_3, \ldots\ldots O_T\}$ we marginalize the Q's:

$$P(O) = \sum_{q1=1}^{N} \ldots \sum_{qT=1}^{N} P(O,Q) = \sum_{q1=1}^{N} \ldots \sum_{qT=1}^{N} P(O_1, O_2, O_3, \ldots O_T, q_1\, q_2\, q_3 \,..\, q_T) \quad \ldots\ldots(3.12)$$

So how long will this take to calculate, well in the inner part we have a product that's two times big T minus one products in total. But how many times do we have to compute this product. This is equal to the number of possible state sequences which is N^T. So in total this is O (T) times N^T. This is exponential growth which is pretty bad so we don't want to do this. To be precise we need $2T.N^T$ calculations, for e.g., for N=5(States) and T = 100 (observations), we would have $2.100.5^{100} = 10^{72}$ computations. Clearly a more straight forward and efficient procedure is required. This would be the forward backward algorithm.

[9] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

So how does the forward backward algorithm work? It is usually described in 3 steps. We need to define a variable called Alpha α. This is the forward variable and it represents the joint probability of seeing a sequence we have observed till now and being in a specific state at that time. So there are two indexes to α, time and i which indexes the state.

$$\textbf{Define: } \alpha(t, i) = P(O_1, O_2, O_3, \dots O_T, q_T = i) \quad \text{.....(3.13)}$$

1. **Initialization step (t=1):** The first step is to calculate the initial value of α. **So** $\alpha(1, i) = \pi_i B(i, O_1)$
   So in general: $\alpha(t, i) = \pi_i B(i, O_t)$ .....(3.14)

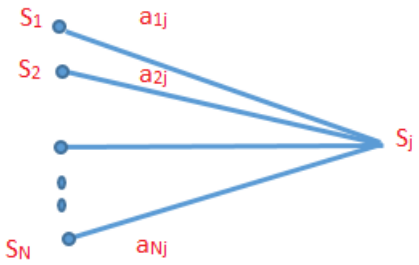


Figure 10: Sequence of operations to compute forward variable α(t+1, i)

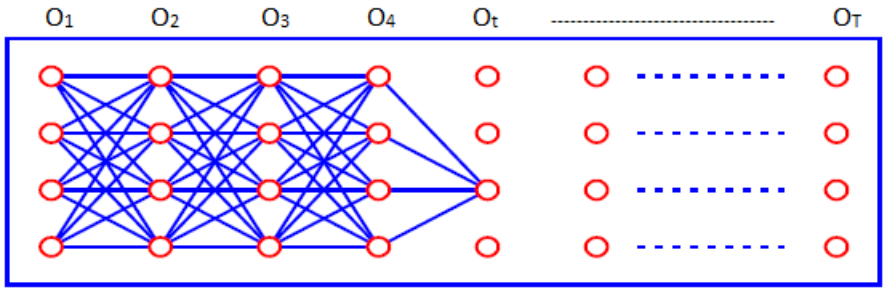


Figure 11: Implementation of computation $\alpha_t(i)$ by spanning a lattice of N states and T times.

Figure 10 shows how S_j state can be reached from N possible states at time t + 1.

2. **Induction step(for t > 1 up to T) :** This is the second step and we will do this for t >1 and for every time up to T.

$$\alpha(t + 1, j) = \sum_{i=1}^{N} \alpha(t, i) A(i, j) B(j, O_{t+1}) \quad \text{.....(3.15)}$$

3. **Termination Step(t=T) :** The final step is the termination step where we marginalize over the hidden states at time T.

$$P(O) = \sum_{i=1}^{N} \alpha(T, i) = \sum_{i=1}^{N} P(O_1, O_2, \dots, O_T, q_T = i) \quad \text{.....(3.16)}$$

[10] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf
[11] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

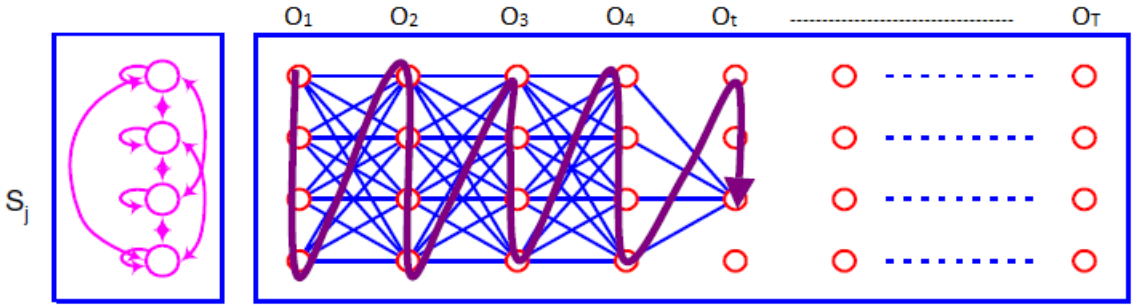


Figure 12: Termination Step

Notice that we already have our answer. We know the probability of the sequence after only having done the forward step of the forward backward algorithm. We can also verify that the time complexity of this algorithm is $O(N^2T)$

