**Tutorial on Hidden Markov Model**

Loc Nguyen

Sunflower Soft Company, Ho Chi Minh city, Vietnam

Email: ng\_phloc@yahoo.com

# Abstract

Hidden Markov model (HMM) is a powerful mathematical tool for prediction and recognition. Many computer software products implement HMM and hide its complexity, which assist scientists to use HMM for applied researches. However, comprehending HMM in order to take advantages of its strong points requires a lot of efforts. This report is a tutorial on HMM with full of mathematical proofs and example, which help researchers to understand it by the fastest way from theory to practice. The report focuses on three common problems of HMM such as evaluation problem, uncovering problem, and learning problem, in which learning problem with support of optimization theory is the main subject.

**Keywords:** Hidden Markov Model, Optimization, Evaluation Problem, Uncovering Problem, Learning Problem

# 1. Introduction

There are many real-world phenomena (so-called states) that we would like to model in order to explain our observations. Often, given sequence of observations symbols, there is demand of discovering real states. For example, there are some states of weather: *sunny*, *cloudy*, *rainy* (Fosler-Lussier, 1998, p. 1). Suppose you are in the room and do not know the weather outside but you are notified observations such as wind speed, atmospheric pressure, humidity, and temperature from someone else. Basing on these observations, it is possible for you to forecast the weather by using hidden Markov model (HMM). Before discussing about HMM, we should glance over the definition of Markov model (MM). First, MM is the statistical model which is used to model the stochastic process. MM is defined as below (Schmolze, 2001):

* Given a finite set of state *S*={*s*1, *s*2,…, *sn*} whose cardinality is *n*. Let ∏ be the *initial state distribution* where *πi*∏ represents the probability that the stochastic process begins in state *si*. In other words *πi* is the initial probability of state *si*, where .
* The stochastic process which is modeled gets only one state from *S* at all time points. This stochastic process is defined as a finite vector *X*=(*x*1, *x*2,…, *xT*) whose element *xt* is a state at time point *t*. The process *X* is called *state stochastic process* and *xt* *S* equals some state *si* *S*. Note that *X* is also called *state sequence*. Time point can be in terms of second, minute, hour, day, month, year, etc. It is easy to infer that the initial probability *πi* = *P*(*x*1*=si*) where *x*1 is the first state of the stochastic process. The state stochastic process *X* must meet fully the *Markov property*, namely, given previous state *xt*–1 of process *X*, the conditional probability of current state *xt* is only dependent on the previous state *xt*–1, not relevant to any further past state (*xt*–2, *xt*–3,…, *x*1). In other words, *P*(*xt | xt*–1, *xt*–2, *xt*–3,…, *x*1) = *P*(*xt | xt*–1) with note that *P*(.) also denotes probability in this research. Such process is called first-order Markov process.
* At each time point, the process changes to the next state based on the *transition probability distribution* *aij*, which depends only on the previous state. So *aij* is the probability that the stochastic process changes current state *si* to next state *sj*. It means that *aij* = *P*(*xt=sj* | *xt–*1*=si*) = *P*(*xt+*1*=sj* | *xt=si*). The probability of transitioning from any given state to some next state is 1*,* we have. All transition probabilities *aij* (s) constitute the *transition probability matrix* *A*. Note that *A* is *n* by *n* matrix because there are *n* distinct states. It is easy to infer that matrix *A* represents state stochastic process *X*. It is possible to understand that the initial probability matrix ∏ is degradation case of matrix *A*.

Briefly, MM is the triple 〈*S*, *A*,∏〉. In typical MM, states are observed directly by users and transition probabilities (*A* and ∏) are unique parameters. Otherwise, hidden Markov model (HMM) is similar to MM except that the underlying states become hidden from observer, they are hidden parameters. HMM adds more output parameters which are called observations. Each state (hidden parameter) has the conditional probability distribution upon such observations. HMM is responsible for discovering hidden parameters (states) from output parameters (observations), given the stochastic process. The HMM has further properties as below (Schmolze, 2001):

* Suppose there is a finite set of possible observations Φ = {*φ*1, *φ*2,…, *φm*} whose cardinality is *m*. There is the second stochastic process which produces *observations* correlating with hidden states. This process is called *observable stochastic process*, which is defined as a finite vector *O* = (*o*1, *o*2,…, *oT*) whose element *ot* is an observation at time point *t*. Note that *ot* Φ equals some *φk*. The process *O* is often known as *observation sequence*.
* There is a probability distribution of producing a given observation in each state. Let *bi*(*k*) be the probability of observation *φk* when the state stochastic process is in state *si*. It means that *bi*(*k*) = *bi*(*ot=φk*) = *P*(*ot=φk* | *xt=si*). The sum of probabilities of all observations which observed in a certain state is 1, we have. All probabilities of observations *bi*(*k*) constitute the *observation probability matrix* *B*. It is convenient for us to use notation *bik* instead of notation *bi*(*k*). Note that *B* is *n* by *m* matrix because there are *n* distinct states and *m* distinct observations. While matrix *A* represents state stochastic process *X*, matrix *B* represents observable stochastic process *O*.

Thus, HMM is the 5-tuple ∆ = 〈*S*, Φ, *A*, *B*,∏〉. Note that components *S*,Φ, *A*, *B*,and∏ are often called parameters of HMM in which *A*, *B*,and∏ are essential parameters. Going back weather example, suppose you need to predict how weather tomorrow is: *sunny*, *cloudy* or *rainy* since you know only observations about the humidity: *dry*, *dryish*, *damp*, *soggy*. The HMM is totally determined based on its parameters *S*,Φ, *A*, *B*,and∏ according to weather example. We have *S* = {*s*1=*sunny*, *s*2=*cloudy*, *s*3=*rainy*}, Φ = {*φ*1=*dry*, *φ*2=*dryish*, *φ*3=*damp*, *φ*4=*soggy*}. Transition probability matrix *A* is shown in table 1.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Weather current day  (Time point *t*) | | |
| *sunny* | *cloudy* | *rainy* |
| Weather previous day  (Time point *t* –1) | *sunny* | *a*11=0.50 | *a*12=0.25 | *a*13=0.25 |
| *cloudy* | *a*21=0.30 | *a*22=0.40 | *a*23=0.30 |
| *rainy* | *a*31=0.25 | *a*32=0.25 | *a*33=0.50 |

**Table 1.1.** Transition probability matrix *A*

From table 1.1, we have *a*11+*a*12+*a*13=1, *a*21+*a*22+*a*23=1, *a*31+*a*32+*a*33=1.

Initial state distribution specified as uniform distribution is shown in table 1.2.

|  |  |  |
| --- | --- | --- |
| *sunny* | *cloudy* | *rainy* |
| *π*1=0.33 | *π*2=0.33 | *π*3=0.33 |

**Table 1.2.** Uniform initial state distribution ∏

From table 1.2, we have *π*1+*π*2+*π*3=1.

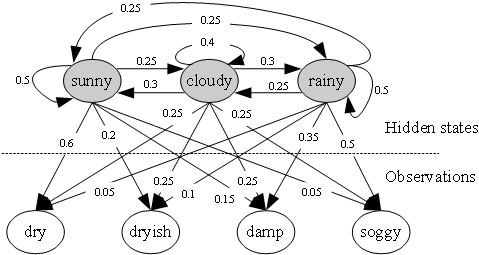
Observation probability matrix *B* is shown in table 1.3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Humidity | | | |
| *dry* | *dryish* | *damp* | *soggy* |
| Weather | *sunny* | *b*11=0.60 | *b*12=0.20 | *b*13=0.15 | *b*14=0.05 |
| *cloudy* | *b*21=0.25 | *b*22=0.25 | *b*23=0.25 | *b*24=0.25 |
| *rainy* | *b*31=0.05 | *b*32=0.10 | *b*33=0.35 | *b*34=0.50 |

**Table 1.3.** Observation probability matrix *B*

From table 1.3, we have *b*11+*b*12+*b*13+*b*14=1, *b*21+*b*22+*b*23+*b*24=1, *b*31+*b*32+*b*33+*b*34=1.

The whole weather HMM is depicted in figure 1.1.



**Figure 1.1.** HMM of weather forecast (hidden states are shaded)

There are three problems of HMM (Schmolze, 2001) (Rabiner, 1989, pp. 262-266):

1. Given HMM ∆ and an observation sequence *O* = {*o*1, *o*2,…, *oT*} where *ot* Φ, how to calculate the probability *P*(*O*|*∆*) of this observation sequence. Such probability *P*(*O*|*∆*) indicates how much the HMM ∆ affects on sequence *O*. This is *evaluation problem* or *explanation problem*. Note that it is possible to denote *O* = {*o*1 → *o*2 →…→ *oT*} and the sequence *O* is aforementioned observable stochastic process.
2. Given HMM ∆ and an observation sequence *O* = {*o*1, *o*2,…, *oT*} where *ot* Φ, how to find the sequence of states *X* = {*x*1, *x*2,…, *xT*} where *xt* *S* so that *X* is most likely to have produced the observation sequence *O*. This is *uncovering problem*. Note that the sequence *X* is aforementioned state stochastic process.
3. Given HMM ∆ and an observation sequence *O* = {*o*1, *o*2,…, *oT*} where *ot* Φ, how to adjust parameters of ∆ such as initial state distribution ∏, transition probability matrix *A*, and observation probability matrix *B* so that the quality of HMM ∆ is enhanced. This is *learning problem*.

These problems will be mentioned in sections 2, 3, and 4, in turn.

