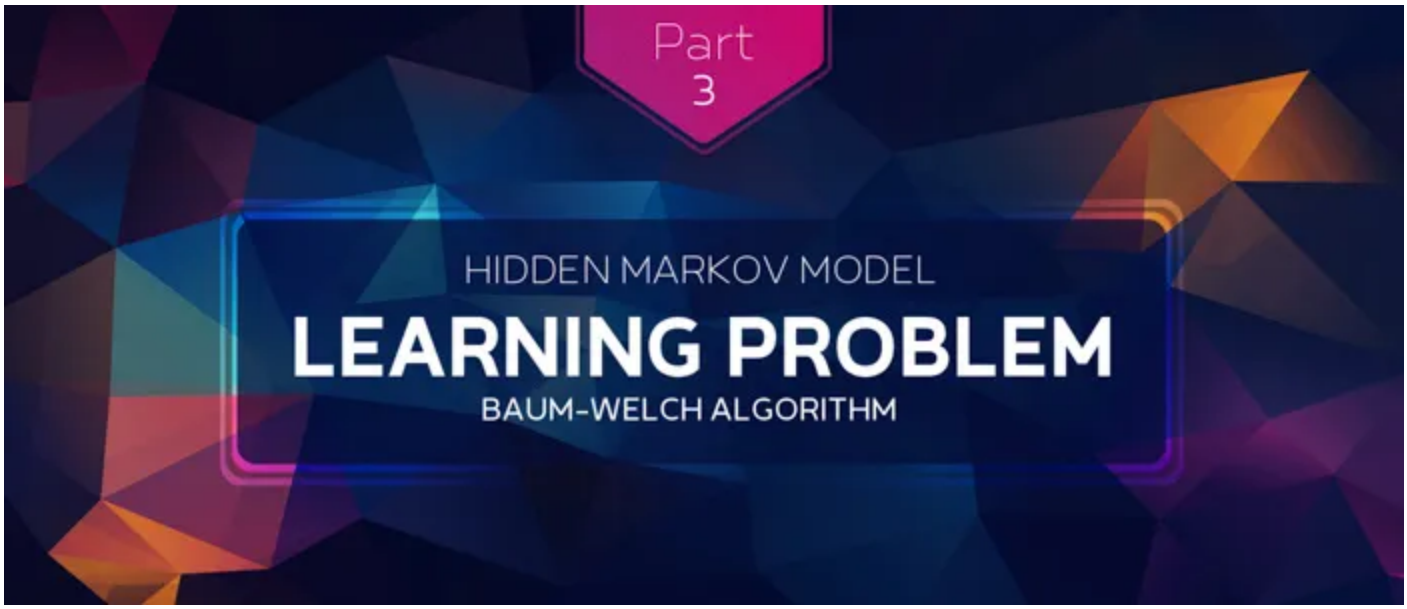
February 20, 2019 By Abhisek Jana — 20 Comments (Edit)

Derivation and implementation of Baum Welch Algorithm for Hidden Markov Model



The most important and complex part of Hidden Markov Model is the `Learning Problem`. Even though it can be used as Unsupervised way, the more common approach is to use Supervised learning just for defining number of hidden states. In this Derivation and implementation of Baum Welch Algorithm for Hidden Markov Model article we will go through step by step derivation process of the `Baum Welch Algorithm` (a.k.a `Forward-Backward` Algorithm) and then implement is using both Python and R.

Quick Recap:

This is the 3rd part of the **Introduction to Hidden Markov Model Tutorial**. So far we have gone through the intuition of HMM, derivation and implementation of the Forward and Backward Algorithm. In case you need a refresher please refer the part 2 of the tutorial series.

Forward and Backward Algorithm in Hidden Markov Model

Introduction to Hidden Markov Model article provided basic understanding of the Hidden Markov Model. We also went through the introduction of the three main problems of HMM (Evaluation, Learning and Decoding). In this Understanding Forward and Backward Algorithm in Hidden Markov Model article we will dive deep into the Evaluation Problem. We will go through … Continue reading

A Developer Diary 10

Learning Problem : HMM Training

- The objective of the `Learning Problem` is to estimate for a_{ij} and b_{jk} using the training data.
- The standard algorithm for Hidden Markov Model training is the `Forward-Backward` or `Baum-Welch` Algorithm.
- This algorithm uses a special case of the `Expectation Maximization` (EM) Algorithm.

Example using Maximum Likelihood Estimate:

Now let's try to get an intuition using an example of `Maximum Likelihood Estimate`.Consider training a `Simple Markov Model` where the hidden state is visible.

We we use our example used in the programming section (You should already have it if you have followed part 2) where we had 2 hidden states `[A,B]` and 3 visible states `[1,2,3]`. (Assume in this example the hidden states are also known)

As you see here we have 4 different sets of sequences (each in alternative colors).

