Solution 1

Score observations versus a given model

Given model $\lambda = (A,B,\pi)$ and observation sequence $O=(O_0,O_1,\ldots,O_{T-1})$, find $P(O|\lambda)$

Denote hidden states as

$X = (X_0, X_1, \ldots, X_{T-1})$

Then from definition of B,

$P(O|X,\lambda)=b_{X0}(O_0)\ b_{X1}(O_1)\ \ldots\ b_{XT-1}(O_{T-1})$

And from definition of A and π,

$P(X|\lambda)=\pi_{X0}\ a_{X0,X1}\ a_{X1,X2}\ \ldots\ a_{XT-2,XT-1}$

Solution 1

Elementary conditional probability fact:

$$P(O,X|\lambda) = P(O|X,\lambda)\, P(X|\lambda)$$

Sum over all possible state sequences X,

$$P(O|\lambda) = \Sigma\, P(O,X|\lambda) = \Sigma\, P(O|X,\lambda)\, P(X|\lambda)$$

$$= \Sigma \pi_{X0} b_{X0}(O_0) a_{X0,X1} b_{X1}(O_1) \ldots a_{XT-2,XT-1} b_{XT-1}(O_{T-1})$$

This “works” but way too costly

Requires about $2TN^T$ multiplications

Why?

There should be a better way…

Forward Algorithm

Instead, use ***forward algorithm***

Or "alpha pass"

For t = 0,1,…,T-1 and i=0,1,…,N-1, let

$$\alpha_t(i) = P(O_0,O_1,\ldots,O_t,X_t=q_i|\lambda)$$

Probability of "partial observation" to t, and Markov process is in state q_i at step t

Can be computed recursively, efficiently

Forward Algorithm

Let $\alpha_0(i) = \pi_i b_i(O_0)$ for $i = 0,1,\ldots,N-1$

For $t = 1,2,\ldots,T-1$ and $i=0,1,\ldots,N-1$, let

$$\alpha_t(i) = \left(\Sigma \alpha_{t-1}(j) a_{ji}\right) b_i(O_t)$$

Where the sum is from $j = 0$ to $N-1$

From definition of $\alpha_t(i)$ we see

$$P(O|\lambda) = \Sigma \alpha_{T-1}(i)$$

Where the sum is from $i = 0$ to $N-1$

This requires only N^2T multiplications

Forward Algorithm

Note: The score $P(O|\lambda)$ is dependent on the length of the observation sequence. Consequently, to compare scores for sequences of different length, we can normalize to a per observation score, that is, score = $\Sigma\alpha_{T-1}(i) / T$.

Forward Algorithm

In a nutshell, $\alpha_T(i)$ is the probability that the sequence of observations up to time T correspond to the state being the i^{th} state at time T, given the model λ.

We compute $\alpha_t(i)$ recursively using $\alpha_{t-1}(j)$'s.

Solution 2

Given a model, find hidden states

- Given $\lambda = (A,B,\pi)$ and O, find an optimal state sequence
- Recall that optimal means "maximize expected number of correct states"
- In contrast, DP finds best scoring path

For temp/tree ring example, solved this

- But hopelessly inefficient approach

A better way: ***backward algorithm***

- Or "beta pass"

Backward Algorithm

For $t = 0,1,\ldots,T-1$ and $i = 0,1,\ldots,N-1$, let $\beta_t(i) = P(O_{t+1},O_{t+2},\ldots,O_{T-1}|X_t=q_i,\lambda)$

Probability of partial observation from t to end and Markov process in state q_i at step t

Analogous to the forward algorithm

As with forward algorithm, this can be computed recursively and efficiently

$\beta_t(i) = P(O_{t+1},O_{t+2},\ldots,O_{T-1}|X_t=q_i,\lambda)$

Backward Algorithm

Let $\beta_{T-1}(i) = 1$ for $i = 0,1,\ldots,N-1$

For $t = T-2,T-3, \ldots,1$ and $i = 0,1,\ldots,N-1$, let

$$\beta_t(i) = \Sigma a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

Where the sum is from $j = 0$ to $N-1$

Solution 2

For t = 1,2,…,T-1 and i=0,1,…,N-1 define

$\gamma_t(i) = P(X_t=q_i|O,\lambda)$

(The probability of being in state i at time t)

Most likely state at t is q_i that maximizes $\gamma_t(i)$

$$\widetilde{X}_t = \max_i \gamma_t(i).$$

Note that $\gamma_t(i) = \alpha_t(i)\beta_t(i)/P(O|\lambda)$

And recall $P(O|\lambda) = \Sigma\alpha_{T-1}(i)$

Solution 2

The bottom line?