# 2. HMM evaluation problem

The essence of evaluation problem is to find out the way to compute the probability *P*(*O*|*∆*) most effectively given the observation sequence *O* = {*o*1, *o*2,…, *oT*}. For example, given HMM ∆ whose parameters *A*, *B*,and∏ specified in tables 1.1, 1.2, and 1.3, which is designed for weather forecast. Suppose we need to calculate the probability of event that humidity is *soggy*, *dry*, and *dryish* in days 1, 2, and 3, respectively. This is evaluation problem with sequence of observations *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*}. There is a complete set of 33=27 mutually exclusive cases of weather states for three days as follows:

* Weather states in days 1, 2, and 3 are *sunny*, *sunny*, and *sunny*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*1=*sunny*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *sunny*, *sunny*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*1=*sunny*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *sunny*, *sunny*, and *rainy*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*1=*sunny*, *x*3=*s*3=*rainy*}.
* Weather states in days 1, 2, and 3 are *sunny*, *cloudy*, and *sunny*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*2=*cloudy*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *sunny*, *cloudy*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*2=*cloudy*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *sunny*, *cloudy*, and *rainy*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*2=*cloudy*, *x*3=*s*3=*rainy*}.
* Weather states in days 1, 2, and 3 are *sunny*, *rainy*, and *sunny*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*3=*rainy*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *sunny*, *rainy*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*3=*rainy*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *sunny*, *rainy*, and *rainy*. State stochastic process is *X* = {*x*1=*s*1=*sunny*, *x*2=*s*3=*rainy*, *x*3=*s*3=*rainy*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *sunny*, and *sunny*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*1=*sunny*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *sunny*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*1=*sunny*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *sunny*, and *rainy*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*1=*sunny*, *x*3=*s*3=*rainy*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *cloudy*, and *sunny*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*2=*cloudy*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *cloudy*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*2=*cloudy*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *cloudy*, and *rainy*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*2=*cloudy*, *x*3=*s*3=*rainy*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *rainy*, and *sunny*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*3=*rainy*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *rainy*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*3=*rainy*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *cloudy*, *rainy*, and *rainy*. State stochastic process is *X* = {*x*1=*s*2=*cloudy*, *x*2=*s*3=*rainy*, *x*3=*s*3=*rainy*}.
* Weather states in days 1, 2, and 3 are *rainy*, *sunny*, and *sunny*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*1=*sunny*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *rainy*, *sunny*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*1=*sunny*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *rainy*, *sunny*, and *rainy*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*1=*sunny*, *x*3=*s*3=*rainy*}.
* Weather states in days 1, 2, and 3 are *rainy*, *cloudy*, and *sunny*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*2=*cloudy*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *rainy*, *cloudy*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*2=*cloudy*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *rainy*, *cloudy*, and *rainy*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*2=*cloudy*, *x*3=*s*3=*rainy*}.
* Weather states in days 1, 2, and 3 are *rainy*, *rainy*, and *sunny*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*3=*rainy*, *x*3=*s*1=*sunny*}.
* Weather states in days 1, 2, and 3 are *rainy*, *rainy*, and *cloudy*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*3=*rainy*, *x*3=*s*2=*cloudy*}.
* Weather states in days 1, 2, and 3 are *rainy*, *rainy*, and *rainy*. State stochastic process is *X* = {*x*1=*s*3=*rainy*, *x*2=*s*3=*rainy*, *x*3=*s*3=*rainy*}.

According to total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101), the probability *P*(*O*|*∆*) is:

We have:

(Because observations *o*1, *o*2, and *o*3 are mutually independent)

(Because an observation is only dependent on the day when it is observed)

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Due to Markov property, current state is only dependent on right previous state)

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(According toparameters *A*, *B*,and∏ specified in tables 1.1, 1.2, and 1.3)

Similarly, we have:

It implies

It is easy to explain that given weather HMM modeled by parameters *A*, *B*,and∏ specified in tables 1.1, 1.2, and 1.3, the event that it is *soggy*, *dry*, and *dryish* in three successive days is rare because the probability of such event *P*(*O|*Δ) is low (1.3%). It is easy to recognize that it is impossible to browse all combinational cases of given observation sequence *O* = {*o*1, *o*2,…, *oT*} as we knew that it is necessary to survey 33=27 mutually exclusive cases of weather states with a tiny number of observations {*soggy*, *dry*, *dryish*}. Exactly, given *n* states and *T* observations, it takes extremely expensive cost to survey *nT* cases. According to (Rabiner, 1989, pp. 262-263), there is a so-called *forward-backward procedure* to decrease computational cost for determining the probability *P*(*O|*Δ). Let *αt*(*i*) be the joint probability of partial observation sequence {*o*1, *o*2,…, *ot*} and state *xt*=*si* where , specified by equation 2.1.

|  |  |
| --- | --- |
|  | (2.1) |

The joint probability *αt*(*i*) is also called *forward variable* at time point *t* and state *si*. The product *αt*(*i*)*aij* where *aij* is the transition probability from state *i* to state *j* counts for probability of join event that partial observation sequence {*o*1, *o*2,…, *ot*} exists and the state *si* at time point *t* is changed to *sj* at time point *t+*1.

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Because the partial observation sequence {*o*1, *o*2,…, *ot*} is independent from next state *xt+*1 given current state *xt*)

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

Summing product *αt*(*i*)*aij* over all *n* possible states of *xt* produces probability of join event that partial observation sequence {*o*1, *o*2,…, *ot*} exists and the next state is *xt+*1=*sj* regardless of the state *xt*.

The forward variable at time point *t+*1 and state *sj* is calculated on *αt*(*i*) as follows:

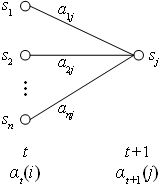
(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Due to observations are mutually independent)

Where *bj*(*ot+*1) is the probability of observation *ot+*1 when the state stochastic process is in state *sj*, please see an example of observation probability matrix shown in table 1.3. In brief, please pay attention to recurrence property of forward variable specified by equation 2.2.

|  |  |
| --- | --- |
|  | (2.2) |

The aforementioned construction of forward recurrence equation 2.2 is essentially to build up Markov chain, illustrated by figure 2.1 (Rabiner, 1989, p. 262).



**Figure 2.1.** Construction of recurrence equation for forward variable

According to the forward recurrence equation 2.2, given observation sequence *O* = {*o*1, *o*2,…, *oT*}, we have:

The probability *P*(*O|*Δ) is sum of *αT*(*i*) over all *n* possible states of *xT*, specified by equation 2.3.

|  |  |
| --- | --- |
|  | (2.3) |

The forward-backward procedure to calculate the probability *P*(*O|*Δ), based on forward equations 2.2 and 2.3, includes three steps as shown in table 2.1 (Rabiner, 1989, p. 262).

|  |
| --- |
| 1. Initialization step: Initializing *α*1(*i*) = *bi*(*o*1)*πi* for all 2. Recurrence step: Calculating all *αt+*1(*j*) for all and according to equation 2.2. 3. Evaluation step: Calculating the probability |

**Table 2.1.** Forward-backward procedure based on forward variable to calculate the probability *P*(*O|*Δ)

It is required to execute *n*+2*n*2(*T*–1)+*n*–1 = 2*n*2(*T*–1)+2*n*–1 operations for forward-backward procedure based on forward variable due to:

* There are *n* multiplications at initialization step.
* There are *n* multiplications, *n*–1 additions, and 1 multiplication over the expression . There are *n* cases of values *αt+*1(*j*) for all at time point *t+*1. So, there are (*n*+*n*–1+1)*n* = 2*n*2 operations over values *αt+*1(*j*) for all at time point *t*. The recurrence step runs over *T*–1 times and so, there are 2*n*2(*T*–1) operations at recurrence step.
* There are *n*–1 additions at evaluation step.

Inside 2*n*2(*T*–1)+2*n*–1 operations, there are *n*+(*n*+1)*n*(*T*–1) = *n*+(*n*2+*n*)(*T*–1) multiplications and (*n*–1)*n*(*T*–1)+*n*–1 = (*n*2+*n*)(*T*–1)+*n*–1 additions.

Going back example with weather HMM whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3. We need to re-calculate the probability of observation sequence *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*} by forward-backward procedure shown in table 2.1 (based on forward variable). According to initialization step of forward-backward procedure based on forward variable, we have:

According to recurrence step of forward-backward procedure based on forward variable, we have:

According to evaluation step of forward-backward procedure based on forward variable, the probability of observation sequence *O* = {*o*1=*s*4=*soggy*, *o*2=*s*1=*dry*, *o*3=*s*2=*dryish*} is:

The result from the forward-backward procedure based on forward variable is the same to the one from aforementioned brute-force method that browses all 33=27 mutually exclusive cases of weather states.

There is interesting thing that the forward-backward procedure can be implemented based on so-called *backward variable*. Let *βt*(*i*) be the backward variable which is conditional probability of partial observation sequence {*ot*, *ot+*1,…, *oT*} given state *xt*=*si* where , specified by equation 2.4.

|  |  |
| --- | --- |
|  | (2.4) |

We have

(Because observations *ot+*1, *ot+*2,…, *oT* are mutually independent)

(Because partial observation sequence *ot+*1, *ot+*2,…, *oT* is independent from state *xt* at time point t)

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

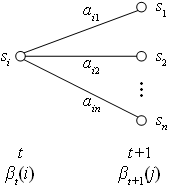
Summing the product *aijbj*(*ot+*1)*βt+*1(*j*) over all *n* possible states of *xt+*1=*sj*, we have:

(Due to the total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101))

In brief, the recurrence property of backward variable specified by equation 2.5.

|  |  |
| --- | --- |
|  | (2.5) |

Where *bj*(*ot+*1) is the probability of observation *ot+*1 when the state stochastic process is in state *sj*, please see an example of observation probability matrix shown in table 1.3. The construction of backward recurrence equation 2.5 is essentially to build up Markov chain, illustrated by figure 2.2 (Rabiner, 1989, p. 263).