3	2	2	1	1	3	1	2	3	2	1	1
B	B	A	A	A	B	B	A	B	B	A	A

Now we will compute the HMM parameters by `Maximum Likelihood Estimation` using the sample data above.

Estimate Initial Probability Distribution

We will initialize π using the probability derived from the above sequences. In the example above, one of the sequence started with `A` and rest all 3 with `B`. We can define,

$\pi_A = 1/3, \pi_B = 2/3$

Estimate Transition Probabilities:

Lets define our `Transition Probability Matrix` first as:

$$\hat{A} = \begin{bmatrix} p(A|A) & p(B|A) \\ p(A|B) & p(B|B) \end{bmatrix}$$

We can calculate the probabilities from the example as (Ignore the final hidden state since there is to state to transition to):

$$\hat{A} = \begin{bmatrix} 2/4 & 2/4 \\ 3/4 & 1/4 \end{bmatrix}$$

Estimate Emission Probabilities:

Same way, following should be our `Emission Probability Matrix`.

$$\hat{B} = \begin{bmatrix} p(1|A) & p(2|A) & p(3|A) \\ p(1|B) & p(2|B) & p(3|B) \end{bmatrix}$$

Here are the calculated probabilities:

$$\hat{B} = \begin{bmatrix} 4/6 & 2/6 & 0/6 \\ 1/6 & 2/6 & 3/6 \end{bmatrix}$$

Baum-Welch Algorithm:

The above maximum likelihood estimate will work only when the sequence of hidden states are known. However thats not the case for us. Hence we need to find another way to estimate the Transition and Emission Matrix.

This algorithm is also known as Forward-Backward or Baum-Welch Algorithm, it's a special case of the Expectation Maximization (EM) algorithm.

High Level Steps of the Algorithm (EM):

Lets first understand what we need in order to get an estimate for the parameters of the HMM. Here are the high level steps:

1. Start with initial probability estimates `[A,B]`. Initially set equal probabilities or define them randomly.
2. Compute expectation of how often each transition/emission has been used. We will estimate latent variables [ξ, γ] (This is common approach for EM Algorithm)
3. Re-estimate the probabilities `[A,B]` based on those estimates (latent variable).
4. Repeat until convergence

How to solve Baum-Welch Algorithm?:

There are two main ways we can solve the Baum-Welch Algorithm.

- **Probabilistic Approach :** HMM is a Generative model, hence we can solve Baum-Welch using Probabilistic Approach.
- **Lagrange Multipliers :** The Learning problem can be defined as a constrained optimization problem, hence it can also be solved using `Lagrange Multipliers`.

The final equation for both A, B will look the same irrespective of any of the above approach since both A,B can be defined using joint and marginal probabilities. Let's look at the formal definition of them :

Estimate for a_{ij}:

$$\hat{a_{ij}} = \frac{\text{expected number of transitions from hidden state i to state j}}{\text{expected number of transition from hidden state i}}$$

Estimate for b_{jk}:

$$\hat{b_{jk}} = \frac{\text{expected number of times in hidden state j and observing v(k)}}{\text{expected number of times in hidden state j}}$$

The above definition is just the generalized view of the Maximum Likelihood Example we went through. Let's use the Probabilistic Approach and find out how we can estimate the parameters `A,B`

Probabilistic Approach:

Derivation of $\hat{a_{ij}}$:

If we know the probability of a given transition from `i` to `j` at time step `t`, then we can sum over all the `T` times to estimate for the numerator in our equation for $\hat{A}$.

By the way $\hat{A}$ is just the matrix representation of $\hat{a_{ij}}$, so don't be confused.

We can define this as the probability of being in state `i` at time `t` and in state `j` at time `t+1`, given the observation sequence and the model.

