MATERIAL FOR HIDDEN MARKOV MODELS
MACHINE LEARNING E20

Mathias Ravn Tversted
Department of Computer Science
Aarhus University
Åbogade 34, 8200 Aarhus N
Tversted@post.au.dk

December 29, 2020

1 Definitioner og basic terminology

Hidden Markov Model A *Hidden Markov Model* consists of

- Transition Probability Matrix: $K \times K$ matrix A that describe K discrete states.
- Emission Probability Matrix: $K \times D$ matrix Φ, where D is possible emissions/symbols.
- Initialization Vector (π) with K entries, one for each state.

We will denote the set of latent states as $Z = \{z_1, \cdots, z_N\}$, the observables as $X = \{x_1, \cdots, x_N\}$ and model parameters as $\Theta = \{\pi, A, \phi\}$

Where the following must apply

- The sum of any column k of A always equal 1 because a column represents the outgoing transition probabilities of a state.
- The sum of any row k of Φ is equal 1, because a row represents a state with d emission possibilities.
- The sum of all indexes in π is equal 1, because a chain must start in some initial state.

A Hidden Markov Model can be used to determine the likelihood of a sequence of observations. It does this with a finite number of states, whose transition probabilities are kind of like a *finite automata* "encoding some process which is being modelled", and where it can emit some output in each state. It is called a *hidden* model if we do not know the probabilities, but are only able to observe the output of the process.

Specifically, our joint distribution is

$$p(X, Z|\Theta) = p(z_1|\pi) \left[\Pi_{n=2}^{N} p(z_n|z_{n-1}, A)\right] \Pi_{n=1}^{N} p(x_n|z_n, \phi) \tag{1}$$

If A and Φ are the same for all n then the HMM is *homogeneous*.

Once all probabilities are in place, the HMM can also be used to generative that which it models. I.E based on a model M - generate a most likely sequence of states from M, followed by a most likely sequence of emissions from these states.

2 Decoding algorithms

2.1 Viterbi decoding

Given X, find Z^* such that:

$$Z^* = \arg\max_Z p(X, Z|\Theta). \tag{2}$$

Where Z^* is the overall most likely explanation of X.

We can express this as:

$$Z_N^* = \arg\max_{Z_N} \max_{Z_1,\ldots,Z_{N-1}} p(x_1, \ldots, x_N, Z_1, \ldots, Z_N). \tag{3}$$

We express the last part of this expression as

$$\omega(Z_N) = \max_{Z_1,\ldots,Z_{N-1}} p(x_1, \ldots, x_N, Z_1, \ldots, Z_N) \tag{4}$$

which is what we wish to maximize.

We can calculate $\omega(Z_n)$ through recursion

- **General case**: $\omega(Z_n) = p(x_n|Z_n) \max_{Z_{n-1}} p(Z_n|Z_{n-1})\omega(Z_{n-1})$
- **Base case:** $\omega(Z_1) = p(x_1, z_1) = p(z_1)p(x_1|z_1)$

We use this to calculate a matrix Ω which can be used for backtracking.

Computing Ω takes $O(K^2N)$ with $O(KN)$ space with memoization.

Backtracking Algorithm

- $Z[N] = \arg\max_k \omega[k][N]$
- for $n = N - 1$ to 1:
  - $Z[n] = \arg\max_k(p(x[n+1]|z[n+1]) \cdot \omega[k][n] \cdot p(z[n+1]|k))$

This algorithm runs in $O(KN)$ time.

When we backtrack we start in the last column of the matrix and select the state that has the maximum value. Then we find the state that was most likely in the Z_{n-1} step. i.e. what gives get largest ω for the state in the column that we picked Z_n.

2.2 Posterior decoding

z_n^* is the most likely state to be in the n'th step.

$$z_n^* = \arg\max_{z_n} p(z_n|x_1, \ldots, x_N) \tag{5}$$

We have that

$$p(z_n|x_1, \ldots, x_N) = \frac{p(x_1, \ldots, x_n, z_n)p(x_{n+1}, \ldots, x_N|z_n)}{p(x_1, \ldots, x_N)} = \frac{\alpha(z_n)\beta(z_n)}{p(X)} \tag{6}$$

Where α is the joint probability of observering up to x_n and being in state z_n, and β is the conditional probability of future observations from x_{n+1} up to x_N if we are in state z_n.

In total, this gives us that

$$z_n^* = \arg\max_{z_n} p(z_n|x_1, \ldots, x_N) = \arg\max_{z_n} \alpha z_n \beta z_n / p(X) \tag{7}$$

Which is found by calculating the value for all of the states Z. This is calculated in $O(K^2N)$ time bounded by α and β.

We also have that $p(X) = \sum_{z_n \in Z} \alpha(z_n)$

2.3 Calculating α and β

The α values are also called *Forward values*. They can be calculated recursively.

- **General case:** $\alpha(z_n) = p(x_n|z_n) \sum_{j=1}^{n-1} p(z_n|z_j)\alpha(z_j)$
- **Base case:** $\alpha(z_1) = p(x_1, z_1) = p(z_1)p(x_1|z_1)$

The β values, also called *Backward* values.