**Figure 2.2.** Construction of recurrence equation for backward variable

According to the backward recurrence equation 2.5, given observation sequence *O* = {*o*1, *o*2,…, *oT*}, we have:

The product *πibi*(*o*1)*β*1(*i*) is:

(Because observations *o*1, *o*2,…, *oT* are mutually independent)

It implies that the probability *P*(*O|*Δ) is:

(Due to the total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101))

Shortly, the probability *P*(*O|*Δ) is sum of product *πibi*(*o*1)*β*1(*i*) over all *n* possible states of *x*1=*si*, specified by equation 2.6.

|  |  |
| --- | --- |
|  | (2.6) |

The forward-backward procedure to calculate the probability *P*(*O|*Δ), based on backward equations 2.5 and 2.6, includes three steps as shown in table 2.2 (Rabiner, 1989, p. 263).

|  |
| --- |
| 1. Initialization step: Initializing *βT*(*i*) = 1for all 2. Recurrence step: Calculating all *βt*(*i*) for all and *t=T*–1, *t=T*–2,…, *t=*1, according to equation 2.5. 3. Evaluation step: Calculating the probability *P*(*O|*Δ) according to equation 2.6, |

**Table 2.2.** Forward-backward procedure based on backward variable to calculate the probability *P*(*O|*Δ)

It is required to execute (3*n*–1)*n*(*T*–1)+2*n*+*n*–1 = 3*n*2(*T*–1)–*n*(*T–*4)–1 operations for forward-backward procedure based on forward variable due to:

* There are 2*n* multiplications and *n*–1 additions over the sum . So, there are (2*n*+*n*–1)*n* = (3*n*–1)*n* operations over values *βt*(*i*) for all at time point *t*. The recurrence step runs over *T*–1 times and so, there are (3*n*–1)*n*(*T*–1) operations at recurrence step.
* There are 2*n* multiplications and *n*–1 additions over the sum at evaluation step.

Inside 3*n*2(*T*–1)–*n*(*T–*4)–1 operations, there are 2*n*2(*T*–1)+2*n* multiplications and (*n*–1)*n*(*T*–1)+*n*–1 = *n*2(*T*–1)–*n*(*T*–2)–1 additions.

Going back example with weather HMM whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3. We need to re-calculate the probability of observation sequence *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*} by forward-backward procedure shown in table 2.2 (based on backward variable). According to initialization step of forward-backward procedure based on backward variable, we have:

According to recurrence step of forward-backward procedure based on backward variable, we have:

According to evaluation step of forward-backward procedure based on backward variable, the probability of observation sequence *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*} is:

The result from the forward-backward procedure based on backward variable is the same to the one from aforementioned brute-force method that browses all 33=27 mutually exclusive cases of weather states and the one from forward-backward procedure based on forward variable.

The research applies solution of uncovering problem into building up learning style model. Thus, the uncovering problem is mentioned particularly in the successive section.

# 3. HMM uncovering problem

Recall that given HMM ∆ and observation sequence *O* = {*o*1, *o*2,…, *oT*} where *ot* Φ, how to find out a state sequence *X* = {*x*1, *x*2,…, *xT*} where *xt S* so that *X* is most likely to have produced the observation sequence *O*. This is the uncovering problem: which sequence of state transitions is most likely to have led to given observation sequence. In other words, it is required to establish an *optimal criterion* so that the state sequence *X* leads to maximizing such criterion. The simple criterion is the conditional probability of sequence *X* with respect to sequence *O* and model ∆, denoted *P*(*X*|*O*,∆). We can apply brute-force strategy: “go through all possible such *X* and pick the one leading to maximizing the criterion *P*(*X|O*,∆)”.

This strategy is impossible if the number of states and observations is huge. Another popular way is to establish a so-called *individually optimal criterion* (Rabiner, 1989, p. 263) which is described right later.

Let *γt*(*i*) be joint probability that the stochastic process is in state *si* at time point *t* with observation sequence *O* = {*o*1, *o*2,…, *oT*}, equation 3.1 specifies this probability based on forward variable *αt* and backward variable *βt*.

|  |  |
| --- | --- |
|  | (3.1) |

The variable *γt*(*i*) is also called *individually optimal criterion* with note that forward variable *αt* and backward variable *βt* are calculated according to recurrence equations 2.2 and 2.5, respectively. Following is proof of equation 3.1.

(Due to Bayes’ rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 99))

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Because observations *o*1, *o*2,…, *oT* are observed independently)

(According to equations 2.1 and 2.4 for determining forward variable and backward variable)

The state sequence *X* = {*x*1, *x*2,…, *xT*} is determined by selecting each state *xt* *S* so that it maximizes *γt*(*i*).

(Due to Bayes’ rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 99))

(Due to equation 3.1)

Because the probability is not relevant to state sequence *X*, it is possible to remove it from the optimization criterion. Thus, equation 3.2 specifies how to find out the optimal state *xt* of *X* at time point *t*.

|  |  |
| --- | --- |
|  | (3.2) |

Note that index *i* is identified with state according to equation 3.2. The optimal state *xt* of *X* at time point *t* is the one that maximizes product *αt*(*i*) *βt*(*i*) over all values *si*. The procedure to find out state sequence *X* = {*x*1, *x*2,…, *xT*} based on individually optimal criterion is called *individually optimal procedure* that includes three steps, shown in table 3.1.

|  |
| --- |
| 1. Initialization step:  * Initializing *α*1(*i*) = *bi*(*o*1)*πi* for all * Initializing *βT*(*i*) = 1for all  1. Recurrence step:  * Calculating all *αt+*1(*i*) for all and according to equation 2.2. * Calculating all *βt*(*i*) for all and *t=T*–1, *t=T*–2,…, *t=*1, according to equation 2.5. * Calculating all *γt*(*i*)=*αt*(*i*)*βt*(*i*) for all and according to equation 3.1. * Determining optimal state *xt* of *X* at time point *t* is the one that maximizes *γt*(*i*) over all values *si*.  1. Final step: The state sequence *X* = {*x*1, *x*2,…, *xT*} is totally determined when its partial states *xt* (s) where are found in recurrence step. |

**Table 3.1.** Individually optimal procedure to solve uncovering problem

It is required to execute *n*+(5*n*2–*n*)(*T*–1)+2*nT* operations for individually optimal procedure due to:

* There are *n* multiplications for calculating *α*1(*i*) (s).
* The recurrence step runs over *T*–1 times. There are 2*n*2(*T*–1) operations for determining *αt+*1(*i*) (s) over all and . There are (3*n*–1)*n*(*T*–1) operations for determining *βt*(*i*) (s) over all and *t=T*–1, *t=T*–2,…, *t=*1. There are *nT* multiplications for determining *γt*(*i*)=*αt*(*i*)*βt*(*i*) over all and . There are *nT* comparisons for determining optimal state over all and . In general, there are 2*n*2(*T*–1)+ (3*n*–1)*n*(*T*–1) + *nT* + *nT* = (5*n*2–*n*)(*T*–1) + 2*nT* operations at the recurrence step.

Inside *n*+(5*n*2–*n*)(*T*–1)+2*nT* operations, there are *n*+(*n*+1)*n*(*T*–1)+2*n*2(*T*–1)+*nT* = (3*n*2+*n*)(*T*–1)+*nT*+*n* multiplications and (*n*–1)*n*(*T*–1)+(*n*–1)*n*(*T*–1) = 2(*n*2–*n*)(*T*–1) additions and *nT* comparisons.

For example, given HMM ∆ whose parameters *A*, *B*,and∏ specified in tables 1.1, 1.2, and 1.3, which is designed for weather forecast. Suppose humidity is *soggy* and *dry* in days 1 and 2, respectively. We apply individual optimal procedure into solving the uncovering problem that finding out the optimal state sequence *X* = {*x*1, *x*2} with regard to observation sequence *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*}. According to equations 2.2 and 2.5, forward variable and backward variable are calculated as follows:

According to recurrence step of individually optimal procedure, individually optimal criterion *γt*(*i*) and optimal state *xt* are calculated as follows:

As a result, the optimal state sequence is *X* = {*x*1=*rainy*, *x*2=*sunny*, *x*3=*sunny*}.

The individually optimal criterion *γt*(*i*) does not reflect the whole probability of state sequence *X* given observation sequence *O* because it focuses only on how to find out each partially optimal state *xt* at each time point *t*. Thus, the individually optimal procedure is heuristic method. Viterbi algorithm (Rabiner, 1989, p. 264) is alternative method that takes interest in the whole state sequence *X* by using joint probability *P*(*X*,*O|*Δ) of state sequence and observation sequence as optimal criterion for determining state sequence *X*. Let *δt*(*i*) be the maximum joint probability of observation sequence *O* and state *xt*=*si* over *t*–1 previous states. The quantity *δt*(*i*) is called *joint optimal criterion* at time point *t*, which is specified by equation 3.3.

|  |  |
| --- | --- |
|  | (3.3) |

The recurrence property of *joint optimal criterion* is specified by equation 3.4.

|  |  |
| --- | --- |
|  | (3.4) |

The semantic content of joint optimal criterion *δt* is similar to the forward variable *αt*. Following is the proof of equation 3.4.

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Due to observations are mutually independent)

(The probability *bj*(*ot+*1) is moved out of the maximum operation because it is independent from states *x*1, *x*2,…, *xt*)

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Because observation *xt+*1 is dependent from *o*1, *o*2,…, *ot*, *x*1, *x*2,…, *xt–*1)

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

Given criterion *δt+*1(*j*), the state *xt+*1=*sj* that maximizes *δt+*1(*j*) is stored in the backtracking state *qt+*1(*j*) that is specified by equation 3.5.

|  |  |
| --- | --- |
|  | (3.5) |

Note that index *i* is identified with state according to equation 3.5. The Viterbi algorithm based on joint optimal criterion *δt*(*i*) includes three steps described in table 3.2.

|  |
| --- |
| 1. Initialization step:  * Initializing *δ*1(*i*) = *bi*(*o*1)*πi* for all * Initializing *q*1(*i*) = 0 for all  1. Recurrence step:  * Calculating all for all and according to equation 3.4. * Keeping tracking optimal states for all and according to equation 3.5.  1. State sequence backtracking step: The resulted state sequence *X* = {*x*1, *x*2,…, *xT*} is determined as follows:  * The last state * Previous states are determined by backtracking: *xt* = *qt+*1(*xt+*1) for *t=T*–1, *t=T*–2,…, *t=*1. |

**Table 3.2.** Viterbi algorithm to solve uncovering problem

The total number of operations inside the Viterbi algorithm is 2*n+*(2*n*2*+n*)(*T*–1) as follows:

* There are *n* multiplications for initializing *n* values *δ*1(*i*) when each *δ*1(*i*) requires 1 multiplication.
* There are (2*n*2*+n*)(*T*–1) operations over the recurrence step because there are *n*(*T*–1) values *δt+*1(*j*) and each *δt+*1(*j*) requires *n* multiplications and *n* comparisons for maximizing plus 1 multiplication.
* There are *n* comparisons for constructing the state sequence *X*, .