Mathematically,
$p(s(t) = i, s(t+1) = j | V^T, \theta)$

We already know from the basic probability theory that,

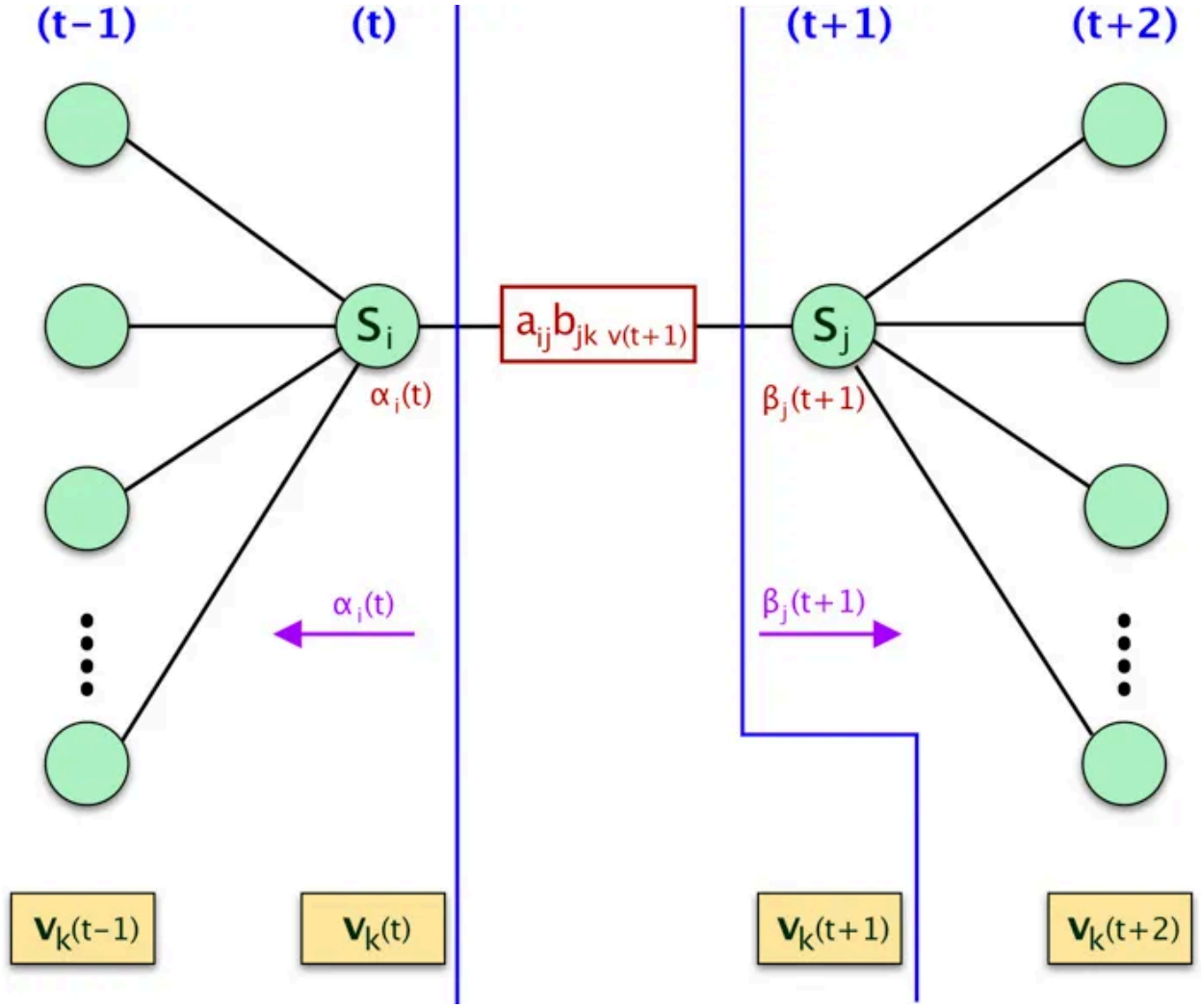
$$p(X,Y|Z) = p(X|Y,Z)p(Y|Z)$$
$$p(X|Y,Z) = \frac{p(X,Y|Z)}{p(Y|Z)}$$

We can now say,

$$p(s(t)=i, s(t+1)=j|V^T,\theta) = \frac{p(s(t)=i, s(t+1)=j, V^T|\theta)}{p(V^T|\theta)}$$

The numerator of the equation can be expressed using `Forward` and `Backward` Probabilities (Refer the diagram below):

$$p(s(t)=i, s(t+1)=j, V^T|\theta) = \alpha_i(t)a_{ij}b_{jk\ v(t+1)}\beta_j(t+1)$$



The denominator $p(V^T|\theta)$ is the probability of the observation sequence V^T by any path given the model θ. It can be expressed as the marginal probability:

$$p(V^T|\theta) = \sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_i(t)a_{ij}b_{jk\ v(t+1)}\beta_j(t+1)$$

We will define ξ as the `latent variable` representing $p(s(t)=i, s(t+1)=j|V^T,\theta)$. We can now define $\xi_{ij}(t)$ as:

$$\xi_{ij}(t) = \frac{\alpha_i(t)a_{ij}b_{jk\ v(t+1)}\beta_j(t+1)}{\sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_i(t)a_{ij}b_{jk\ v(t+1)}\beta_j(t+1)}$$

The $\xi_{ij}(t)$ defined above is only for `one time step`, we need to sum over all `T` to get the total `joint probability` for all the transitions from hidden state `i` to hidden state `j`. This will be our numerator of the equation of $\hat{a_{ij}}$.