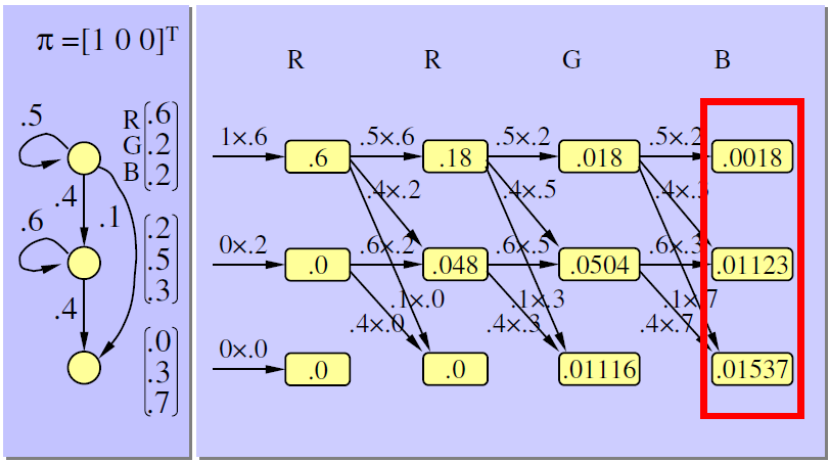


Figure 13: Numerical example P(RRGB | λ)

Backward algorithm:

It is essentially just the reverse of forward algorithm. To perform the backward algorithm we will define a variable called β which is also indexed by time and a state. The process is similar to the forward algorithm. At each state i at time t, the sum of probabilities of all the outgoing paths are kept.

1. **Initialization step:** The initialization step is to define β at time T to be 1 for every state.

$$\beta(t, i) = 1 \qquad \text{.....(3.17)}$$

2. **Induction Step:** The induction step is to then calculate the previous β for every state. Similar to what we did with the forward algorithm. Again we want to do this for all times down to 1 or zero depending on how you index and for every state at each time.

$$\beta(t, i) = \sum_{j=1}^{N} A(i, j)B(j, O_{t+1})\, \beta(t+1, j) \qquad \text{.....(3.18)}$$

[12] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf
[13] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

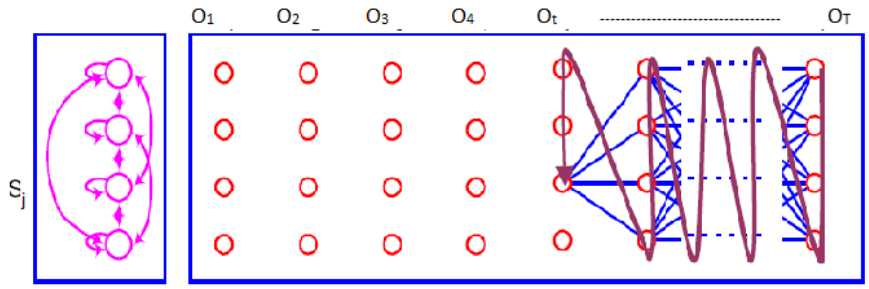


Figure 14: Backward Algorithm

3.6.2 Viterbi Algorithm – Solution to problem 2

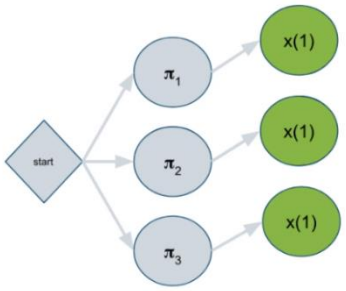
This algorithm helps to determine what the sequence of hidden States is, given the observation. For example given a sound sample of someone speaking we might want to determine the words they're saying, this is what the Viterbi algorithm calculates, the most probable hidden state sequence given the observed sequence under current model. This algorithm works a lot like the forward algorithm we just discussed. Except that instead of doing a sum we just take the max. One extra thing we need to add is backtracking because we want to keep track of the actual states, not just determine the final probability.