Inside 2*n+*(2*n*2*+n*)(*T*–1) operations, there are *n*+(*n*2+*n*)(*T*–1) multiplications and *n*2(*T*–1)+*n* comparisons. The number of operations with regard to Viterbi algorithm is smaller than the number of operations with regard to individually optimal procedure when individually optimal procedure requires (5*n*2–*n*)(*T*–1)+2*nT*+*n* operations. Therefore, Viterbi algorithm is more effective than individually optimal procedure. Besides, individually optimal procedure does not reflect the whole probability of state sequence *X* given observation sequence *O*.

Going backing the weather HMM ∆ whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3. Suppose humidity is *soggy* and *dry* in days 1 and 2, respectively. We apply Viterbi algorithm into solving the uncovering problem that finding out the optimal state sequence *X* = {*x*1, *x*2, *x*3} with regard to observation sequence *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*}. According to initialization step of Viterbi algorithm, we have:

According to recurrence step of Viterbi algorithm, we have:

According to state sequence backtracking of Viterbi algorithm, we have:

As a result, the optimal state sequence is *X* = {*x*1=*rainy*, *x*2=*sunny*, *x*3=*sunny*}. The result from the Viterbi algorithm is the same to the one from aforementioned individually optimal procedure described in table 3.1.

Essentially, Viterbi algorithm maximizes the joint probability *P*(*X*,*O|*∆) instead of maximizing the conditional probability *P*(*X*|*O*,∆). I propose so-called *longest-path algorithm* based on longest path of graph for solving uncovering problem. This algorithm that maintains using the conditional probability *P*(*X*|*O*,∆) as optimal criterion gives a viewpoint different from the viewpoint of Viterbi algorithm although it is easy for you to recognize that the ideology of the longest-path algorithm does not go beyond the ideology of Viterbi algorithm after you comprehend the longest-path algorithm. Following is description of longest-path algorithm.

The optimal criterion *P*(*X*|*O*,∆) of graphic method is:

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Due to Markov property: the probability of current state is only dependent on the probability of right previous state)

(Because an observation is only dependent on the time point when it is observed)

By recurrence calculation on probability , we have:

Applying Bayes’ rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 99) into the probability , we have:

(Applying Bayes’ rule into the probability )

(Applying Bayes’ rule into the probability )

(Because an observation is only dependent on the time point when it is observed, )

Applying Bayes’ rule into the probability by another way, we have:

(Because an observation is only dependent on the time point when it is observed, )

(Applying multiplication rule into the probability )

Because we had , it implies that

We have

Where,

Because the constant *c* is independent from state transitions, maximizing the criterion *P*(*X*|*O*,∆) with regard to state transitions is the same to maximizing the product *w*1*w*2…*wt*…*wT*. Let *ρ* be this product and so, *ρ* is the optimal criterion of longest-path algorithm, re-written by equation 3.6.

|  |  |
| --- | --- |
|  | (3.6) |

Where,

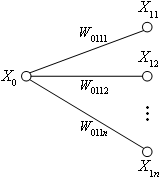
The essence of longest-path algorithm is to construct a graph and then, the algorithm finds out the longest path inside such graph with attention that the optimal criterion *ρ* represents length of every path inside the graph. There is an interesting thing that such length *ρ* is product of weights instead of sequence of additions as usual. The criterion *ρ* is function of state transitions and the longest-path algorithm aims to maximize *ρ*. Following is description of how to build up the graph.

Each *wt* representing the influence of state *xt* on the observation sequence *O* = {*o*1, *o*2,…, *oT*} at time point *t* is dependent on states *xt*–1 and *xt*. We will create a graph from these *wt* (*s*). Because there are *n* possible values of *xt*, the state *xt* is decomposed into *n* nodes *Xt*1, *Xt*2,…, *Xtn*. There are *T* time points, we have *nT* time nodes. Let *X* = {*X*0, *X*11, *X*12,…, *X*1*n*, *X*21, *X*22,…, *X*2*n*,…, *XT*1, *XT*2,…, *XTn*} be a set of 1+*nT* nodes where *X*0 is null node. Firstly, we create *n* weighted arcs from node *X*0 to *n* nodes *X*11, *X*12,…, *X*1*n* at the first time point. These directed arcs are denoted *W*0111, *W*0112,…, *W*011*n* and their weights are also denoted *W*0111, *W*0112,…, *W*011*n*. These weights *W*011*j* (s) at the first time point are calculated according to *w*1 (see equation 3.6). Equation 3.7 determines *W*1*j* (s).

|  |  |
| --- | --- |
|  | (3.7) |

*Your attention please, it is conventional that* *W*0*i*1*j* *is equal to* *W*011*j*, because the null node *X*0 has no state.

Moreover, these weights *W*011*j* (s) are depicted by figure 3.1.



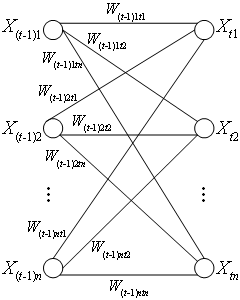
**Figure 3.1.** Weighted arcs from null node *X*0 to *n* nodes *X*11, *X*12,…, *X*1*n*

For example, given weather HMM ∆ whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3, suppose observation sequence is *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*}, we have 3 weights at the initial time point as follows:

For each node *X*(*t–*1)*i* where *t* > 1, we create *n* weighted arcs from node *X*(*t–*1)*i* to *n* nodes *Xt*1, *Xt*2,…, *Xtn* at the time point *t*. These directed arcs are denoted *W*(*t–*1)*it*1, *W*(*t–*1)*it*2,…, *W*(*t–*1)*itn* and their weights are also denoted *W*(*t–*1)*it*1, *W*(*t–*1)*it*2,…, *W*(*t–*1)*itn*. These weights *W*(*t–*1)*itj* at the time point *t* are calculated according to *wt* (see equation 3.6). Equation 3.8 determines *W*(*t–*1)*itj*.

|  |  |
| --- | --- |
|  | (3.8) |

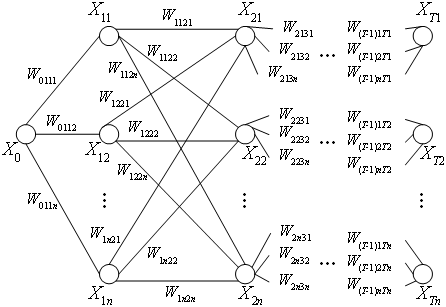
Moreover, these weights *W*(*t–*1)*itj* (s) are depicted by figure 3.2.



**Figure 3.2.** Weighted arcs from *n* node *X*(*t–*1)*i* to *n* nodes *Xtj* at time point *t*

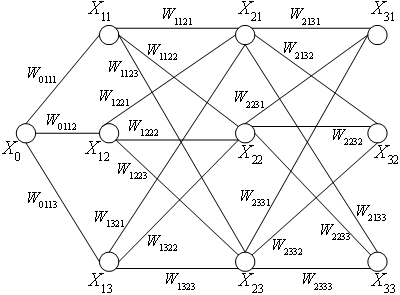
Going back given weather HMM ∆ whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3, suppose observation sequence is *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*}, we have 18 weights from time point 1 to time point 3 as follows:

In general, there are (*T*–1)*n*2 weights from time point 1 to time point *T*. Moreover, there are *n* weights derived from null node *X*0 at time point 1. Let *W* be set of *n*+(*T*–1)*n*2 weights from null node *X*0 to nodes *XT*1, *XT*2,…, *XTn* at the last time point *T*. Let *G* = <*X*, *W*> be the graph consisting of the set of nodes *X* = {*X*0, *X*11, *X*12,…, *X*1*n*, *X*21, *X*22,…, *X*2*n*,…, *XT*1, *XT*2,…, *XTn*} be a set of *n*+(*T*–1)*n*2 weights *W*. The graph *G* is called *state transition graph* shown in figure 3.3.



**Figure 3.3.** State transition graph

Please pay attention to a very important thing that both graph *G* and its weights are not determined before the longest-path algorithm is executed because there are a huge number of nodes and arcs. State transition graph shown in figure 3.3 is only illustrative example. Going back given weather HMM ∆ whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3, suppose observation sequence is *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3=*φ*2=*dryish*}, the state transition graph of this weather example is shown in figure 3.4.



**Figure 3.4.** State transition graph of weather example

The ideology of the longest-path algorithm is to solve uncovering problem by finding the longest path of state transition graph where the whole length of every path is represented by the optimal criterion *ρ* (see equation 3.6). In other words, the longest-path algorithm maximizes the optimal criterion *ρ* by finding the longest path. Let *X* = {*x*1, *x*2,…, *xT*} be the longest path of state transition graph and so, length of *X* is maximum value of the path length *ρ*. The path length *ρ* is calculated as product of weights *W*(*t–*1)*itj* (s). By heuristic assumption, *ρ* is maximized locally by maximizing weights *W*(*t–*1)*itj* (s) at each time point. The longest-path algorithm is described by pseudo-code shown in table 3.3 with note that *X* is state sequence that is ultimate result of the longest-path algorithm.

|  |
| --- |
| *X* is initialized to be empty, .  Calculating initial weights *W*0111, *W*0112,…, *W*011*n* according to equation 3.7.  Adding state *x*1=*sj* to the longest path:  For *t* = 2 to *T*  Calculating *n* weights , ,…, according to equation 3.8.  Adding state *xt=sj* to the longest path:  End for |

**Table 3.3.** Longest-path algorithm

The total number of operations inside the longest-path algorithm is 2*n*+4*n*(*T*–1) as follows:

* There are *n* multiplications for initializing *n* weights *W*0111, *W*0112,…, *W*011*n* when each weight *W*011*j* requires 1 multiplication. There are *n* comparisons due to finding maximum weight index .
* There are 3*n*(*T*–1) multiplications over the loop inside the algorithm because there are *n*(*T*–1) weights *W*(*t–*1)*jtk* over the loop and each *W*(*t–*1)*jtk* requires 3 multiplications. There are *n*(*T*–1) comparisons over the loop inside the algorithm due to finding maximum weight indices: .