For the denominator, we need to get the marginal probability which can be expressed as following,

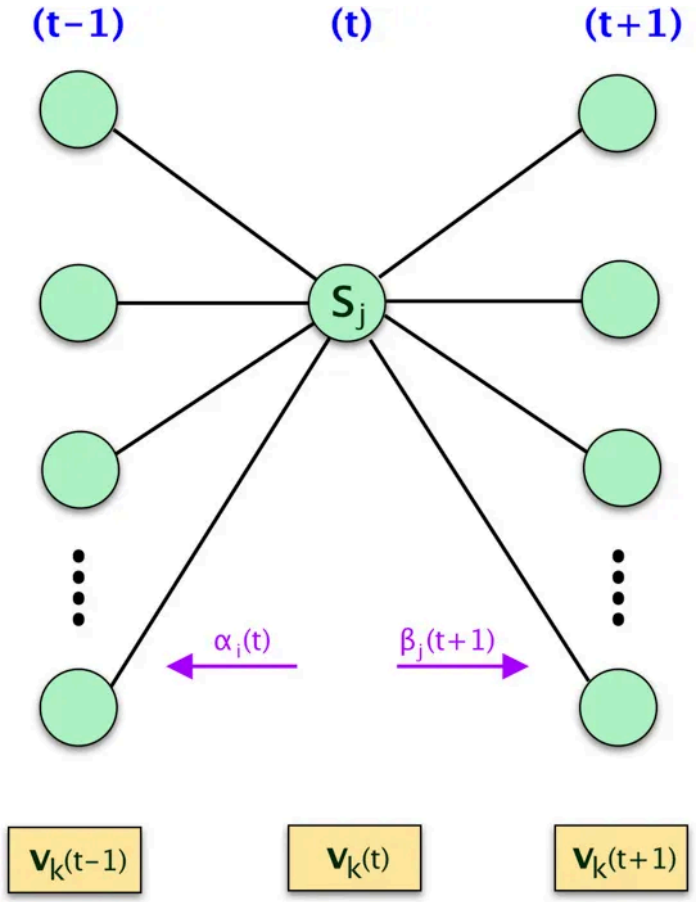
$\sum_{t=1}^{T-1}\sum_{j=1}^{M}\xi_{ij}(t)$
Now we can define $\hat{a_{ij}}$ as,

$$\hat{a_{ij}} = \frac{\sum_{t=1}^{T-1}\xi_{ij}(t)}{\sum_{t=1}^{T-1}\sum_{j=1}^{M}\xi_{ij}(t)} \ldots\ldots\ldots (1)$$

Probabilistic view of the Denominator:

Before we move on estimating `B`, let's understand more on the denominator of $\hat{a_{ij}}$. The denominator is the probability of a state `i` at time `t`, which can be expressed as :

$$\begin{aligned} p(s(t)=i|V^T,\theta) &= \frac{p(s(t)=i, V^T|\theta)}{p(V^T|\theta)} \\ &= \frac{p(v(1)\ldots v(t), s(t)=i|\theta)p(v(t+1)\ldots v(T)|s(t)=i,\theta)}{p(V^T|\theta)} \\ &= \frac{\alpha_i(t)\beta_i(t)}{p(V^T|\theta)} \\ &= \frac{\alpha_i(t)\beta_i(t)}{\sum_{i=1}^{M}\alpha_i(t)\beta_i(t)} = \gamma_i(t) \end{aligned}$$



if we use the above equation to define our estimate for A, it will be,

$$\hat{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma(t)} \ldots\ldots\ldots (2)$$

This is the same equation as (1) we derived earlier.

However, since
$\gamma_i(t) = \sum_{j=1}^{M} \xi_{ij}(t)$

we can just use $\xi_{ij}(t)$ to define the $\hat{a_{ij}}$. This will same some computation.

In summary, in case you see the estimate of a_{ij} with this equation, don't be confused, since both (1) and (2) are identical, even through the representations are different.

Derivation of $\hat{b_{jk}}$:

b_{jk} is the probability of a given symbol v_k from the observations `V` given a hidden state `j`.

We already know the probability of being in state `j` at time `t`.

$\gamma_j(t) = \frac{\alpha_j(t)\beta_j(t)}{\sum_{j=1}^{M} \alpha_j(t)\beta_j(t)}$
We can compute $\hat{b_{jk}}$ using $\gamma_j(t)$,

$\hat{b_{jk}} = \frac{\sum_{t=1}^{T} \gamma_j(t) 1(v(t)=k)}{\sum_{t=1}^{T} \gamma_j(t)}$
where $1(v(t) = k)$ is the `indicator function`.

Final EM Algorithm:

- **initialize** A and B
- **iterate** until convergence
  - **E-Step**
    - $\xi_{ij}(t) = \frac{\alpha_i(t) a_{ij} b_{jk\, v(t+1)} \beta_j(t+1)}{\sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i(t) a_{ij} b_{jk\, v(t+1)} \beta_j(t+1)}$
    - $\gamma_i(t) = \sum_{j=1}^{M} \xi_{ij}(t)$
  - **M-Step**
    - $\hat{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \sum_{j=1}^{M} \xi_{ij}(t)}$
    - $\hat{b_{jk}} = \frac{\sum_{t=1}^{T} \gamma_j(t) 1(v(t)=k)}{\sum_{t=1}^{T} \gamma_j(t)}$
- **return** A,B

Lagrange Multipliers:

We can represent the Learning problem as a `constrained optimization` problem and define it as,

$$\begin{aligned}&\text{Optimize } p(V^T|\theta)\\&\quad\text{where } \theta = \{\pi, A, B\}\\&\text{Subject to } \begin{cases}\sum_{i=1}^{M} \pi_i = 1\\ \sum_{j=1}^{M} a_{ij} = 1, \forall i \in \{1, \ldots, M\}\\ \sum_{k=1}^{M} b_{jk} = 1, \forall j \in \{1, \ldots, M\}\end{cases}\end{aligned}$$

We can then solve this using `Lagrange Multipliers` and by taking the derivatives. We are not going to through the details of that derivation here, however if you are interested let me know I can expand this section if needed.