We will create two new variables, $\delta(t,i)$ which is indexed by time and state to represent the maximum probability of ending up in state i at time T which is a joint probability distribution over the state sequence and observed sequence. We also define $\psi(t,i)$ which is also indexed by time and state to keep track of the actual state sequences that end up at time t in state i. We will again have three steps.

1. **Initialization step**: The first step is to consider just one observation x(1), since one observation corresponds to just one state. Then we just want to find the most probable single state. This is just the maximum π times B which is the probability of going to a state and then observing what we observed from that state.

$$\delta(t,i) = \pi_i B(i, x(1)) \qquad \text{……(3.19)}$$

$$\psi(1,i) = 0$$



max(
$\pi_1 B(1,x(1))$,
$\pi_2 B(2,x(1))$,
$\pi_3 B(3,x(1))$,
)

Figure 15

2. **Recursion Step:** Let's now consider a two-step sequence. x(1) isn't shown here for compactness. Suppose we consider just state one that's the one at the very top. How can we get there. Well in general we can get there from any other state. So the main message here is again pick the one that gives us the maximum probability so far. How can we get that probability. That's the previous δ which if you remember is π times B at times

[14] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

equals 1 and then transitioning from that state to the current state which we're right now assuming is 1 and then multiplying by the observation probability from this state.

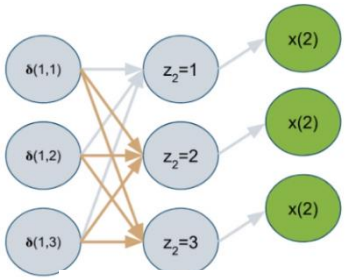


Figure 16

Well this is just the definition of the next δ. So δ is also defined recursively in terms of its previous value. In the end, what we want is the best state sequence considering all the observed variables as a whole. So that means when we're looking at only the first observation perhaps being in state one gives us the best total probability. But then if we consider both observations being in state two and then transitioning to state one might give us the best total probability for the entire sequence. So then we should say that the state at t=1 is 2.

Because we can only determine the best sequence at the very last Delta, We need to keep track of all possible state transitions we encountered. That's not exponential because it doesn't consider all possible state transitions in total just the ones that were giving us the best probabilities along the way.

$$\delta(t,j)=\max_{1\le i\le N}\{\delta(t-1,i)A(i,j)\}B(j,x(t)) \quad \ldots\ldots(3.20)$$

$$\psi(t,j)=\operatorname{argmax}_{1\le i\le N}\{\delta(t-1,i)A(i,j)\} \quad \ldots\ldots(3.21)$$

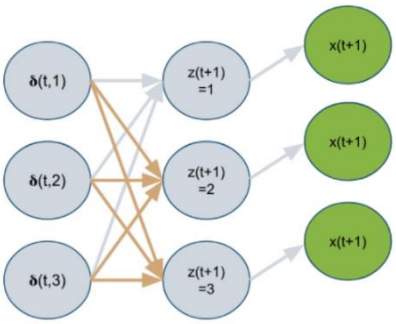


Figure 17

3. **Termination step:** Since δ (T,i) is the best probability for observing the whole sequence X(1)……,X(T) and ending up in state i. And the max of that will give us the best probability overall. Furthermore the argmax will give us the best last state and since we were keeping track of all the transitions we know we got to the last state from the second last state. We know the third last state and so on.

$$P^* = \max_{1\le i\le N}\delta(T,i) \quad \ldots\ldots(3.22)$$

$$q_T = \operatorname{argmax}_{1\le i\le N}\delta(T,i) \quad \ldots\ldots(3.23)$$

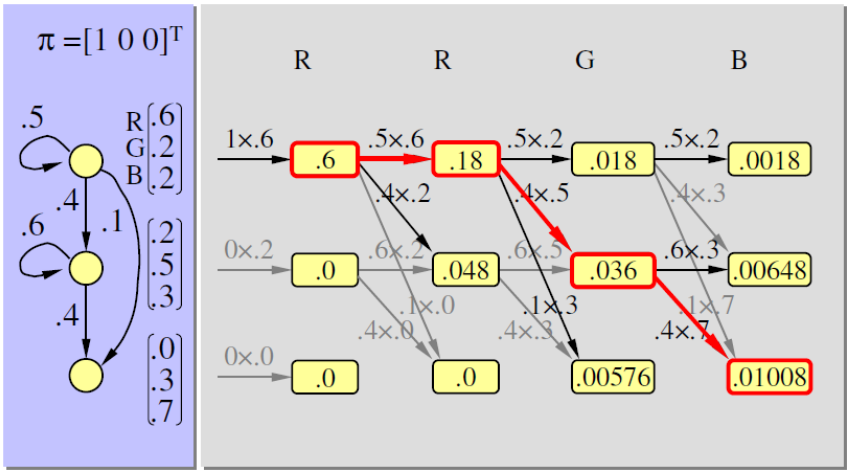


Figure 18: Example P(RRGB | λ)