Inside 2*n*+4*n*(*T*–1) operations, there are *n*+3*n*(*T*–1) multiplications and *n*+*n*(*T*–1) comparisons.

The longest-path algorithm is similar to Viterbi algorithm (see table 3.2) with regard to the aspect that the path length *ρ* is calculated accumulatively but computational equations and viewpoints of longest-path algorithm and Viterbi algorithm are different. The longest-path algorithm is more effective than Viterbi algorithm because it requires 2*n*+4*n*(*T*–1) operations while Viterbi algorithm executes 2*n+*(2*n*2*+n*)(*T*–1) operations. However, longest-path algorithm does not produce the most accurate result because the path length *ρ* is maximized locally by maximizing weights *W*(*t–*1)*itj* (s) at each time point, which leads that the resulted sequence *X* may not be global longest path. In general, the longest-path algorithm is heuristic algorithm that gives a new viewpoint of uncovering problem when applying graphic approach into solving uncovering problem.

Going back given weather HMM ∆ whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3, suppose observation sequence is *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3*=φ*2*=dryish*}, the longest-path algorithm is applied to find out the optimal state sequence *X* = {*x*1, *x*2, *x*3} as below.

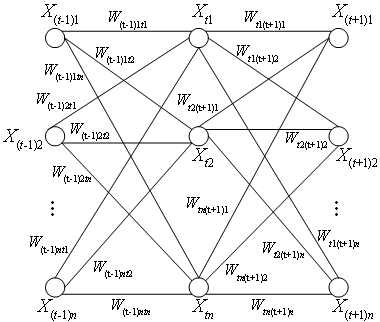
At the first time point, we have:

At the second time point, we have:

At the third time point, we have:

As a result, the optimal state sequence is *X* = {*x*1=*rainy*, *x*2=*sunny*, *x*3=*sunny*}. The result from the longest-path algorithm in this example is the same to the one from individually optimal procedure (see table 3.1) and Viterbi algorithm (see table 3.2).

The longest-path algorithm does not result out accurate state sequence *X* because it assumes that two successive nodes *X*(*t–*1)*i* and *Xtj* are mutually independent, which leads that the path length *ρ* is maximized locally by maximizing weight *W*(*t–*1)*itj* at each time point, while equation 3.6 indicates that the former node *X*(*t–*1)*i* is dependent on the prior node *Xtj*. However, according to Markov property, two intermittent nodes *X*(*t–*1)*i* and *X*(*t+*1)*k* are conditional independent given the middle node *Xtj*. This observation is very important, which helps us to enhance the accuracy of longest-path algorithm. The advanced longest-path algorithm divides the path represented by *ρ* into a set of 2-weight intervals. Each 2-weight interval includes two successive weights *W*(*t–*1)*itj* and *Wti*(*t*+1)*j* corresponding three nodes *X*(*t–*1)*i*, *Xtj*, and *X*(*t+*1)*k* where the middle node *Xtj* is also called the *midpoint* of 2-weight interval. The advanced longest-path algorithm maximizes the path *ρ* by maximizing every 2-weight interval. Each 2-weight interval has 2*n*2 connections (sub-paths) because each weight *W*(*t–*1)*itj* or *Wti*(*t*+1)*j* has *n*2 values. Figure 3.5 depicts an example of 2-weight interval.



**Figure 3.5.** 2-weight interval

The advanced longest-path algorithm is described by pseudo-code shown in table 3.4.

|  |
| --- |
| *X* is initialized to be empty, .  *i* = 1  For *t* = 1 to *T* step 2  *// Note that time point t is increased by 2 as follows: 1, 3, 5,…*  Calculating *n* weights *W*(*t–*1)*it*1, *W*(*t–*1)*it*2,…, *W*(*t–*1)*itn* according to equations 3.7 and 3.8.  For *j* = 1 to *n*  Calculating *n* weights *Wtj*(*t*+1)1, *Wtj*(*t*+1)2,…, *Wtj*(*t*+1)*n* according to equation 3.8.  End for  Adding two states and to the longest path:  End for |

**Table 3.4.** Advanced longest-path algorithm

Because two intermittent nodes *X*(*t–*1)*i* and *X*(*t+*1)*k* that are two end-points of a 2-weight interval are conditional independent given the midpoint *Xtj*, the essence of advanced longest-path algorithm is to adjust the midpoint of 2-weight interval so as to maximize such 2-weight interval.

The total number of operations inside the longest-path algorithm is (2*n*2+1.5*n*)*T* as follows:

* There are *n* multiplications for determining weights *W*(*t–*1)*it*1, *W*(*t–*1)*it*2,…, *W*(*t–*1)*itn*. Shortly, there are *nT*/2 = 0.5*nT* multiplications over the whole algorithm because time point is increased by 2.
* There are 3*n*2 multiplications for determining *n*2 weights *Wtj*(*t*+1)*l* (s) at each time point when each weight requires 3 multiplications. There are *n* multiplications for determining product . Shortly, there are (3*n*2+*n*)*T*/2 = (1.5*n*2+0.5*n*)*T* multiplications over the whole algorithm because time point is increased by 2.
* There are *n*2+*n* comparisons for maximizing: and . Shortly, there are (*n*2+*n*)*T*/2 = (0.5*n*2+0.5*n*)*T* multiplications over the whole algorithm because time point is increased by 2.

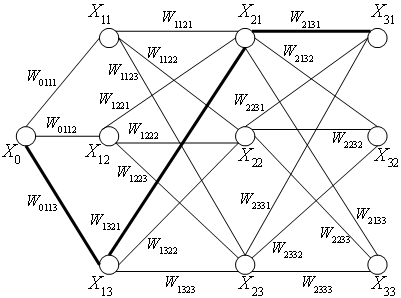
Inside (2*n*2+1.5*n*)*T* operations, there are (1.5*n*2+*n*)*T* multiplications and (0.5*n*2+0.5*n*)*T* comparisons. The advanced longest-path algorithm is not more effective than Viterbi algorithm because it requires (2*n*2+1.5*n*)*T* operations while Viterbi algorithm executes 2*n+*(2*n*2*+n*)(*T*–1) operations but it is more accurate than normal longest-path algorithm aforementioned in table 3.3.

Going back given weather HMM ∆ whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3, suppose observation sequence is *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3*=φ*2*=dryish*}, the advanced longest-path algorithm is applied to find out the optimal state sequence *X* = {*x*1, *x*2, *x*3} as follows:

At *t=*1, we have:

At *t=*3, we have:

As a result, the optimal state sequence is *X* = {*x*1=*rainy*, *x*2=*sunny*, *x*3=*sunny*}, which is the same to the one from individually optimal procedure (see table 3.1), Viterbi algorithm (see table 3.2), and normal longest-path algorithm (see table 3.3). The resulted sequence *X* = {*x*1=*rainy*, *x*2=*sunny*, *x*3=*sunny*} that is the longest path is drawn as bold line from node *X*0 to node *X*13 to node *X*21 to node *X*31 inside the state transition graph, as seen in following figure 3.6.



**Figure 3.6.** Longest path drawn as bold line inside state transition graph

The uncovering problem is described thoroughly in this section because it is very important in the research when I apply algorithms such as individually optimal procedure, Viterbi algorithm, longest-path algorithm, and advanced longest-path algorithm into solving such problem in order to predict learner’s styles given observations about her/him. The longest-path algorithm is also published in (Nguyen L. , Longest-path Algorithm to Solve Uncovering Problem of Hidden Markov Model, 2016). Successive section will mention the last problem of HMM that is the learning problem.

# 4. HMM learning problem

The learning problem is to adjust parameters such as initial state distribution ∏, transition probability matrix *A*, and observation probability matrix *B* so that given HMM ∆ gets more appropriate to an observation sequence *O* = {*o*1, *o*2,…, *oT*} with note that ∆ is represented by these parameters. In other words, the learning problem is to adjust parameters by maximizing probability of observation sequence *O*, as follows:

The Expectation Maximization (EM) algorithm is applied successfully into solving HMM learning problem, which is equivalently well-known Baum-Welch algorithm (Rabiner, 1989). Successive section 4.1 describes EM algorithm in detailed before going into Baum-Welch algorithm.

## 4.1. EM algorithm

Expectation Maximization (EM) is effective parameter estimator in case that incomplete data is composed of two parts: observed part and hidden part (missing part). EM is iterative algorithm that improves parameters after iterations until reaching optimal parameters. Each iteration includes two steps: E(xpectation) step and M(aximization) step. In E-step the hidden data is estimated based on observed data and current estimate of parameters; so the lower-bound of likelihood function is computed by the expectation of complete data. In M-step new estimates of parameters are determined by maximizing the lower-bound. Please see document (Sean, 2009) for short tutorial of EM. This sub-section focuses on practice general EM algorithm; the theory of EM algorithm is described comprehensively in article “Maximum Likelihood from Incomplete Data via the EM algorithm” by authors Dempster, Laird, and Rubin (Dempster, Laird, & Rubin, 1977).

Suppose *O* and *X* are observed data and hidden data, respectively. Note *O* and *X* can be represented in any form such as discrete values, scalar, integer number, real number, vector, list, sequence, sample, and matrix. Let represent parameters of probability distribution. Concretely, includes initial state distribution ∏, transition probability matrix *A*, and observation probability matrix *B* inside HMM*.* In other words, represents HMM Δ itself. EM algorithm aims to estimate by finding out which maximizes the likelihood function = .

Where is the optimal estimate of parameters which is called usually *parameter estimate*. Because the likelihood function is product of factors, it is replaced by the log-likelihood function *LnL*() that is natural logarithm of the likelihood function , for convenience. We have:

Where,

The method finding out the parameter estimate by maximizing the log-likelihood function is called maximum likelihood estimation (MLE). Of course, EM algorithm is based on MLE.