Code :

R-Script:

Here is the implementation of the algorithm.

- In line# 23-24, we are appending the `T`'th data into the γ since ξ's length is `T-1`
- We are using ξ to derive γ.
- The `indicator` function has been implemented using which in line# 26.

```
BaumWelch = function(v, a, b, initial_distribution,
n.iter = 100){

  for(i in 1:n.iter){
    T = length(v)
    M = nrow(a)
    K=ncol(b)
    alpha = forward(v, a, b, initial_distribution)
    beta = backward(v, a, b)
    xi = array(0, dim=c(M, M, T-1))

    for(t in 1:T-1){
```

```
      denominator = ((alpha[t,] %*% a) * b[,v[t+1]]) %*%
matrix(beta[t+1,])
      for(s in 1:M){
        numerator = alpha[t,s] * a[s,] * b[,v[t+1]] *
beta[t+1,]
        xi[s,,t]=numerator/as.vector(denominator)
      }
    }



    xi.all.t = rowSums(xi, dims = 2)
    a = xi.all.t/rowSums(xi.all.t)

    gamma = apply(xi, c(1, 3), sum)
    gamma = cbind(gamma, colSums(xi[, , T-1]))
    for(l in 1:K){
      b[, l] = rowSums(gamma[, which(v==l)])
    }
    b = b/rowSums(b)

  }
  return(list(a = a, b = b, initial_distribution =
initial_distribution))
}
```

Here is the full code.

```
forward = function(v, a, b, initial_distribution){

  T = length(v)
  M = nrow(a)
  alpha = matrix(0, T, M)
```

```
  alpha[1, ] = initial_distribution*b[, v[1]]

  for(t in 2:T){
    tmp = alpha[t-1, ] %*% a
    alpha[t, ] = tmp * b[, v[t]]
  }
  return(alpha)
}

backward = function(v, a, b){
  T = length(v)
  M = nrow(a)
  beta = matrix(1, T, M)

  for(t in (T-1):1){
    tmp = as.matrix(beta[t+1, ] * b[, v[t+1]])
    beta[t, ] = t(a %*% tmp)
  }
  return(beta)
}



BaumWelch = function(v, a, b, initial_distribution,
n.iter = 100){

  for(i in 1:n.iter){
    T = length(v)
    M = nrow(a)
    K=ncol(b)
    alpha = forward(v, a, b, initial_distribution)
    beta = backward(v, a, b)
    xi = array(0, dim=c(M, M, T-1))
```

```
    for(t in 1:T-1){
      denominator = ((alpha[t,] %*% a) * b[,v[t+1]]) %*%
matrix(beta[t+1,])
      for(s in 1:M){
        numerator = alpha[t,s] * a[s,] * b[,v[t+1]] *
beta[t+1,]
        xi[s,,t]=numerator/as.vector(denominator)
      }
    }



    xi.all.t = rowSums(xi, dims = 2)
    a = xi.all.t/rowSums(xi.all.t)

    gamma = apply(xi, c(1, 3), sum)
    gamma = cbind(gamma, colSums(xi[, , T-1]))
    for(l in 1:K){
      b[, l] = rowSums(gamma[, which(v==l)])
    }
    b = b/rowSums(b)

  }
  return(list(a = a, b = b, initial_distribution =
initial_distribution))
}

data = read.csv("data_r.csv")

M=2; K=3
A = matrix(1, M, M)
A = A/rowSums(A)
B = matrix(1:6, M, K)
B = B/rowSums(B)
```

```
initial_distribution = c(1/2, 1/2)

(myout = BaumWelch(data$Visible, A, B,
initial_distribution, n.iter = 100))
```

Output:

```
$a
          [,1]      [,2]
[1,] 0.5381634 0.4618366
[2,] 0.4866444 0.5133556

$b
          [,1]      [,2]      [,3]
[1,] 0.1627751 0.2625807 0.5746441
[2,] 0.2514996 0.2778097 0.4706907

$initial_distribution
[1] 0.5 0.5
```

Validate Result:

Let's validate our result with the HMM R package.

```
library(HMM)
hmm =initHMM(c("A", "B"), c(1, 2, 3),
             startProbs = initial_distribution,
             transProbs = A, emissionProbs = B)

true.out = baumWelch(hmm, data$Visible,
maxIterations=100, pseudoCount=0)
true.out$hmm
```