[18] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

3.6.3 The Baum – Welch Algorithm –Solution to Problem 3

This is solution for how to train HMM, similar to Gaussian mixture models. We have a latent variable where all the possibilities need to be summed over, so we can't easily find the maximum likelihood solution. We need to use the expectation maximization algorithm which is an iterative algorithm. The bomb welch algorithm is named after the mathematicians who invented it and it relies on the forward backward algorithm that we learned about earlier. In fact the first step to completing the bomb welch update is to first compute the forward and backward variables α and β.

Once we have those we can create another quantity called ξ. ξ has three indices, t, i and j. It is defined as the probability of being in state i at time t, transitioning to state j at time t=1 given the observation sequence.

The below figure shows this.

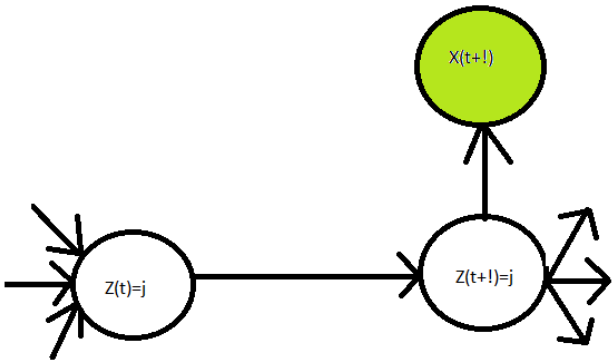


Figure 19: Visual of φ

$$\xi\,(t,i,j) = \frac{\alpha(t,i)A(i,j)B(j,x(t+1)\beta(t+1,j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}A(i,j)B(j,x(t+1)\beta(t+1,j)} \qquad \text{…….(3.24)}$$

Once we have ξ we can define another variable γ which depends only on t and i so it sums over j.

$$\gamma(t,i) = \sum_{i=1}^{N}\xi\,(t,i,j) \qquad \text{…….(3.25)}$$

The key is that when we sum of these variables over all-time γ represents the expected number of transitions from state i . ξ represents the expected number of transitions from state i to state j. Now this is starting to sound a lot like how we train the Markov model. So we can define our **update equations** as follows.

$$\pi_i = \gamma(1,i) \qquad \text{…….(3.26)}$$

$$A(i,j) = \frac{\sum_{t=1}^{T-1}\xi\,(t,i,j)}{\sum_{t=1}^{NT-1}\gamma(t,i)} \qquad \text{…….(3.27)}$$

$$B(i,k) = \frac{\sum_{t=1}^{T-1}\gamma(t,i)\ if\ x(t)=k\ ,else\ 0}{\sum_{t=1}^{NT-1}\gamma(t,i)} \qquad \text{…….(3.28)}$$

Now keep in mind we're only considering one sequence at this point. Of course we want to fit our model to all of our training sequences. How can we do that?

The fundamental principles depend on an algorithm called the expectation maximization algorithm. The basic idea is that it's an iterative procedure that depends on two steps. The expectations step and the maximization step. It might be helpful to first consider what would happen if the hidden states in the hmm were not hidden.

One example of this is parts of speech tagging in natural language processing. In parts of speech tagging we train on a data set that contains sentences made up of sequences of words and corresponding sequences of parts of speech tags. Parts of speech tags tell us the role of a word in a sentence, for example noun, verb, adjective and so on. We can see that because of the structure of the English language there is a Markovian probability model underlying the sequence of POS tags. When we make predictions we would be given a new sentence with no pos tags and then we would find the most likely hidden state sequence and that would be the sequence of POS tags we predict. So what algorithm is that. That's the Viterbi algorithm, which we previously discussed gives you the most likely sequence of hidden States. What's interesting about this problem is that in the training data the hidden states are not actually hidden. We know exactly what they are. Therefore we can use maximum likelihood in closed form to determine all the hidden state transitions. For example the probability of transitioning from noun to verb is the number of times we transition from noun to a verb divided by the number of times we had a noun.

P(VERB | NOUN) = count(NOUN→VERB) / count(NOUN)

Similarly we can find observation probabilities. For example the word milk can be a noun or verb. To find the probability of the word milk given noun and the probability of the word milk given verb are both easy because we already have all those pairs of data in the training set. So it's just counting.

