Hidden Markov Model(HMM)

Jai Vrat Singh

It is one of the Probabilistic Graphical models used widely in data science, machine learning , finance and computational biology. There are many potential uses in Finance, especially inferring states of the economy/cycle given the observable proxies (real world data).

To mention briefly, this is about state of world/variable, which is unobserved directly - but you do get observations/symptoms in form of other variables. e.g if you find a records of number ice-creams eaten by a person from AD 1850-1900, can you infer the heat/temperatures of summers those years?

For theory and mathematical details, you may refer to any standard literature such as Bishop

For those who like pen-paper calculations and really verify what is happening under the hood - I will do some selected calculations (forward and backward) on a toy model. These are two core recursive algorithms used in HMM model.

I will be using R package **HMM**

```
library("HMM")
hmm = initHMM(c("A","B"), c("L","R"),
              transProbs=matrix(c(.8,.2,.2,.8),2),
              emissionProbs=matrix(c(.6,.4,.4,.6),2))
print(hmm)
```

```
## $States
## [1] "A" "B"
##
## $Symbols
## [1] "L" "R"
##
## $startProbs
##   A   B
## 0.5 0.5
##
## $transProbs
##     to
## from   A   B
##    A 0.8 0.2
##    B 0.2 0.8
##
## $emissionProbs
##       symbols
## states   L   R
##      A 0.6 0.4
##      B 0.4 0.6
```

Sequence of observations

```
observations = c("L","L","R","R", "L", "L")
```

Calculate forward probablities

In literature they refer is as $\alpha_i(T)$

To computer the forward probabilities. The forward probability for state X up to observation at time k is defined as the probability of observing the sequence of observations e_1,... ,e_k and that the state at time k is X.

That is: $f[X,k] := Prob(E_1 = e_1, ..., E_k = e_k, X_k = X)$

Where $E_1...E_n = e_1...e_n$ is the sequence of observed emissions and X_k is a random variable that represents the state at time k.

```
# Calculate forward probablities
logForwardProbabilities = forward(hmm,observations)
print(exp(logForwardProbabilities))
```

```
##       index
## states   1     2      3        4         5           6
##      A 0.3 0.168 0.0608 0.024448 0.0162048 0.009443328
##      B 0.2 0.088 0.0624 0.037248 0.0138752 0.005736448
```

We can read this as e.g elements[1,1] => which represents $X_1 = 1$ and $E_1 = L$, which is the observation we feed in.

Mathematically, it means $P(X_1 = A, E_1 = L)$.

We can manually calculate(and intepret above result matrix as) it as

This can be calculated as $P(X_1 = A) * P(E_1 = L|X_1 = A) = 0.5 * 0.6 = 0.3$. This is what we see in elements[1,1].

Similarly element[1,2] represents $E_1 = L$, $E_2 = L$ and $X_2 = A$.

$$
\begin{aligned}
element[1,2] &= P(E_1 = L, E_2 = L, X_2 = A) \\
&= P(E_1 = L, E_2 = L, X_1 = A, X_2 = A) + P(E_1 = L, E_2 = L, X_1 = B, X_2 = A) \\
&= P(X_1 = A) * P(E_2 = L|X_1 = A) * P(X_2 = A|X1 = A) * P(E_2 = L|X_2 = A) + P(X_1 = A) * P(E_2 = L|X_1 = \\
&= 0.5 * 0.6 * 0.8 * 0.6 + 0.5 * 0.4 * 0.2 * 0.6 \\
&= 0.168
\end{aligned}
\tag{1}
$$

Please note that we can calculate **total probabilities of all observations** using forward menthod recursively. $P(E_1 = e_1, ..E_k = e_k) = \sum_{i \in States} \alpha_i(k)$

And recursive relation of α or f is $\alpha_j(t) = \sum_{i \in States} \alpha_i(t-1) A_{ij} B_{je_t}$ $\forall j = 1..T$, $t = 1..T$. Here A and B are the transition and emission probability matrices respectively.

A similar algorithm known as **BACKWARD PROCEDURE** can be used to compute $\beta_i(t) = P(E_T = e_T, E_{T-1} = e_{T-1}, ..., E_{t+1} = e_{t+1}, X_t = s_i; A, B)$ where s_i's are the states.