Here is the output, which is exactly same as our output.

```
$States
[1] "A" "B"

$Symbols
[1] 1 2 3

$startProbs
  A   B
0.5 0.5

$transProbs
    to
from         A         B
   A 0.5381634 0.4618366
   B 0.4866444 0.5133556

$emissionProbs
      symbols
states         1         2         3
     A 0.1627751 0.2625807 0.5746441
     B 0.2514996 0.2778097 0.4706907
```

Python:

Here is the python code for the Baum Welch algorithm, the logic is same as we have used in R.

```
def baum_welch(V, a, b, initial_distribution,
n_iter=100):
    M = a.shape[0]
    T = len(V)

    for n in range(n_iter):
        alpha = forward(V, a, b, initial_distribution)
```

```
        beta = backward(V, a, b)

        xi = np.zeros((M, M, T - 1))
        for t in range(T - 1):
            denominator = np.dot(np.dot(alpha[t, :].T, a)
* b[:, V[t + 1]].T, beta[t + 1, :])
            for i in range(M):
                numerator = alpha[t, i] * a[i, :] * b[:,
V[t + 1]].T * beta[t + 1, :].T
                xi[i, :, t] = numerator / denominator

        gamma = np.sum(xi, axis=1)
        a = np.sum(xi, 2) / np.sum(gamma,
axis=1).reshape((-1, 1))

        # Add additional T'th element in gamma
        gamma = np.hstack((gamma, np.sum(xi[:, :, T - 2],
axis=0).reshape((-1, 1))))

        K = b.shape[1]
        denominator = np.sum(gamma, axis=1)
        for l in range(K):
            b[:, l] = np.sum(gamma[:, V == l], axis=1)

        b = np.divide(b, denominator.reshape((-1, 1)))

    return {"a":a, "b":b}
```

Here is the full code:

```
import pandas as pd
import numpy as np
```

```
def forward(V, a, b, initial_distribution):
    alpha = np.zeros((V.shape[0], a.shape[0]))
    alpha[0, :] = initial_distribution * b[:, V[0]]

    for t in range(1, V.shape[0]):
        for j in range(a.shape[0]):
            # Matrix Computation Steps
            #                  ((1x2) . (1x2))      *
(1)
            #                        (1)            *
(1)
            alpha[t, j] = alpha[t - 1].dot(a[:, j]) *
b[j, V[t]]

    return alpha


def backward(V, a, b):
    beta = np.zeros((V.shape[0], a.shape[0]))

    # setting beta(T) = 1
    beta[V.shape[0] - 1] = np.ones((a.shape[0]))

    # Loop in backward way from T-1 to
    # Due to python indexing the actual loop will be T-2
to 0
    for t in range(V.shape[0] - 2, -1, -1):
        for j in range(a.shape[0]):
            beta[t, j] = (beta[t + 1] * b[:, V[t +
1]]).dot(a[j, :])

    return beta
```

```
def baum_welch(V, a, b, initial_distribution,
n_iter=100):
    M = a.shape[0]
    T = len(V)

    for n in range(n_iter):
        alpha = forward(V, a, b, initial_distribution)
        beta = backward(V, a, b)

        xi = np.zeros((M, M, T - 1))
        for t in range(T - 1):
            denominator = np.dot(np.dot(alpha[t, :].T, a)
* b[:, V[t + 1]].T, beta[t + 1, :])
            for i in range(M):
                numerator = alpha[t, i] * a[i, :] * b[:,
V[t + 1]].T * beta[t + 1, :].T
                xi[i, :, t] = numerator / denominator

        gamma = np.sum(xi, axis=1)
        a = np.sum(xi, 2) / np.sum(gamma,
axis=1).reshape((-1, 1))

        # Add additional T'th element in gamma
        gamma = np.hstack((gamma, np.sum(xi[:, :, T - 2],
axis=0).reshape((-1, 1))))

        K = b.shape[1]
        denominator = np.sum(gamma, axis=1)
        for l in range(K):
            b[:, l] = np.sum(gamma[:, V == l], axis=1)

        b = np.divide(b, denominator.reshape((-1, 1)))
```

```
    return {"a":a, "b":b}


data = pd.read_csv('data_python.csv')

V = data['Visible'].values

# Transition Probabilities
a = np.ones((2, 2))
a = a / np.sum(a, axis=1)

# Emission Probabilities
b = np.array(((1, 3, 5), (2, 4, 6)))
b = b / np.sum(b, axis=1).reshape((-1, 1))

# Equal Probabilities for the initial distribution
initial_distribution = np.array((0.5, 0.5))

print(baum_welch(V, a, b, initial_distribution,
n_iter=100))
```

Output:

Here is the output of our code. Its the same as previous one, however the precision is different.

```
{
'a': array([[0.53816345, 0.46183655],
       [0.48664443, 0.51335557]]),

'b': array([[0.16277513, 0.26258073, 0.57464414],
       [0.2514996 , 0.27780971, 0.47069069]])
}
```

Conclusion:

We went through the details of the Learning Algorithm of HMM here. I hope that this article helped you to understand the concept.