P("milk | NOUN) = count("milk" is NOUN)/count(NOUN)

4. HMM's For Classification

Even though the H&M is an **unsupervised** model, because it models a probability we can turn it around and use it as a classifier with baye's rule. We actually have a name for these types of classifiers. They're called generative models because you take the probability density of a class and you pretend that it generates the data that you observed. Compare this to discriminative models like logistic regression and deep neural networks. Recall that these models give you a probability of a class given the input directly. There is no modeling of the likelihood probability. Sometimes these two types of models intersect

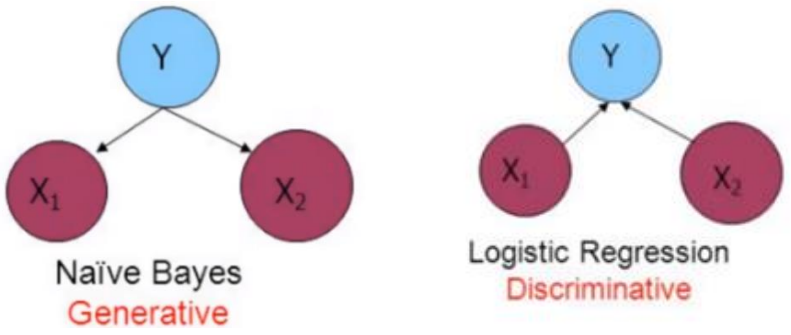


Figure 20

In logistic regression remember that you can't solve for the weights directly so you need to use a method like gradient descent. There was an exception to this rule if we assume that each class had the same covariance but different means and they are both Gaussian in this situation we could solve for the weights directly. Now if we know that both classes are Gaussian and we know their means and covariances, well then we know the likelihoods. So this is a generative model also. And in this particular setup generative and discriminative models intersect. Now

that we know there's a difference between the two, is there any reason to use one over the other.? We have seen in that generative models have a lot of theory behind them. They are very principled. The research has shown that discriminative models simply work better. One of the drawbacks of deep learning is that it's hard to explain exactly what's going on or what features are being learned. There is no graphical model you can point to and say well here's a strong connection between these two nodes which represent something in physical reality. That's fine because often what we care about is accuracy and functionality, not just simple relationships between input variables. The strength of neural networks is that they model complex relationships.

Unfortunately that complexity also means that they can't be explained in terms that would be satisfactory to someone like a statistician.

5. Applications

5.1 Human identification using Gait

Figure 21: Dataset showing different positions

Dataset consists of 122 individuals, 12 unique Probe sets (walking surfaces, w/o briefcases, shoe type, different sessions, camera orientation) Human Gait is often referred as a collection of gait cycles. One Gait cycle means a sequence of: rest position-left foot forward-rest position-right foot forward-rest position. The two components of human gait: Structural component which is One's physical features and Dynamic component which is the body's motion dynamics .These are modeled using HMM's.

Figure 22: Applying bounding

Figure 23: Segregating into foreground and background

A bounding box is defined manually for each frame in Figure 22 in the gait video. From the statistics of pixels outside the bounding box, a background model is built. After learning the background distribution, the pixels are classified as foreground or background pixels which are inside the bounding box. Figure 23.

Figure 24

Figure 24 shows the one half of gait cycle.

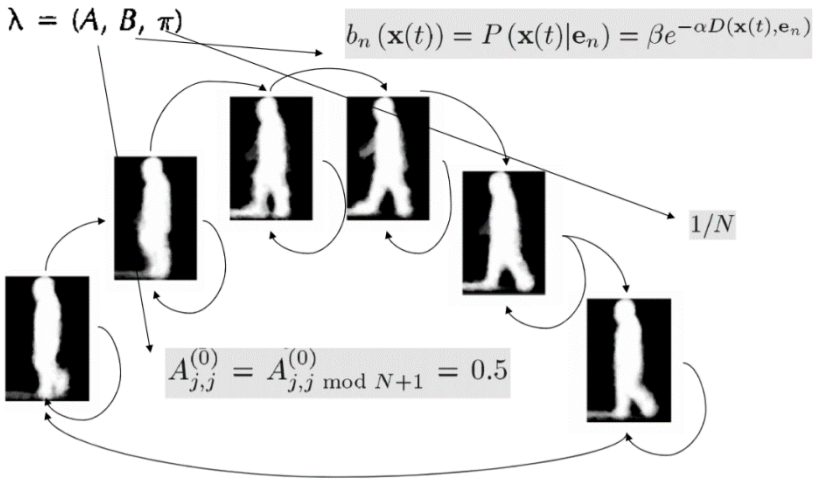
[21,22,23,24] www.cs.umd.edu/~djacobs/CMSC828/ApplicationsHMMs.pdf

- Observation symbols: The entire background subtracted silhouette is chosen as observation symbol.
- System State: Initial state are computed as follows:

Figure 25: Segregated into six gatherings of adjacent positions

1. The gait cycle limits are recognized for a mobile succession 2. Every walk cycle is apportioned in 6 gatherings of transiently adjacent positions. 3. The midpoints of all positions that have a place with a specific segment is figured and system state is identified.