Calculate backward probablities

The backward probability for state X and observation at time k is defined as the probability of observing the sequence of observations e_{k+1}, ... ,e_n under the condition that the state at time k is X.

That is: $b_{X,k} = Prob(E_{k+1} = e_{k+1}, ..., E_n = e_n | X_k = X)$

Where $E_1...E_n = e_1...e_n$ is the sequence of observed emissions and X_k is a random variable that represents the state at time k.

```
logBackwardProbabilities = backward(hmm,observations)
print(exp(logBackwardProbabilities))
```

```
##       index
## states         1        2       3     4    5 6
##      A 0.03147776 0.054016 0.12224 0.304 0.56 1
##      B 0.02868224 0.069376 0.12416 0.208 0.44 1
```

```
#apply(exp(logBackwardProbabilities),2,sum)
```

e.g Let us evaluate b[A,5] manually.

$$
\begin{aligned}
b[A,5] &= P(E_6 = L|X_5 = A) \\
&= P(E_6 = L, X_6 = A|X_5 = A) + P(E_6 = L, X_6 = B|X_5 = A) \\
&= P(X_6 = A|X_5 = A) * P(E_6 = L|X_6 = A) + P(X_6 = B|X_5 = A) * P(E_6 = L|X_6 = B) \\
&= 0.8 * 0.6 + 0.2 * 0.4 \\
&= 0.56
\end{aligned} \tag{2}
$$

So it matches with the values.

But what if we do not know the parameters A and B?

We do not know the transition probabilities - what to do? => We have Baum Wlech Algorithm to help us. It is a variant of Expectation Maximization algorithm, which converges to a local optimum instead of global.

Let us try creating a new model with our known hidden paramateres and try to see if Baum-Welch can infer parameters from the fed in **simulattions**.

We will define 2 states again with transition matrix and correspoding emission probilities.

```
hmm_simul  = initHMM(c("A","B"), c("L","R"),
                     transProbs=matrix(c(0.9,0.1,0.1,.9),2),
                     emissionProbs=rbind(c(.7,.3),c(.4,.6)))
print(hmm_simul)
```

```
## $States
## [1] "A" "B"
##
## $Symbols
## [1] "L" "R"
##
## $startProbs
##   A   B
## 0.5 0.5
##
## $transProbs
##      to
## from   A   B
##    A 0.9 0.1
##    B 0.1 0.9
```

```
##
## $emissionProbs
##       symbols
## states   L   R
##      A 0.7 0.3
##      B 0.4 0.6
```

Let us simulate observations

```
set.seed(144)

obs.length    = 500
actual.transProbs    = rbind(c(0.8,0.2),
                       c(0.5,.5))
actual.emissionProbs = rbind(c(.8,.2),
                       c(.4,.6))
hmm_simul  = initHMM(c("A","B"), c("L","R"),
                  transProbs=actual.transProbs,
                  emissionProbs=actual.emissionProbs)
print(hmm_simul)
```

```
## $States
## [1] "A" "B"
##
## $Symbols
## [1] "L" "R"
##
## $startProbs
##   A   B
## 0.5 0.5
##
## $transProbs
##     to
## from   A   B
##    A 0.8 0.2
##    B 0.5 0.5
##
## $emissionProbs
##       symbols
## states   L   R
##      A 0.8 0.2
##      B 0.4 0.6
```

```
simulated = simHMM(hmm_simul, length = obs.length)
```

Let us try infer parameters, starting with a guess model

There are two ways to do it - Baum-Welch and Viterbi.

Baum Welch

```r
#1. Constricted an hmm
guess_hmm = initHMM(c("A","B"), c("L","R"),
                    transProbs=rbind(c(0.6,0.4),
                                     c(0.3,0.7)),
                    emissionProbs=rbind(c(.6,.4),
                                        c(.2,.8)))
retBW = baumWelch(guess_hmm, observation = simulated$observation, maxIterations=10000, delta=1E-9, pseu

print(retBW$hmm$transProbs)
```

```
##     to
## from         A            B
##    A 0.9280488 7.195124e-02
##    B 1.0000000 4.221411e-27
```

```r
print(retBW$hmm$emissionProbs)
```

```
##       symbols
## states            L         R
##      A 7.300127e-01 0.2699873
##      B 8.509265e-27 1.0000000
```