Forward algorithm solves Problem 1

Forward/backward algorithms solve Problem 2 by computing $\gamma_t(i)$ for each state at each time step (using both forward and backward paths) and choosing the state that maximizes it.

$$\tilde{X}_t = \max_i \gamma_t(i).$$

Solution 2

Why is it necessary to normalize gamma by dividing by P(O | lambda)? Because these probabilities are computed assuming the observation sequence is known (i.e., given O), as opposed to being computed relative to the larger probability space.

Solution 3

Train a model: Given O, N, and M, find λ that maximizes probability of O

We'll iteratively adjust $\lambda = (A,B,\pi)$ to better fit the given observations O

The size of matrices are fixed (N and M)

But elements of matrices can change

It is nice that this works…

…and amazing that it's efficient!

Solution 3

For t=0,1,…,T-2 and i,j in {0,1,…,N-1}, define "di-gammas" as

$$\gamma_t(i,j) = P(X_t=q_i, X_{t+1}=q_j|O,\lambda)$$

Note $\gamma_t(i,j)$ is prob of being in state q_i at time t and transiting to state q_j at t+1

Then $\gamma_t(i,j) = \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)/P(O|\lambda)$

And $\gamma_t(i) = \Sigma\gamma_t(i,j)$

Where sum is from j = 0 to N – 1

Baum-Welch Model Re-estimation

Given di-gammas and gammas…

For $i = 0,1,\ldots,N-1$ let $\pi_i = \gamma_0(i)$

For $i = 0,1,\ldots,N-1$ and $j = 0,1,\ldots,N-1$

$$a_{ij} = \Sigma\gamma_t(i,j)/\Sigma\gamma_t(i)$$

Where both sums are from $t = 0$ to $T-2$

For $j = 0,1,\ldots,N-1$ and $k = 0,1,\ldots,M-1$

$$b_j(k) = \Sigma\gamma_t(j)/\Sigma\gamma_t(j)$$

Both sums from from $t = 0$ to $T-1$ but only t for which $O_t = k$ are counted in numerator

Baum-Welch Model Re-estimation

Why does this work?

For the given observation sequence O, the sum $\Sigma\gamma_t(i)$ gives us the current best estimate for the total probability of being in state i, while $\Sigma\gamma_t(i,j)$ gives us the total probability of transitioning from state i to state j. Hence, the ratio $\Sigma\gamma_t(i,j)/\Sigma\gamma_t(i)$ enables us to re-estimate a_{ij} based on the current model parameters and observation sequence.

Solution 3

To summarize…

1. Initialize $\lambda = (A,B,\pi)$
2. Compute $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i,j)$, $\gamma_t(i)$
3. Re-estimate the model $\lambda = (A,B,\pi)$
4. If $P(O|\lambda)$ increases by more than ε, goto 2 (where ε is small)

Solution 3

Some fine points…

Model initialization

- If we have a good guess for $\lambda = (A,B,\pi)$ then we can use it for initialization
- If not, let $\pi_i \approx 1/N$, $a_{i,j} \approx 1/N$, $b_j(k) \approx 1/M$
- Subject to row stochastic conditions
- But, do ***not*** initialize to exactly uniform values

Stopping conditions

- Stop after some number of iterations and/or…
- Stop if increase in $P(O|\lambda)$ is too small

HMM as Discrete Hill Climb

Algorithm on previous slides shows that HMM is a “discrete hill climb”

HMM consists of discrete parameters

- Specifically, the elements of the matrices

And re-estimation process improves model by modifying parameters

- So, process “climbs” toward improved model
- This happens in a high-dimensional space

Dynamic Programming: The Viterbi Algorithm

Keith Chngg

Brief detour…

For $\lambda = (A,B,\pi)$ as above, it's easy to define a dynamic program (DP)

Executive summary:

DP is forward algorithm, with "sum" replaced by "max"