- Parameter Initialization :



- Training phase: Iteration is performed in two stages : 1. Using current values of System state and Transition Matrix (Ao) , Viterbi decoding is performed on the input sequence and the most probable sequence of states is obtained Q={q1, q2, q3…qT,}. Using system state at t+1 and state transition matrix A at time t, we estimate A at time t+1 using Baum welch algorithm.
- Testing phase: Given test sequence then we compute probability P_j=log(P(y|λ_j) where y is test sequence and λ_j is HMM parameter corresponding to j'th individual in gallery.

[25] www.cs.umd.edu/~djacobs/CMSC828/ApplicationsHMMs.pdf

5.2 Handwriting Recognition

HMM's are widely used in handwriting recognition**.**

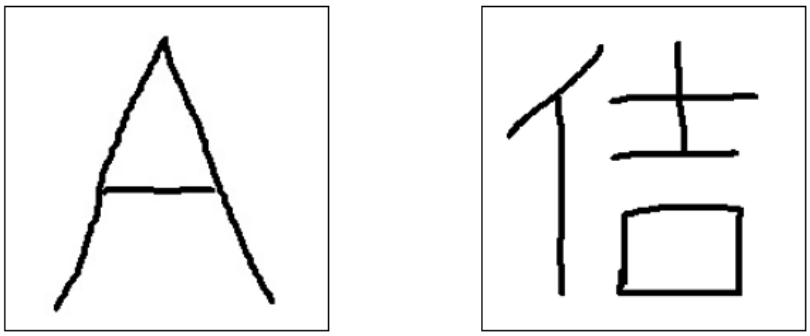


Figure 26: English alphabet and Chinese alphabet

For example to recognize English and Chinese character. This is an example for noisy channel.

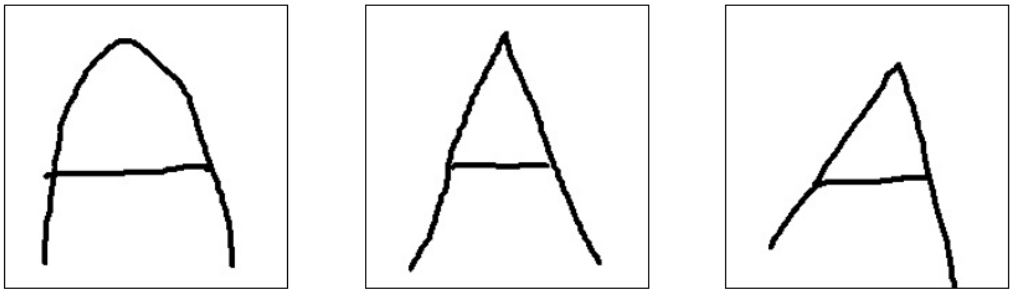


Figure 27: Different noisy inputs

To find the intended input based on noisy input received. The input can be from speech recognition software from phone or human handwriting.

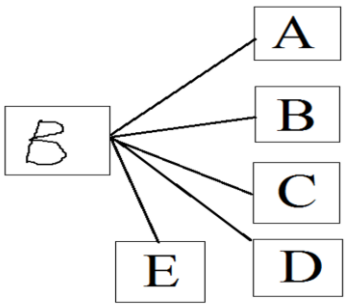


Figure 28

So based on input we need to recognize the correct one. So hmm does this for us. How?

First **pre-processing** is done. Like sharpening, applying median filter and shrinking the image.