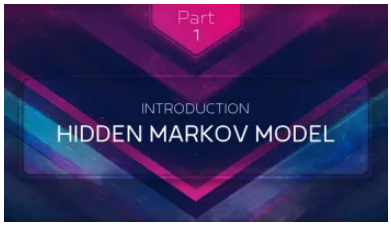
Click on the link to get the code:

Code

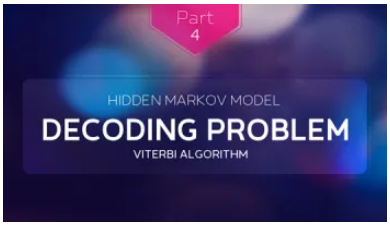
Also, here are the list of all the articles in this series:

1. Introduction to Hidden Markov Model
2. Forward and Backward Algorithm in Hidden Markov Model
3. Derivation and implementation of Baum Welch Algorithm for Hidden Markov Model
4. Implement Viterbi Algorithm in Hidden Markov Model using Python and R

Related



Introduction to Hidden Markov Model
In "Machine Learning"



Implement Viterbi Algorithm in Hidden Markov Model using Python and R
In "Machine Learning"

Forward and Backward Algorithm in Hidden Markov Model
In "Machine Learning"

Filed Under: Machine Learning | Tagged With: Algorithm, Baum-Welch, Forward Backward, Hidden Markov Model, HMM, lagrange multiplier, Machine Learning, Python, R

Comments

alex says
April 26, 2020 at 10:04 am

(Edit)

thanks for the illustration of HMM.
After reading the details of above logic, I found that when training the HMM using Baum Welch Algorithm, the true state column in training data was not used. then how can the training process learn the information from the true state?

So, in prediction of testing data, how can we know the predicted state is for A or B?

correct me if i make any mistake. thanks

Reply

Abhisek Jana says
April 26, 2020 at 1:14 pm

(Edit)

Hi Alex,

This is very good question actually !

We are assuming we have two hidden states, this assumption comes from the hidden state data `[data$Hidden]`. The `data$Visible` is used during the training process to determine the probabilities. The next tutorial explains how to use the probabilities and visible data to predict the hidden state A or B.

https://www.adeveloperdiary.com/data-science/machine-learning/implement-viterbi-algorithm-in-hidden-markov-model-using-python-and-r/

Once you have have the predicted data, you can use that to calculate model accuracy.

Let me know whether this helps to answer your question.

Reply

alex says
April 28, 2020 at 11:49 am

(Edit)

yes, i read the post also. So in general for HMM, use Baum Welch to estimate transition and emission matrix from training data. then use viterbi to make state prediction for testing data.

Let me use Y to indicate the true state and X to indicate the observation. i found that none of your function nor the package in R involve the true state when using Baum Welch. is it the characteristic of HMM that true state (Y) is not used in training?

and do you know what package i should use in python/R that support multiple continuous/discrete observations (X matrix)?

thanks!

Reply

Abhisek Jana says

April 28, 2020 at 12:25 pm

(Edit)

You are right, the hidden state (Y) is not used in training like other supervised algorithms, hence HMM in general is an Unsupervised* Algorithm. I have denoted (*) as we are defining the # of hidden state in the matrix, hence its used partly.

You can use the `mhsmm` package in R, when you have multiple observations. Here is the link:

https://cran.r-project.org/web/packages/mhsmm/index.html

Reply

alex says

April 29, 2020 at 12:07 pm

(Edit)

thanks a lot! I think i am much clear about HMM now.

i just have a look on the mhsmm package. below is a simple code from https://cran.r-project.org/web/packages/mhsmm/mhsmm.pdf

in page 4

```
J<-2
initial <- rep(1/J,J)
P <- matrix(c(.3,.5,.7,.5),nrow=J)
b <-
list(mu=list(c(-3,0),c(1,2)),sigma=list(diag(2),matrix(c(4,2,2,3),
ncol=2)))
model <- hmmspec(init=initial, trans=P,
parms.emission=b,dens.emission=dmvnorm.hsmm)
```

if i understand correctly, there is 2 states and also 2 observations in X matrix. I mean there are 2 features/columns in matrix X, say i call it column 1 and column 2. For emission matrix b, there are two sets of multinormal distribution. the first multinormal distribution with mean c(-3,0) is for column 1 while c(1,2) is for column 2.

For c(-3,0), why there is two means inside? i guess it is because of two states. if yes, can i just use the sample mean of column 1 with respect to two states as the initial guess? and similar for sigma.

is my understanding above correct?

thanks!