Suppose the current parameter is after the *tth* iteration. Next we must find out the new estimate that maximizes the next log-likelihood function . In other words it maximizes the deviation between current log-likelihood and next log-likelihood with regard to .

Where is the deviation between current log-likelihood and next log-likelihood with note that is function of when was determined.

Suppose the total probability of observed data can be determined by marginalizing over hidden data:

The expansion of is total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101). The deviation is re-written:

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

Because hidden *X* is the complete set of mutually exclusive variables, the sum of conditional probabilities of *X* is equal to 1 given *O* and .

Applying Jensen’s inequality (Sean, 2009, pp. 3-4)

into deviation , we have:

Where,

Because *C* is constant with regard to , it is possible to eliminate *C* in order to simplify the optimization criterion as follows:

The expression is essentially expectation of given conditional probability distribution when is totally determined. Let denote this conditional expectation, equation 4.1.1 specifies EM optimization criterion for determining the parameter estimate, which is the most important aspect of EM algorithm.

|  |  |
| --- | --- |
| Where, | (4.1.1) |

If is continuous density function, the continuous version of this conditional expectation is:

Finally, the EM algorithm is described in table 4.1.1.

|  |
| --- |
| Starting with initial parameter , each iteration in EM algorithm has two steps:   1. *E-step*: computing the conditional expectation based on the current parameter according to equation 4.1.1. 2. *M-step*: finding out the estimate that maximizes such conditional expectation. The next parameter is assigned by the estimate , we have:   Of course becomes current parameter for next iteration. How to maximize the conditional expectation is optimization problem which is dependent on applications. For example, the popular method to solve optimization problem is Lagrangian duality (Jia, 2013, p. 8).  EM algorithm stops when it meets the terminating condition, for example, the difference of current parameter and next parameter is smaller than some pre-defined threshold *ε*.  In addition, it is possible to define a custom terminating condition. |

**Table 4.1.1.** General EM algorithm

General EM algorithm is simple but please pay attention to the concept of lower-bound and what the essence of EM is. Recall that the next log-likelihood function is current likelihood function plus the deviation . We have:

Where,

Let denote the lower-bound of the log-likelihood function given current parameter (Sean, 2009, pp. 7-8). The lower-bound is the function of as specified by equation 4.1.2:

|  |  |
| --- | --- |
|  | (4.1.2) |

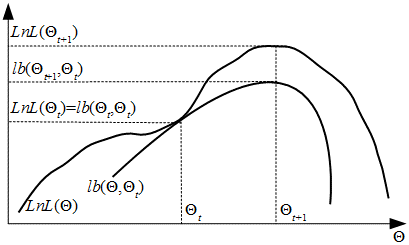
Determining is to calculate the EM conditional expectation because terms and *C* were totally determined. The lower-bound has a feature where its evaluation at equals the log-likelihood function .

In fact,

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

It implies

Figure 4.1.1 (Borman, 2004, p. 7) shows relationship between the log-likelihood function and its lower-bound .



**Figure 4.1.1.** Relationship between the log-likelihood function and its lower-bound

The essence of maximizing the deviation is to maximize the lower-bound with respect to . For each iteration the new lower-bound and its maximum are computed based on previous lower-bound. A single iteration in EM algorithm can be understood as below:

1. E-step: the new lower-bound is determined based on current parameter according to equation 4.1.2. Of course, determining is to calculate the EM conditional expectation .
2. M-step: finding out the estimate so that reaches maximum at . The next parameter is assigned by the estimate , we have:

Of course becomes current parameter for next iteration. Note, maximizing is to maximize the EM conditional expectation .

In general, it is easy to calculate the EM expectation but finding out the estimate based on maximizing such expectation is complicated optimization problem. It is possible to state that the essence of EM algorithm is to determine the estimate . Now the EM algorithm is introduced with full of details. How to apply it into solving HMM learning problem is described in successive sub-section.

## 4.2. Applying EM algorithm into solving learning problem

Now going back the HMM learning problem, the EM algorithm is applied into solving this problem, which is equivalently well-known Baum-Welch algorithm by authors Leonard E. Baum and Lloyd R. Welch (Rabiner, 1989). The parameter becomes the HMM model Δ = (*A*, *B*, ∏). Recall that the learning problem is to adjust parameters by maximizing probability of observation sequence *O*, as follows:

*Where* , , *are parameter estimates and so, the purpose of HMM learning problem is to determine them*.

The observation sequence *O* = {*o*1, *o*2,…, *oT*} and state sequence *X* = {*x*1, *x*2,…, *xT*} are observed data and hidden data within context of EM algorithm, respectively. Note *O* and *X* is now represented in sequence. According to EM algorithm, the parameter estimate is determined as follows:

Where Δ*r* = (*Ar*, *Br*, ∏*r*) is the known parameter at the current iteration. Note that we use notation Δ*r* instead of popular notation Δ*t* in order to distinguish iteration indices of EM algorithm from time points inside observation sequence *O* and state sequence *X*. The EM conditional expectation in accordance with HMM is:

(Because observations *o*1, *o*2,…, *oT* are mutually independent)

(Because each observations *ot* is only dependent on state *xt*)

(Because each state *xt* is only dependent on previous state *xt–*1)

(Due to recurrence onprobability *P*(*x*1, *x*2,…, *xt*))

It is conventional that where *x*0 is pseudo-state, equation 4.2.1 specifies general EM conditional expectation for HMM:

|  |  |
| --- | --- |
|  | (4.2.1) |

Let and are two index functions so that

We have:

Because of the convention , matrix ∏ is degradation case of matrix *A* at time point *t=*1. In other words, the initial probability *πj* is equal to the transition probability *aij* from pseudo-state *x*0 to state *x*1=*sj*.

Note that *n*=|*S*| is the number of possible states and *m*=|Φ| is the number of possible observations.

Shortly, the EM conditional expectation for HMM is specified by equation 4.2.2.

|  |  |
| --- | --- |
|  | (4.2.2) |

Where,

Note that the conditional expectation is function of Δ. There are two constraints for HMM as follows:

Maximizing with subject to these constraints is optimization problem that is solved by Lagrangian duality theorem (Jia, 2013, p. 8). Original optimization problem mentions minimizing target function but it is easy to infer that maximizing target function shares the same methodology. Let *l*(Δ, *λ*, *μ*) be Lagrangian function constructed from together with these constraints (Ramage, 2007, p. 9), we have equation 4.2.3 for specifying HMM Lagrangian function as follows:

|  |  |
| --- | --- |
|  | (4.2.3) |

Where *λ* is *n*-component vector *λ* = (*λ*1, *λ*2,…, *λn*) and *μ* is *m*-component vector *μ* = (*μ*1, *μ*2,…, *μm*). Factors *λi* ≥ 0 and *μj* ≥ 0 are called Lagrange multipliers or Karush-Kuhn-Tucker multipliers (Wikipedia, Karush–Kuhn–Tucker conditions, 2014) or dual variables. The expectation is specified by equation 4.2.2.

The parameter estimate is extreme point of the Lagrangian function. According to Lagrangian duality theorem (Boyd & Vandenberghe, 2009, p. 216) (Jia, 2013, p. 8), we have:

The parameter estimate is determined by setting partial derivatives of *l*(Δ, *λ*, *μ*) with respect to *aij* and *bj*(*k*) to be zero. The partial derivative of *l*(Δ, *λ*, *μ*) with respect to *aij* is:

Setting the partial derivative to be zero:

The parameter estimate is solution of equation , we have:

It is required to estimate the Lagrange multiplier *λi*. The multiplier estimate is determined by setting the partial derivative of *l*(Δ, *λ*, *μ*) with respect to *λi* to be zero as follows:

Substituting for *aij*, we have:

It implies:

Where, is index function.

Substituting for *λi* inside

We have:

Evaluating the numerator, we have:

(Due to total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101))

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

Evaluating the denominator, we have:

(Due to total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101))

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

It implies

Because of the convention , the estimate is fixed as follows:

The estimate of initial probability is known as specific estimate from pseudo-state *x*0 to state *x*1=*sj*. It means that

Recall that the parameter estimate is determined by setting partial derivatives of *l*(Δ, *λ*, *μ*) with respect to *aij* and *bj*(*k*) to be zero. The parameter estimate was determined. Now it is required to calculate the parameter estimate . The partial derivative of Lagrangian function *l*(Δ, *λ*, *μ*) with respect to *bj*(*k*) is:

Setting the partial derivative to be zero:

The parameter estimate is solution of equation , we have:

It is required to estimate the Lagrange multiplier *μj*. The multiplier estimate is determined by setting the partial derivative of *l*(Δ, *λ*, *μ*) with respect to *μj* to be zero as follows:

Substituting for *bj*(*k*) we have:

It implies:

Where, is index function.

Substituting for *μj* inside

We have:

Evaluating the numerator, we have:

(Due to total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101))

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

Note, the expression expresses the sum of probabilities over *T* time points with condition *ot* = *φk*.

Evaluating the denominator, we have:

(Due to total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101))

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

It implies

In general, the parameter estimate is totally determined as follows:

As a convention, we use notation Δ instead of Δ*r* for denoting known HMM at current iteration of EM algorithm. We have equation 4.2.4 for specifying HMM parameter estimate given current parameter Δ = (*aij*, *bj*(*k*), *πj*) as follows:

|  |  |
| --- | --- |
|  | (4.2.4) |

The parameter estimate is the ultimate solution of the learning problem. As seen in equation 4.2.4, it is necessary to calculate probabilities *P*(*O*, *xt–*1=*si*, *xt*=*sj*) and *P*(*O*, *xt–*1*=si*) when other probabilities *P*(*O*, *xt*=*sj*), *P*(*O*, *x*1=*si*), and *P*(*O*, *x*1=*sj*) are represented by the joint probability *γt* specified by equation 3.1.

Let *ξt*(*i*, *j*) is the joint probability that the stochastic process receives state *si* at time point *t*–1 and state *sj* at time point *t* given observation sequence *O* (Rabiner, 1989, p. 264).