[26,27,28] https://emunix.emich.edu/~sverdlik/CharRec.ppt

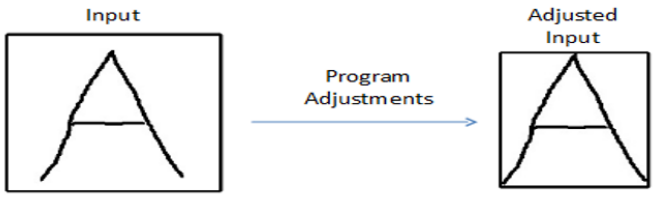
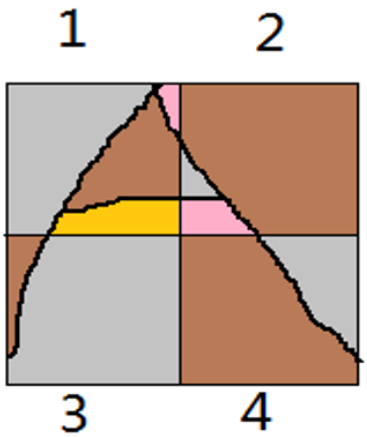


Figure 29: Preprocessing

Next step is **feature extraction.** Here number of regions is counted in each area to represent observation states



Area 1- 4 regions →S4
Area 2- 3 regions →S3
Area 3- 2 regions →S2
Area 4- 2 regions →S2
S4-S3-S2-S2

Figure 30: Dividing into areas and finding regions

Next step is to compare the areas.

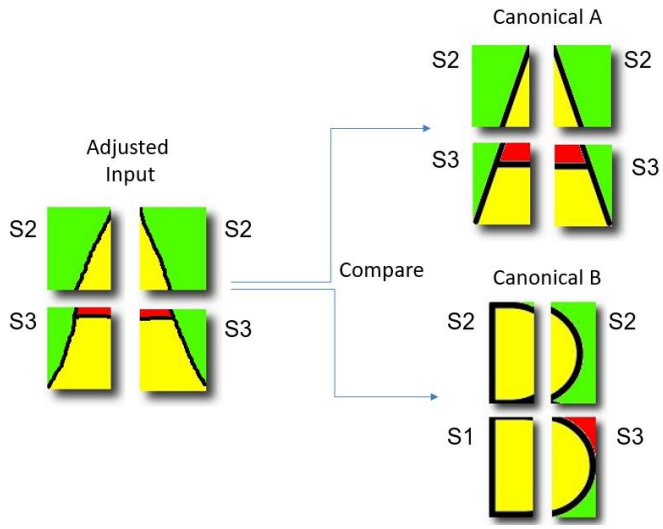


Figure 31: Comparing

[29,30,31] https://emunix.emich.edu/~sverdlik/CharRec.ppt

5.3 Computational Biology- Gene finding and prediction

HMM's can be used to predict the gene structure. DNA nucleotides have coding and non coding districts. The goal is to discover the coding and non-coding districts of an unlabeled string of DNA nucleotides.

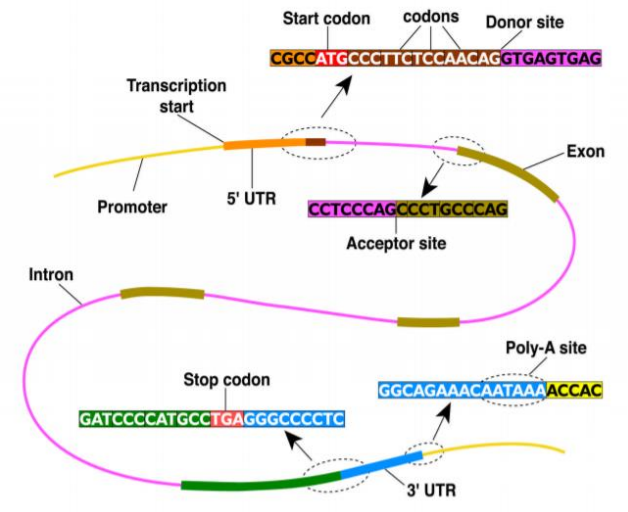


Figure 32 : Structure of DNA

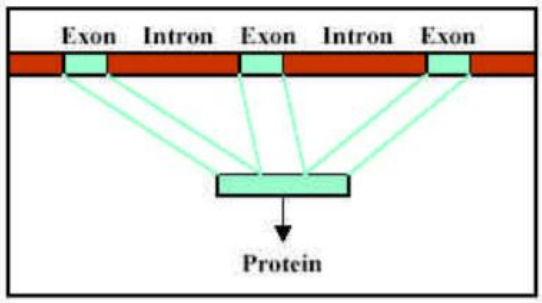


Figure 33: Protein formation

A string of DNA nucleotides containing genes have separate regions:

- **Introns**- non coding regions
- **Exons**- coding regions

These regions are separated by sites:

- Start and stop codons
- Splice sites- acceptors and donors

During process of transcription, only the exons contribute to form the protein sequence as seen in figure 28.