Dynamic Programming: The Viterbi Algorithm

Let $\delta_0(i) = \pi_i\, b_i(O_0)$ for $i=0,1,\ldots,N-1$

For $t=1,2,\ldots,T-1$ and $i=0,1,\ldots,N-1$ compute

$$\delta_t(i) = \max\left(\delta_{t-1}(j)a_{ji}\right)b_i(O_t)$$

Where the max is over j in $\{0,1,\ldots,N-1\}$

Note that at each t, the DP computes best path for each state, up to that point

So, probability of best path is $\max \delta_{T-1}(j)$

This max gives the highest probability

Not the best path, for that, see next slide

Dynamic Programming: The Viterbi Algorithm

- To determine optimal path
  - While computing deltas, keep track of pointers to previous state
  - When finished, construct optimal path by tracing back points

Dynamic Programming: The Viterbi Algorithm

For example, consider temp example: recall that we observe (0,1,0,2)

$$\pi = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}$$

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{ccc} S & M & L \end{array} \\ \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$

Probabilities for path of length 1:

$$P(H) = \pi_0 b_0(0) = 0.6(0.1) = 0.06 \text{ and } P(C) = \pi_1 b_1(0) = 0.4(0.7) = 0.28.$$

These are the only “paths” of length 1

Dynamic Programming: The Viterbi Algorithm

Probabilities for each path of length 2

$$
\begin{aligned}
P(HH) &= 0.06(0.7)(0.4) = 0.0168 \\
P(HC) &= 0.06(0.3)(0.2) = 0.0036 \\
P(CH) &= 0.28(0.4)(0.4) = 0.0448 \\
P(CC) &= 0.28(0.6)(0.2) = 0.0336
\end{aligned}
$$

Best path of length 2 ending with H is CH

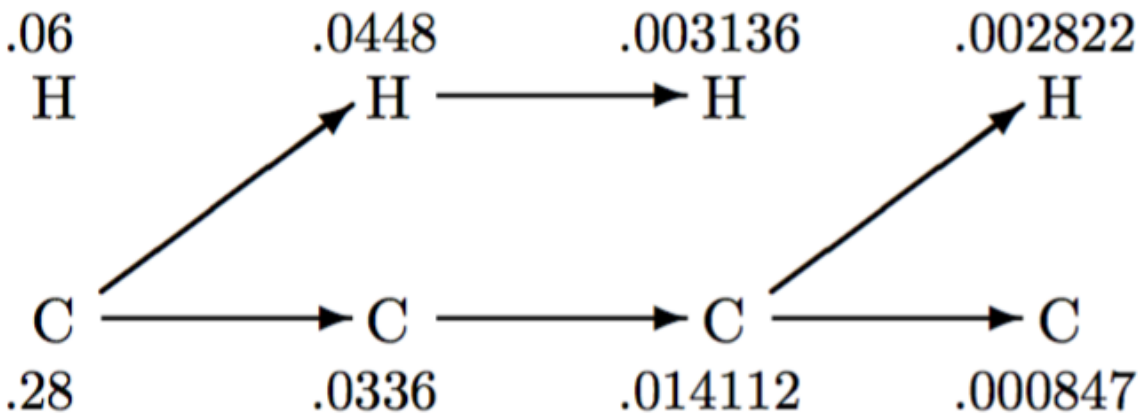
Best path of length 2 ending with C is CC

$$\delta_t(i) = \max \left(\delta_{t-1}(j) a_{ji} \right) b_i(O_t)$$

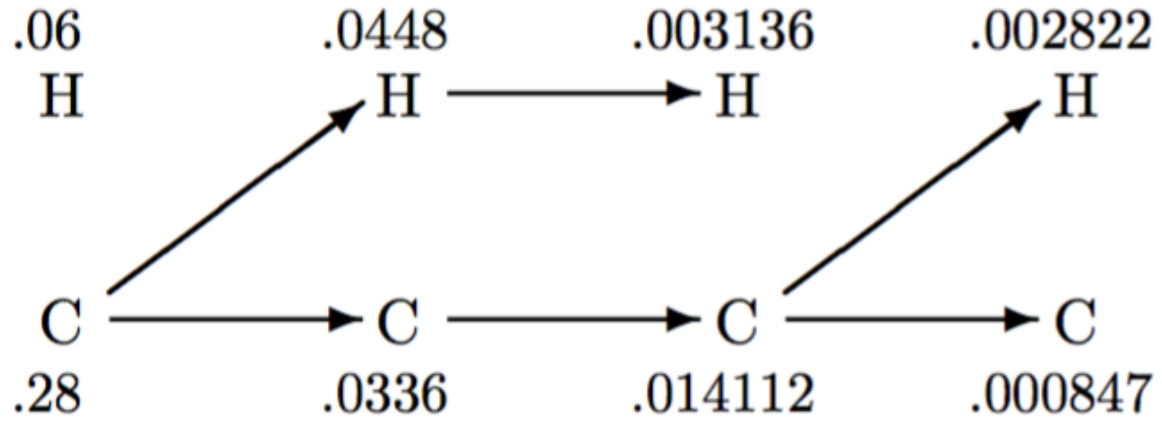
Dynamic Program

Continuing, we compute best path ending at H and C at each step

And save pointers: why?