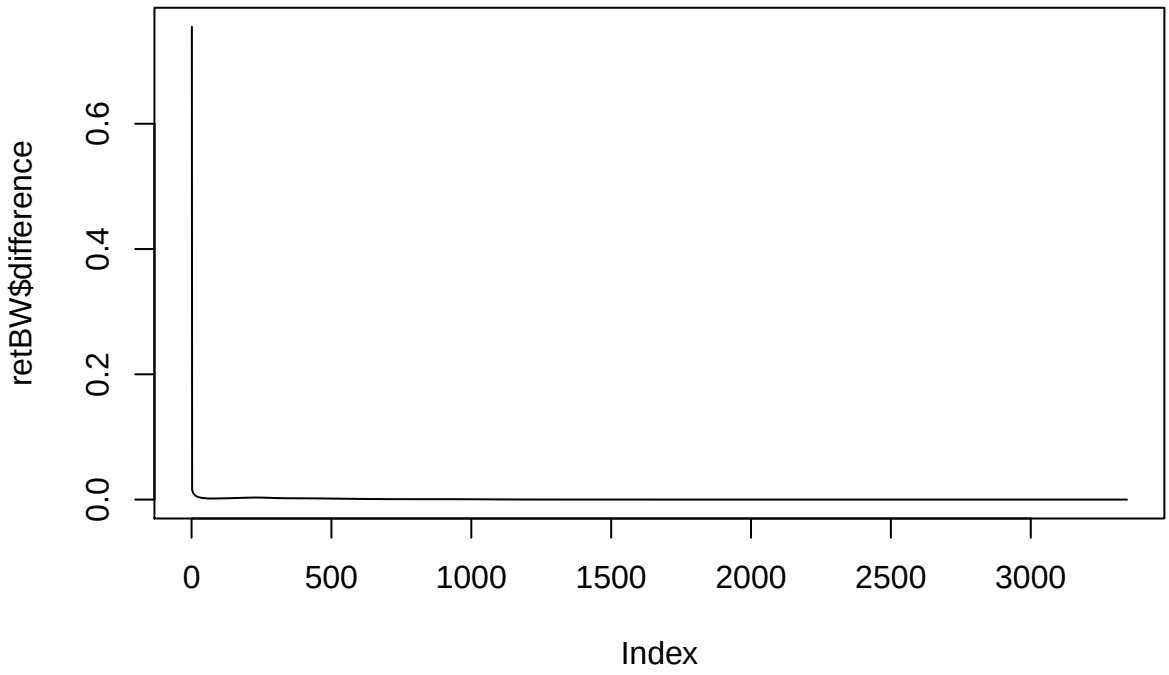
```r
#Diffs from actual
retBW$hmm$transProbs - actual.transProbs
```

```
##     to
## from         A          B
##    A 0.1280488 -0.1280488
##    B 0.5000000 -0.5000000
```

```r
retBW$hmm$emissionProbs - actual.emissionProbs
```

```
##       symbols
## states           L          R
##      A -0.06998727 0.06998727
##      B -0.40000000 0.40000000
```

```r
plot(retBW$difference, type="l")
```



Viterbi Training

```
#
retViterbiTrain  = viterbiTraining(guess_hmm, observation = simulated$observation, maxIterations=10000,
print(retViterbiTrain$hmm$transProbs)
```

```
##     to
## from A B
##    A 1 0
##    B 1 0
```

```
print(retViterbiTrain$hmm$emissionProbs)
```

```
##       symbols
## states         L         R
##      A 0.6813627 0.3186373
##      B 0.0000000 1.0000000
```

```
#Diffs from actual
retViterbiTrain$hmm$transProbs - actual.transProbs
```

```
##     to
## from   A    B
##    A 0.2 -0.2
##    B 0.5 -0.5
```

```
retViterbiTrain$hmm$emissionProbs - actual.emissionProbs
```

```
##       symbols
## states          L         R
##      A -0.1186373 0.1186373
##      B -0.4000000 0.4000000
```

```
plot(retViterbiTrain$difference, type="l")
```


retViterbiTrain$difference
0.0
0.2
0.4
0.6
0.8
1
2
3
4
5
Index