- **General case:** $\beta(z_n) = \sum_{j=0}^{n-1} \beta(j)p(x_{n+1}|z_{n+1})p(z_{n+1}|z_n)$
- **Base case:** $\beta(z_N) = 1$

2.4 Logspace and scaling

To avoid issues with really small values, we can compute $\log(\omega(z_n))$. Because log is monotonically increasing, it preserves inequalities. It also turns products into sums. If a factor is 0, the product becomes 0. We turn $log(0)$ into a representation of "minus infinity". This works because we are using products. This new transformed ω gets denoted as $\hat{\omega}$ and can also be calculated using recursion as so:

- $\hat{\omega}(z_n) = \log p(x_n|z_n) + \max_{j=1}^{n-1}(\hat{\omega}(z_j) + \log p(z_n|z_j))$

The Basis is:

- $\hat{\omega}(z_1) = \log(p(z_1)p(x_1|z_1)) = \log p(z_1) + \log p(x_1|z_1)$

We can't use log transform for α and β since $\log(\Sigma f) \neq \Sigma(\log f)$.

Instead, we calculate a scaled (normalized) version of α called $\hat{\alpha}$ using recursion like so. We calculate the following in each step.

- $\delta(z_n) = p(x_n|z_n) \sum_{j=1}^{n-1} \hat{\alpha}(z_j)p(z_n|z_j)$
- $c_n = \sum_{k=1}^{K} \delta(z_{nk})$
- $\hat{\alpha}(z_{nk}) = \delta(z_{nk})/c_n$

The Basis is:

- $\hat{\alpha}(z_1) = \alpha(z_n)$
- $c_1 = \sum_{k=1}^{K} \pi_k p(x_1|\phi_k)$

The αs has the following relation to $\hat{\alpha}$: $\alpha(z_n) = (\prod_{m=1}^{n} c_m)\, \hat{\alpha}(z_n)$

β can not be scaled using log transform neither. So here we also make a scaling. $\hat{\beta}$ is calculated using recursion as so (c_{n+1} is the c from the forward recursion):

- $\epsilon(z_n) = \sum_{j=1}^{n-1} \hat{\beta}(z_j)p(x_{n+1}|z_j)p(z_j|z_n)$
- $\hat{\beta}(z_n k) = \epsilon(z_n k)/c_{n+1}$

The Basis is:

- $\hat{\beta}(z_N) = 1$

3 Selection of initial model parameters

We can count how many times the transition from j to k is taken, over how many times a transition from state j is taken.

$$A_{jk} = \frac{\sum_{n=2}^{N} z_{n-1,j} z_{nk}}{\sum_{l=1}^{K} \sum_{n=2}^{N} z_{n-1,j} z_{nl}} \tag{8}$$

With *pseudo-counting* we assume that every transition has been seen at least once.

$$\pi_k = \frac{z_1 k}{\sum_{j=1}^{K} z_{1j}} \tag{9}$$

We can take how many times a symbol i is emitted from state k over how many times a symbol os emitted from state k.

$$\phi_{ik} = \frac{\sum_{n=1}^{N} z_{nk} x_{ni}}{\sum_{n=1}^{N} z_{nk}} \tag{10}$$

This yields a *maximum likelihood estimate* (MLE) Θ^* of $P(X, Z|\Theta)$. This is known as *training by counting* and is a nice analytic solution.

4 Training

4.1 Viterbi Training

If the latent states Z are unknown, but we only have a sequence X of observations, then we are looking to solve

$$\max p(X|\Theta) = \sum_{Z} p(X, Z|\Theta) w.r.t \Theta \tag{11}$$

This is a hard problem, and to solve it we will use *Viterbi Training.*

Viterbi Training

1. Decide on initial parameter Θ^0
2. Compute most likely sequence of states Z^* under X using the Viterby Algorithm, yielding Θ^i
3. Update to Θ^{i+1} by pseudocounting according to (X, Z^*)
4. Repeat 3-4 until $P(X, Z^*|\Theta^i)$ makes you happy, or the sequence of states does not change

This yields $\text{VIT}_X(\Theta) = \max_Z p(X, Z|\Theta)$. This is NOT an MLE, but it works.

$p(X|\Theta^i_{\text{Vit}})$ is not guaranteed, but usually is, close to $p(X|\Theta^*_X)$

4.2 Expectation Maximisation

Given a sequence X of observations, pick an initial set of parameters Θ^0_{EM} and consider expectation of $\log p(X, Z|\theta)$ over Z (given X and Θ^0_{EM}) as a function of Θ. This can be solved analytically for HMMs.

TLDR: We want to maximise the log likelihood $\log p(X, Z|\Theta)$ as a function of Θ.

E-Step Define the Q-function as

$$Q(\Theta, \theta^{old}) = \sum_{Z} p(Z|X, \Theta^{old}) log p(X, Z|\Theta) \tag{12}$$

Recall definition of entropy.

M-Step Maximise $Q(\Theta, \Theta^{old})$ w.r.t to Θ.

When we iterate this function, likelihood $p(X|\Theta)$ converges to a (local) maximum.

4.2.1 EM algorithm

Init: Pick suitable non-zero parameters (zero remains zero)

E-Step Run forward and backward algorithms for current choice of parameters to get parameters of Q-function. (WHY?)

Stop?: Compute likelihood $p(X|\Theta)$. Stop if this meets stopping criteria

M-Step Compute new parameters using the values stored by the forward and backward algorithms. Repeat until you are happy.

Running time is $O(K^2N + KK + K^2NK + KDN)$ per iteration, or $O(K^2N + KDN)$ with memorization.

5 Extensions