Dynamic Program



Best final score is .002822

And thanks to pointers, best path is CCCH

But what about underflow?

A serious problem in bigger cases

Underflow Resistant DP

Common trick to prevent underflow:

Instead of multiplying probabilities…

…add logarithms of probabilities

Why does this work?

Because log(xy) = log x + log y

Adding logs does not tend to 0

Note that these logs are negative…

…and we must avoid 0 probabilities

Underflow Resistant DP

Underflow resistant DP algorithm:

Let $\delta_0(i) = \log(\pi_i\, b_i(O_0))$

for i=0,1,…,N-1

For t=1,2,…,T-1 and i=0,1,…,N-1 compute

$$\delta_t(i) = \max\left(\delta_{t-1}(j) + \log(a_{ji}) + \log(b_i(O_t))\right)$$

Where the max is over j in {0,1,…,N-1}

And ***score*** of best path is max $\delta_{T-1}(j)$

As before, must also keep track of paths

Underflow Resistant DP

DP with logs:

https://www.youtube.com/channel/UCiA9pqv9ztRaAySyIg0o5Ow

HMM Scaling

Trickier to prevent underflow in HMM

We consider solution 3

Since it includes solutions 1 and 2

Recall for t = 1,2,…,T-1, i=0,1,…,N-1,

$$\alpha_t(i) = \left(\Sigma \alpha_{t-1}(j) a_{j,i}\right) b_i(O_t)$$

The idea is to normalize alphas so that they sum to 1

Algorithm on next slide

HMM Scaling

Given $\alpha_t(i) = \left(\Sigma\alpha_{t-1}(j)a_{j,i}\right)b_i(O_t)$

Let $a_0(i) = \alpha_0(i)$ for i=0,1,…,N-1

Let $c_0 = 1/\Sigma a_0(j)$

For i = 0,1,…,N-1, let $\alpha_0(i) = c_0 a_0(i)$

This takes care of t = 0 case

Algorithm continued on next slide…

HMM Scaling

For t = 1,2,…,T-1 do the following:
For i = 0,1,…,N-1,

$$a_t(i) = \left(\Sigma \alpha_{t-1}(j) a_{j,i}\right) b_i(O_t)$$

Let $c_t = 1/\Sigma a_t(j)$
For i = 0,1,…,N-1 let $\alpha_t(i) = c_t a_t(i)$

HMM Scaling

Easy to show $\alpha_t(i) = c_0 c_1 \ldots c_t \, \alpha_t(i)$ (♯)

Simple proof by induction

So, $c_0 c_1 \ldots c_t$ is scaling factor at step t

Also, easy to show that

$\alpha_t(i) = \alpha_t(i) / \Sigma \alpha_t(j)$

Which implies $\Sigma a_{T-1}(i) = 1$ (♯♯)

HMM Scaling

By combining (♯) and (♯♯), we have

$$1 = \Sigma \alpha_{T-1}(i) = c_0 c_1 \ldots c_{T-1} \Sigma \alpha_{T-1}(i)$$

$$= c_0 c_1 \ldots c_{T-1} P(O|\lambda)$$

Therefore, $P(O|\lambda) = 1 / c_0 c_1 \ldots c_{T-1}$

To avoid underflow, we compute

$$\log P(O|\lambda) = -\Sigma \log(c_j)$$

Where sum is from j = 0 to T-1

HMM Scaling

Similarly, scale betas as $c_t\beta_t(i)$

For re-estimation,

- Compute $\gamma_t(i,j)$ and $\gamma_t(i)$ using original formulas, but with scaled alphas, betas

This gives us new values for $\lambda = (A,B,\pi)$

"Easy exercise" to show re-estimate is exact when scaled alphas and betas used

Also, $P(O|\lambda)$ cancels from formula

- Use $\log P(O|\lambda) = -\Sigma \log(c_j)$ to decide if iterate improves

All Together Now

Complete pseudo code for Solution 3

Given: $(O_0,O_1,\ldots,O_{T-1})$ and N and M

Initialize: $\lambda = (A,B,\pi)$

A is NxN, B is NxM and π is 1xN

$\pi_i \approx 1/N$, $a_{ij} \approx 1/N$, $b_j(k) \approx 1/M$, each matrix row stochastic, but not uniform