Given forward variable *αt* and backward variable *βt*, if , we have:

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Because the partial observation sequence {*o*1, *o*2,…, *ot*} is independent from current state *xt* given previous state *xt–*1)

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

(Because observations *ot*, *ot+*1, *ot+*2,…, *oT* are mutually independent)

(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

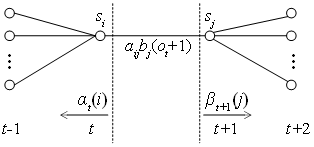
(Due to multiplication rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 100))

In general, equation 4.2.5 determines the joint probability *ξt*(*i*, *j*) based on forward variable *αt* and backward variable *βt*.

|  |  |
| --- | --- |
|  | (4.2.5) |

Where forward variable *αt* and backward variable *βt* are calculated by previous recurrence equations 2.2 and 2.5.

Shortly, the joint probability *ξt*(*i*, *j*) is constructed from forward variable and backward variable, as seen in figure 4.2.1 (Rabiner, 1989, p. 264).



**Figure 4.2.1.** Construction of the joint probability *ξt*(*i*, *j*)

Recall that *γt*(*j*) is the joint probability that the stochastic process is in state *sj* at time point *t* with observation sequence *O* = {*o*1, *o*2,…, *oT*}, specified by previous equation 3.1.

According to total probability rule (Nguyen L. , Mathematical Approaches to User Modeling, 2015, p. 101), it is easy to infer that *γt* is sum of *ξt* over all states with , as seen in following equation 4.2.6.

|  |  |
| --- | --- |
|  | (4.2.6) |

Deriving from equations 4.2.5 and 4.2.6, we have:

By extending equation 4.2.4, we receive equation 4.2.7 for specifying HMM parameter estimate given current parameter Δ = (*aij*, *bi*(*k*), *πi*) in detailed.

|  |  |
| --- | --- |
|  | (4.2.7) |

The equation 4.2.7 and its proof are found in (Ramage, 2007, pp. 9-12). Followings are interpretations relevant to the joint probabilities *ξt*(*i*, *j*) and *γt*(*j*) with observation sequence *O*.

* The sum expresses expected number of transitions from state *si* to state *sj* (Rabiner, 1989, p. 265).
* The double sum expresses expected number of transitions from state *si* (Rabiner, 1989, p. 265).
* The sum expresses expected number of times in state *sj* and in observation *φk* (Rabiner, 1989, p. 265).
* The sum expresses expected number of times in state *sj* (Rabiner, 1989, p. 265).

Followings are interpretations of the parameter estimate :

* The transition estimate is expected frequency of transitions from state *si* to state *sj*.
* The observation estimate is expected frequency of times in state *sj* and in observation *φk*.
* The initial estimate is (normalized) expected frequency of state *sj* at the first time point (*t=*1).

It is easy to infer that the parameter estimate is based on joint probabilities *ξt*(*i*, *j*) and *γt*(*j*) which, in turn, are based on current parameter Δ = (*aij*, *bj*(*k*), *πj*). The EM conditional expectation is determined by joint probabilities *ξt*(*i*, *j*) and *γt*(*j*); so, the main task of E-step in EM algorithm is essentially to calculate the joint probabilities *ξt*(*i*, *j*) and *γt*(*j*) according to equations 4.2.5 and 3.1. The EM conditional expectation gets maximal at estimate and so, the main task of M-step in EM algorithm is essentially to calculate according to equation 4.2.7. The EM algorithm is interpreted in HMM learning problem, as shown in table 4.2.1.

|  |
| --- |
| Starting with initial value for Δ, each iteration in EM algorithm has two steps:   1. *E-step*: Calculating the joint probabilities *ξt*(*i*, *j*) and *γt*(*j*) according to equations 4.2.5 and 3.1 given current parameter Δ = (*aij*, *bj*(*k*), *πj*).   Where forward variable *αt* and backward variable *βt* are calculated by previous recurrence equations 2.2 and 2.5.   1. *M-step*: Calculating the estimate based on the joint probabilities *ξt*(*i*, *j*) and *γt*(*j*) determined at E-step, according to equation 4.2.7.   The estimate becomes the current parameter for next iteration.  EM algorithm stops when it meets the terminating condition, for example, the difference of current parameter Δ and next parameter is insignificant. It is possible to define a custom terminating condition. |

**Table 4.2.1.** EM algorithm for HMM learning problem

The algorithm to solve HMM learning problem shown in table 4.2.1 is known as Baum-Welch algorithm by authors Leonard E. Baum and Lloyd R. Welch (Rabiner, 1989). Please see document “Hidden Markov Models Fundamentals” by author Ramage (Ramage, 2007, pp. 8-13) for more details about HMM learning problem. As aforementioned in previous sub-section 4.1, the essence of EM algorithm applied into HMM learning problem is to determine the estimate .

As seen in table 4.2.1, it is not difficult to run E-step and M-step of EM algorithm but how to determine the terminating condition is considerable problem. It is better to establish a computational terminating criterion instead of applying the general statement “EM algorithm stops when it meets the terminating condition, for example, the difference of current parameter Δ and next parameter is insignificant”. Going back the learning problem that EM algorithm solves, the EM algorithm aims to maximize probability *P*(*O|*Δ) of given observation sequence *O*=(*o*1, *o*2,… , *oT*) so as to find out the estimate . Maximizing the probability *P*(*O|*Δ) is equivalent to maximizing the conditional expectation. So it is easy to infer that EM algorithm stops when probability *P*(*O|*Δ) approaches to maximal value and EM algorithm cannot maximize *P*(*O|*Δ) any more. In other words, the probability *P*(*O|*Δ) is terminating criterion. Calculating criterion *P*(*O|*Δ) is evaluation problem described in section 2. Criterion *P*(*O|*Δ) is determined according to forward-backward procedure; please see tables 2.1 and 2.2 for more details about forward-backward procedure.

|  |
| --- |
| 1. Initialization step: Initializing *α*1(*i*) = *bi*(*o*1)*πi* for all 2. Recurrence step: Calculating all *αt+*1(*j*) for all and according to equation 2.2. 3. Evaluation step: Calculating the probability |

At the end of M-step, the next criterion *P*(*O|*) that is calculated based on the next parameter (also estimate) is compared with the current criterion *P*(*O|*Δ) that is calculated in the previous time. If these two criteria are the same or there is no significantly difference between them then, EM algorithm stops. This implies EM algorithm cannot maximize *P*(*O|*Δ) any more. However, calculating the next criterion *P*(*O|*) according to forward-backward procedure causes EM algorithm to run slowly. This drawback is overcome by following comment and improvement. The essence of forward-backward procedure is to determine forward variables *αt* while EM algorithm must calculate all forward variables and backward variables in its learning process (E-step). Thus, the evaluation of terminating condition is accelerated by executing forward-backward procedure inside the E-step of EM algorithm. In other words, when EM algorithm results out forward variables in E-step, the forward-backward procedure takes advantages of such forward variables so as to determine criterion *P*(*O|*Δ) the at the same time. As a result, the speed of EM algorithm does not decrease. However, there is always a redundant iteration; suppose that the terminating criterion approaches to maximal value at the end of the *rth* iteration but the EM algorithm only stops at the E-step of the (*r*+1)*th* iteration when it really evaluates the terminating criterion. In general, the terminating criterion *P*(*O|*Δ) is calculated based on the current parameter Δ at E-step instead of the estimate at M-step. Table 4.2.2 shows the proposed implementation of EM algorithm with terminating criterion *P*(*O|*Δ). Pseudo-code like programming language C is used to describe the implementation of EM algorithm. Note, variables are marked as *italic words*, programming language keywords (*while*, *for*, *if*, [], *==*, !=, &&, //, etc.) are marked blue and comments are marked gray. For example, notation [] denotes array index operation; concretely, *α*[*t*][*i*] denotes forward variable *αt*(*i*) at time point *t* with regard to state *si*.

|  |
| --- |
| Input:  HMM with current parameter Δ = {*aij*, *πj*, *bjk*}  Observation sequence *O* = {*o*1, *o*2,…, *oT*}  Output:  HMM with optimized parameter Δ = {*aij*, *πj*, *bjk*}  Allocating memory for two matrices *α* and *β* representing forward variables and backward variables.  *previous\_criterion* = –1  *current\_criterion* = –1  *iteration* = 0  //Pre-defined number *MAX\_ITERATION* is used to prevent from infinite loop.  *MAX\_ITERATION* = 10000  While (*iteration* < *MAX\_ITERATION*)    //Calculating forward variables and backward variables  For *t* = 1 to *T*  For *i* = 1 to *n*  Calculating forward variables *α*[*t*][*i*] and backward variables *β*[*T*–*t*+1][*i*] based on observation sequence *O* according to equations 2.2 and 2.5.  End for *i*  End for *t*    //Calculating terminating criterion *current\_criterion* = *P*(*O|*Δ)  *current\_criterion* = 0  For *i* = 1 to *n*  *current\_criterion* = *current\_criterion* + *α*[*T*][*i*]  End for i  //Terminating condition  If *previous\_criterion* >= 0 && *previous\_criterion* == *current\_criterion* then  break //breaking out the loop, the algorithm stops  Else  *previous\_criterion* = *current\_criterion*  End if    //Updating transition probability matrix  For *i* = 1 to *n*  *denominator* = 0  Allocating *numerators* as a 1-dimension array including *n* zero elements.  For *t* = 2 to *T*  For *k* = 1 to *n*  *ξ* = *α*[*t*–1][*i*] \* *aik* \* *bk*(*ot*) \* *β*[*t*][*k*]  *numerators*[*k*] = *numerators*[*k*] + *ξ*  *denominator* = *denominator* + *ξ*  End for *k*  End for *t*    If *denominator* != 0 then  For *j* = 1 to *n*  *aij* = *numerators*[*j*] / *denominator*  End for *j*  End if    End for *i*    //Updating initial probability matrix  Allocating *g* as a 1-dimension array including *n* elements.  *sum* = 0  For *j* = 1 to *n*  *g*[*j*] = *α*[1][*j*] \* *β*[1][*j*]  *sum* = *sum* + *g*[j]  End for *j*    If *sum* != 0 then  For *j* = 1 to *n*  *πj* = *g*[*j*] / *sum*  End for *j*  End if  //Updating observation probability distribution  For *j* = 1 to *n*  Allocating *γ* as a 1-dimension array including *T* elements.  *denominator* = 0  For *t* = 1 to *T*  *γ*[*t*] = *α*[*t*][*j*] \* *β*[*t*][*j*]  *denominator* = *denominator* + *γ*[*t*]  End for *t*    Let *m* be the columns of observation distribution matrix *B*.  For *k* = 1 to *m*  *numerator* = 0  For *t* = 1 to *T*  If *ot* == *k* then  *numerator* = *numerator* + *γ*[*t*]  End if  End for *t*    *bjk* = *numerator* / *denominator*  End for *k*  End for *j*  *iteration* = *iteration* + 1  End while |

**Table 4.2.2.** Proposed implementation of EM algorithm for learning HMM with terminating criterion *P*(*O|*Δ)

According to table 4.2.2, the number of iterations is limited by a pre-defined maximum number, which aims to solve a so-called infinite loop optimization. Although it is proved that EM algorithm always converges, maybe there are two different estimates and at the final convergence. This situation causes EM algorithm to alternate between and in infinite loop. Therefore, the final estimate or is totally determined but the EM algorithm does not stop. This is the reason that the number of iterations is limited by a pre-defined maximum number.