Reply

Sara As says
May 27, 2020 at 4:54 am

(Edit)

Hello, Thank you for the great article.
But for HMMs with more states, what happens is that the probabilities of the matrix elements get close to zero.
I tried to compute the logarithm of the transition and emission matrix. but I still get the error.
Can I ask how exactly we should solve this problem?

Thanks

Reply

Rajni says
September 30, 2020 at 11:06 am

(Edit)

Try to readjust the values of alpha between 0 and 1 (look at rabiner Scaling section)

Reply

Lam says
July 27, 2020 at 5:56 am

(Edit)

Hi this series of tutorial is very helpful since u give a numerical example to help me check that I truly understand the concept
here some more details which helps us more truly understand the concept behind
http://www.cs.cmu.edu/~10715-f18/lectures/lecture2-crf.pdf
I also have problem with maximum entropy Markov model with numerical example which is so rare in internet if u can do example again it helps people a lot

Reply

Abhisek Jana says
July 27, 2020 at 12:42 pm

(Edit)

Hi Lam,
Thanks a lot for your feedback!

The cmu ppt on hmm is the nicest one I have seen so far. Really appreciate for sharing this.

Thanks,
Abhisek Jana

Reply

Lam says
October 11, 2020 at 4:50 am

(Edit)

LoL U really reply my comment
well I draft notes and prepare for Tutorial this week and I find that

“In line# 23-24, we are appending the T‘th data into the γ since ξ’s length is T-1”

This part is wired since it just copy the end element of an array because in estimation of b matrix
its T limit for the summon notation
are u sure its just that simple

at the end Million thanks to your tutorial it helps lots in my NLP study

Reply

Debabrot Bhuyan says
July 28, 2020 at 7:42 am

(Edit)

Hi Abhishek,
this tutorial is amazing. Thanks for adding the python implementation as

well.

Reply

Rich Poole says

August 22, 2020 at 4:50 pm

(Edit)

Hi – this is great explanation. Appreciate the thorough presentation of concepts.

FYI, I did notice that the BaumWelch function above does not yield the same output as shown above. It doesn't match the output of hmm() any longer. Any ideas on what changed? I may have a different version of R running that may have changed the behavior of some of the code you originally wrote. I copied both from the GitHub and from this page and same results.

Thanks, Rich

Reply

Rich Poole says

August 22, 2020 at 5:31 pm

(Edit)

Never mind – I found the error – on my side. Again, thanks for the wonderful writeup.

Reply

paul says
August 30, 2020 at 2:37 am

(Edit)

great article！ I learned a lot. thank you very much.
Does you run python codes with n_iter=100：
{
'a': array([[0.53816345, 0.46183655],
[0.48664443, 0.51335557]]),

'b': array([[0.16277513, 0.26258073, 0.57464414],
[0.2514996 , 0.27780971, 0.47069069]])
}

my result is :
{'a': array([[0. , 1.],
[0.25, 0.75]]), 'b': array([[2.26746249e-12, 2.81018639e-40,
1.00000000e+00],
[5.55555556e-01, 4.44444444e-01, 5.39459475e-21]])}
I don't know why I get different result. I use python 3.8 environment. And my training dataset is V = [2 1 1 0 0 2 0 1 2 1 0 0]
Could you give me some advice?

Reply

paul says
August 30, 2020 at 9:25 pm

(Edit)

sorry, I got the problem. dataSet is incomplete.

Reply

John says
December 8, 2020 at 2:36 pm

(Edit)

Amazing article, one of the very best out there!

Could you kindly confirm the value of p(B|B) under the section “Estimate Initial Probability Distribution”? I am counting 3 such transitions out of 6 B states.

Kind regards
John

Reply

Batch says
January 3, 2021 at 1:28 pm

(Edit)

Hi,
Thank you for this thorough explanation!
i’m wondering when looping on the 100 iterations in the Baum-welch function, are we doing it (i.e estimating alpha-beta) on the same sequence in V?

is it one single sequence, or should it be multiple sequences (i.e 100 sequences) . i'm asking since the n parameter is not used (unless i'm missing something) , so it seems to run on the same V of size t and not n*t

thanks!

Reply

Bruna says

May 19, 2022 at 8:48 am

(Edit)

I have the same doubt. By running the code I see there is no clear separation between sequences indicated anywhere. It runs an unique array of size 500 (which makes me think either this one observation sequence has 500 steps or the observations were concatenated).

It would be nice to have a clarification on that.

Reply

Jack says

January 24, 2021 at 10:40 am

(Edit)

Why πA=1/3,πB=2/3? Isn't it should be πA=1/4,πB=3/4

Reply

Mike says

January 13, 2022 at 2:39 pm

(Edit)

That's what I'm seeing too.
1 / 4 start with A
3 / 4 start with B

Reply

Leave a Reply

Logged in as Abhisek Jana. Edit your profile. Log out? Required fields are marked *

Comment *

Post Comment