Initialize:

maxIters = max number of re-estimation steps

iters = 0

oldLogProb = $-\infty$

Forward Algorithm

Forward algorithm
With scaling

```
// compute α_0(i)
c_0 = 0
for i = 0 to N − 1
    α_0(i) = π(i) b_i(O_0)
    c_0 = c_0 + α_0(i)
next i

// scale the α_0(i)
c_0 = 1/c_0
for i = 0 to N − 1
    α_0(i) = c_0 α_0(i)
next i

// compute α_t(i)
for t = 1 to T − 1
    c_t = 0
    for i = 0 to N − 1
        α_t(i) = 0
        for j = 0 to N − 1
            α_t(i) = α_t(i) + α_{t−1}(j) a_{ji}
        next j
        α_t(i) = α_t(i) b_i(O_t)
        c_t = c_t + α_t(i)
    next i

    // scale α_t(i)
    c_t = 1/c_t
    for i = 0 to N − 1
        α_t(i) = c_t α_t(i)
    next i
next t
```

Backward Algorithm

Backward algorithm or “beta pass”

With scaling

Note: same scaling factor as alphas

// Let $\beta_{T-1}(i) = 1$ scaled by c_{T-1}
for $i = 0$ to $N - 1$
    $\beta_{T-1}(i) = c_{T-1}$
next i

// β-pass
for $t = T - 2$ to 0 by -1
    for $i = 0$ to $N - 1$
        $\beta_t(i) = 0$
        for $j = 0$ to $N - 1$
            $\beta_t(i) = \beta_t(i) + a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{T+1}(j)$
        next j
        // scale $\beta_t(i)$ with same scale factor as $\alpha_t(i)$
        $\beta_t(i) = c_t \beta_t(i)$
    next i
next t

Gammas

Using scaled alphas and betas

for $t = 0$ to $T - 2$
    $\texttt{denom} = 0$
    for $i = 0$ to $N - 1$
        for $j = 0$ to $N - 1$
            $\texttt{denom} = \texttt{denom} + \alpha_t(i) a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j)$
        next j
    next i
    for $i = 0$ to $N - 1$
        $\gamma_t(i) = 0$
        for $j = 0$ to $N - 1$
            $\gamma_t(i, j) = (\alpha_t(i) a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j))/\texttt{denom}$
            $\gamma_t(i) = \gamma_t(i) + \gamma_t(i, j)$
        next j
    next i
next t

// Special case for $\gamma_{T-1}(i)$
$\texttt{denom} = 0$
for $i = 0$ to $N - 1$
    $\texttt{denom} = \texttt{denom} + \alpha_{T-1}(i)$
next i
for $i = 0$ to $N - 1$
    $\gamma_{T-1}(i) = \alpha_{T-1}(i)/\texttt{denom}$
next i

Re-Estimation

Again, using scaled gammas
So formulas unchanged

```
// re-estimate π
for i = 0 to N − 1
    π_i = γ_0(i)
next i

// re-estimate A
for i = 0 to N − 1
    for j = 0 to N − 1
        numer = 0
        denom = 0
        for t = 0 to T − 2
            numer = numer + γ_t(i, j)
            denom = denom + γ_t(i)
        next t
        a_ij = numer/denom
    next j
next i

// re-estimate B
for i = 0 to N − 1
    for j = 0 to M − 1
        numer = 0
        denom = 0
        for t = 0 to T − 2
            if(O_t == j) then
                numer = numer + γ_t(i)
            end if
            denom = denom + γ_t(i)
        next t
        b_i(j) = numer/denom
    next j
next i
```

Stopping Criteria

Check that probability increases

- In practice, want logProb > oldLogProb + ε

And don't exceed max iterations

```
// Compute log[P(O | λ)]

logProb = 0
for i = 0 to T − 1
    logProb = logProb + log(c_i)
next i
logProb = −logProb

// To iterate or not to iterate, that is the question...

iters = iters + 1
if (iters < maxIters and logProb > oldLogProb) then
    oldLogProb = logProb
    goto α-pass
else
    output λ = (π, A, B)
end if
```

References

M. Stamp, A revealing introduction to hidden Markov models
L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition
R.L. Cave & L.P. Neuwirth, Hidden Markov models for English