In figure 29 , is a straightforward model for unspliced qualities that perceives the begin codon, stop codon (just a single of the three conceivable stop codons are appeared) and the coding/non-coding districts. This model has been prepared with a test set of quality information.

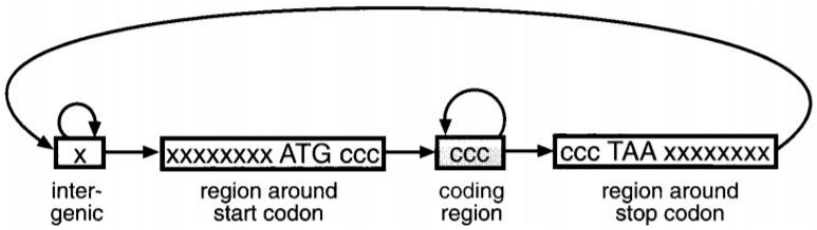


Figure 34: Gene model

Having such a model, we can predict genes in a sequence of anonymous DNA by simply using the Viterbi algorithm to find the most probable path through the model.

[32,33,34] https://pdfs.semanticscholar.org/9d0e/c3f52be23cbff7b430726f606831d497c96c.pdf

6. Conclusion

Processes with a hidden state can be modelled with HMMs, based on observable parameters. HMMs are an important tool in pattern recognition. The two main problems solved with HMMs are, determining how likely a set of observations come from a particular model, and determining the most likely sequence of hidden states. Determining appropriate model parameters is one of the stoughest problem with HMM's. An all around tuned HMM by and large gives preferred pressure over a straightforward Markov demonstrate, enabling more sequences to be fundamentally found. The Viterbi algorithm is expensive, both in terms of memory and compute time. The Viterbi algorithm and Baum-Welch algorithm calculate different things. If we know our model and we just want the latent states, then there is no reason to use the Baum-Welch algorithm. If we don't know our model, then we can't be using the Viterbi algorithm.

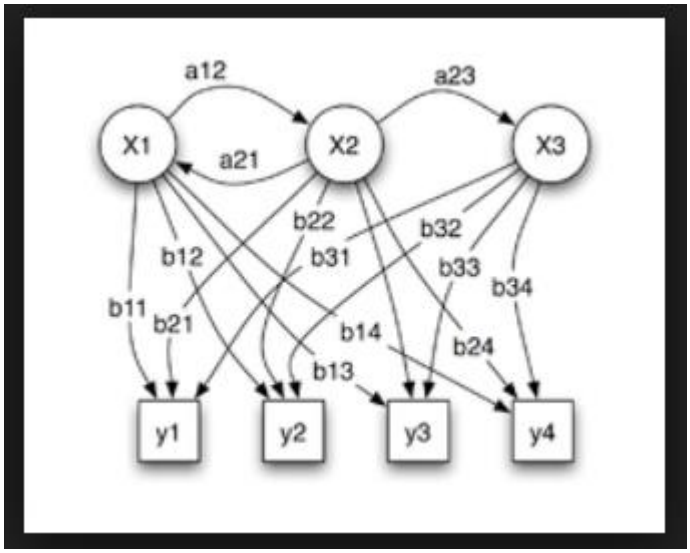


Figure 35: HMM model

[35] https://en.wikipedia.org/wiki/Hidden_Markov_model

7. References

[1] Rabiner, L.R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. IEEE 1989, 77, 257–286

[2] https://pdfs.semanticscholar.org/9d0e/c3f52be23cbff7b430726f606831d497c96c.pdf

[3] https://emunix.emich.edu/~sverdlik/CharRec.ppt

[4] http://isoft.postech.ac.kr/Course/CS704/LectureNotes/HiddenMarkovModel.pdf

[5] https://www.researchgate.net/figure/Hidden-Markov-Model_fig2_283103282

[6] D.H. Kil and F. B. Shin, Pattern Recognition and Prediction with Applications to Signal Characterization American Institute of Physics, Woodbury, NY, 1996.

[7] https://compbio.soe.ucsc.edu/html_format_papers/tr-94-24/node11.html

[8] www.cs.umd.edu/~djacobs/CMSC828/ApplicationsHMMs.pdf

[9] Viterbi, A.J. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. IEEE Trans. Informat. Theory 1967, IT-13, 260–269.

[10] L.R. Rabiner and B.H. Juang, "An Introduction to Hidden Markov Models," IEEE ASSP magazine, Vol.3, No.1, pp. 4-16, 1986.

[11] https://pdfs.semanticscholar.org/9d0e/c3f52be23cbff7b430726f606831d497c96c.pdf