Going back given weather HMM ∆ whose parameters *A*, *B*,and∏ are specified in tables 1.1, 1.2, and 1.3, suppose observation sequence is *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3*=φ*2*=dryish*}, the EM algorithm and its implementation described in tables 4.2.1 and 4.2.2 are applied into calculating the parameter estimate which is the ultimate solution of the learning problem, as below.

**At the first iteration** (*r=*1) we have:

Within the E-step of the first iteration (*r=*1), the terminating criterion *P*(*O|*Δ) is calculated according to forward-backward procedure (see table 2.1) as follows:

Within the E-step of the first iteration (*r=*1), the joint probabilities *ξt*(*i*,*j*) and *γt*(*j*) are calculated based on equations 4.2.5 and 3.1 as follows:

Within the M-step of the first iteration (*r=*1), the estimate is calculated based on joint probabilities *ξt*(*i*,*j*) and *γt*(*j*) determined at E-step.

**At the second iteration** (*r=*2), the current parameter Δ = (*aij*, *bj*(*k*), *πj*) is received values from the previous estimate , as seen in table 4.2.3.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  | |
|  | | | |
| Terminating criterion *P*(*O|*Δ) = 0.013 | | | |

**Table 4.2.3.** HMM parameters resulted from the first iteration of EM algorithm

We have:

Similarly, within the E-step of the second iteration (*r=*2), the terminating criterion *P*(*O|*Δ) is calculated according to forward-backward procedure (see table 2.1) as follows:

Within the E-step of the second iteration (*r=*2), the joint probabilities *ξt*(*i*,*j*) and *γt*(*j*) are calculated based on equations 4.2.5 and 3.1 as follows:

Similarly, within the M-step of the second iteration (*r=*2), the estimate is re-calculated based on joint probabilities *ξt*(*i*,*j*) and *γt*(*j*) determined at E-step.

Table 4.2.4 summarizes HMM parameters resulted from the first iteration and the second iteration of EM algorithm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration | HMM parameters | | | |
| 1st |  |  |  | |
|  |  |  | |
|  |  |  | |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  | |
|  | | | |
| Terminating criterion *P*(*O|*Δ) = 0.013 | | | |
|  | | | | |
| 2nd |  |  |  | |
|  |  |  | |
|  |  |  | |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  | |
|  | | | |
| Terminating criterion *P*(*O|*Δ) = 0.0776 | | | |

**Table 4.2.4.** HMM parameters resulted from the first iteration and the second iteration of EM algorithm

As seen in table 4.2.4, the EM algorithm does not converge yet when it produces two different terminating criteria (0.013 and 0.0776) at the first iteration and the second iteration. It is necessary to run more iterations so as to gain the most optimal estimate. Within this example, the EM algorithm converges absolutely after 10 iterations when the criterion *P*(*O*|Δ) approaches to the same value 1 at the 9th and 10th iterations. Table 4.2.5 shows HMM parameter estimates along with terminating criterion *P*(*O|*Δ) at the 1st, 2nd, 9th, and 10th iterations of EM algorithm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration | HMM parameters | | | |
| 1st |  |  |  | |
|  |  |  | |
|  |  |  | |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  | |
|  | | | |
| Terminating criterion *P*(*O|*Δ) = 0.013 | | | |
| 2nd |  |  |  | |
|  |  |  | |
|  |  |  | |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  | |
|  | | | |
| Terminating criterion *P*(*O|*Δ) = 0.0776 | | | |
| 9th |  |  |  | |
|  |  |  | |
|  |  |  | |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  | |
|  | | | |
| Terminating criterion *P*(*O|*Δ) = 1 | | | |
| 10th |  |  |  | |
|  |  |  | |
|  |  |  | |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  | |
|  | | | |
| Terminating criterion *P*(*O|*Δ) = 1 | | | |

**Table 4.2.5.** HMM parameters along with terminating criteria after 10 iterations of EM algorithm

As a result, the learned parameters *A*, *B*, and ∏ are shown in table 4.2.6:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Weather current day  (Time point *t*) | | |
| *sunny* | *cloudy* | *rainy* |
| Weather previous day  (Time point *t* –1) | *sunny* | *a*11=0 | *a*12=1 | *a*13=0 |
| *cloudy* | *a*21=0 | *a*22=1 | *a*23=0 |
| *rainy* | *a*31=1 | *a*32=0 | *a*33=0 |

|  |  |  |
| --- | --- | --- |
| *sunny* | *cloudy* | *rainy* |
| *π*1=0 | *π*2=0 | *π*3=1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Humidity | | | |
| *dry* | *dryish* | *damp* | *soggy* |
| Weather | *sunny* | *b*11=1 | *b*12=0 | *b*13=0 | *b*14=0 |
| *cloudy* | *b*21=0 | *b*22=1 | *b*23=0 | *b*24=0 |
| *rainy* | *b*31=0 | *b*32=0 | *b*33=0 | *b*34=1 |

**Table 4.2.6.** HMM parameters of weather example learned from EM algorithm

Such learned parameters are more appropriate to the training observation sequence *O* = {*o*1=*φ*4=*soggy*, *o*2=*φ*1=*dry*, *o*3*=φ*2*=dryish*} than the original ones shown in tables 1.1, 1.2, and 1.3 when the terminating criterion *P*(*O|*Δ) corresponding to its optimal state sequence is 1.

Now three main problems of HMM are described; please see an excellent document “A tutorial on hidden Markov models and selected applications in speech recognition” written by author Rabiner (Rabiner, 1989) for advanced details about HMM.

# Conclusions

In general, there are three main problems of HMM such as evaluation problem, uncovering problem, and learning problem. For evaluation problem and uncovering problem, researchers should pay attention to forward variable and backward variable. Most computational operations are relevant to them. They reflect unique aspect of HMM. The Viterbi algorithm is very effective to solve the uncovering problem. The Baum-Welch algorithm is often used to solve the learning problem. It is easier to explain Baum-Welch algorithm by combination of EM algorithm and optimization theory, in which the Lagrangian function is maximized so as to find out optimal parameters of EM algorithms when such parameters are also learned parameters of HMM.

Observations of normal HMM described in this report are quantified by discrete probability distribution which is observation probability matrix *B*. In the most general case, observation is represented by continuous variable and matrix *B* is replaced by probability density function. At that time the normal HMM becomes continuously observational HMM. Readers are recommended to research continuously observational HMM, an enhanced variant of normal HMM.

# References

Borman, S. (2004). *The Expectation Maximization Algorithm - A short tutorial.* University of Notre Dame, Department of Electrical Engineering. South Bend, Indiana: Sean Borman's Home Page.

Boyd, S., & Vandenberghe, L. (2009). *Convex Optimization.* New York, NY, USA: Cambridge University Press.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. (M. Stone, Ed.) *Journal of the Royal Statistical Society, Series B (Methodological), 39*(1), 1-38.

Fosler-Lussier, E. (1998). *Markov Models and Hidden Markov Models: A Brief Tutorial.* Technical Report TR-98-041, International Computer Science Institute, USA.

Jia, Y.-B. (2013). *Lagrange Multipliers.* Lecture notes on course “Problem Solving Techniques for Applied Computer Science”, Iowa State University of Science and Technology, USA.

Nguyen, L. (2015). *Mathematical Approaches to User Modeling.* London Mathematical Society. Journals Consortium. Retrieved from https://www.researchgate.net/publication/282691564\_Mathematical\_Approaches\_to\_User\_Modeling

Nguyen, L. (2016, June 17). Longest-path Algorithm to Solve Uncovering Problem of Hidden Markov Model. (L. Nguyen, & M. A. MELLAL, Eds.) *Special Issue "Some Novel Algorithms for Global Optimization and Relevant Subjects", Applied and Computational Mathematics (ACM), 6*(4-1), 39-47. doi:10.11648/j.acm.s.2017060401.13

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. (J. H. Trussell, Ed.) *Proceedings of the IEEE, 77*(2), 257-286.

Ramage, D. (2007). *Hidden Markov Models Fundamentals.* Lecture Notes, Stanford University, USA.

Schmolze, J. G. (2001). *An Introduction to Hidden Markov Models.* Lecture Notes on course "COMP 232-Knowledge Based Systems", Department of Computer Science, Tufts University.

Sean, B. (2009). *The Expectation Maximization Algorithm - A short tutorial.* University of Notre Dame, Indiana, Department of Electrical Engineering. Sean Borman's Homepage.

Wikipedia. (2014, August 4). *Karush–Kuhn–Tucker conditions*. (Wikimedia Foundation) Retrieved November 16, 2014, from Wikipedia website: http://en.wikipedia.org/wiki/Karush–Kuhn–Tucker\_conditions