MATLAB Codes for Implementation of Hidden Markov Model

Vikram Voleti

Indian Institute of Technology Kharagpur

MATLAB CODES

- Hidden Markov Model
  - Forward Algorithm
  - Backward Algorithm
  - Baum-Welch Algorithm
  - Viterbi Algorithm
- Finger-Tip Gesture Recognition [Triangle, Square or Diamond]

Hidden Markov Model

The parameters of a Hidden Markov Model are *Pi* : the initial probability matrix, *a* : the transition probability matrix, and *b* : the emission matrix.

The various variables used in describing an HMM are:

N = total number of states,

M = total number of possible observations,

T = total number of observations made (for each observation set),

EG = total number of examples/observation sets recorded.

Hence, the dimensions of the matrices used in the algorithms are:

Pi : *Nx1* matrix : Vector of initial probabilities of states,

a : *NxN* matrix : Probability of transition from state S_i to state S_j,

b : *NxM* matrix : Probability of observing V_k for state S_i,

Ob : *Mx1* matrix : Vector of all possible observations,

O : *EGxT* matrix : Matrix of *EG* no. of *1xT* dimensional observation sets,

Alpha : *NxT* matrix; *Alpha(i,t)* = Probability of observing partial observation sequence from *start* to time *t*, i.e. $O_1,O_2,...,O_t$, and being in state S_i, at time *t*,

Betaa : *NxM* matrix; *Betaa(i, find(Ob==O(eg,t),1))* = Probability of partial observation sequence from *t+1* to *end*, i.e. $O_{t+1},...,O_T$, given the state at time *t* was S_i.

Forward Algorithm

The Forward Algorithm calculates the quantity *Alpha*, given the inputs *Pi*, *a*, *b*, *Ob* and *O*. *Alpha(i,t)* is the probability of observing partial observation sequence from start to time *t*, i.e., $O_1,O_2,...,O_t$, and being in state S_i, at time *t*. The function call is:

[Alpha, c, P] = ForwardAlgo(Pi, a, b, Ob, O)

where *c* is a *Tx1* matrix, *c(t)* = Probability of the partial observation sequence till time *t*, and

P is an *Nx1* matrix, *P(n)* = Probability of being in state n at the end time *T*.

Backward Algorithm

The Backward Algorithm calculates the quantity *Betaa* (in order to avoid confusion with the MATLAB built-in "Beta"), given the inputs *Pi*, *a*, *b*, *Ob* and *O*. *Betaa(i, find(Ob==O(eg,t),1))* is the probability of observing the partial observation sequence from time *t+1* to *end*, i.e. $O_{t+1},...,O_T$, given the state at time *t* was S_i . The function call is:

Betaa = BackwardAlgo(Pi, a, b, Ob, O)

Baum-Welch Algorithm

The Baum-Welch Algorithm is used to train the Hidden Markov Model using training data, i.e. to determine the values of the parameters of the Hidden Markov Model, viz. *Pi* : the initial probability matrix, *a* : the transition probability matrix, and *b* : the emission matrix. The training data is in the form of the observation sets *O*.

Initially, the Baum-Welch algorithm requires certain (generally random) initial values for these matrices, and the values of *Alpha* and *Betaa* determined using these initial values. In each iteration of the Baum-Welch algorithm, it calculates new values for *Pi*, *a* and *b, Alpha* and *Betaa*. It iterates its algorithm for a total of *maxIters* times to get the best possible values that concur with the training data *O*. As the algorithm iterates, *iters* is the variable used to store the current iteration number. *PiNew*, *aNew*, *bNew, AlphaNew* and *BetaaNew* are the new values of *Pi*, *a*, *b, Alpha and Betaa*, respectively, calculated in the next iteration. *oldLogProb* is a variable used to store the log-likelihood which describes how correct the current values for *Pi*, *a* and *b* are. This shall be used to compare with *logProb*, the same quantity calculated with the new values *PiNew*, *aNew* and *bNew*, to check if the algorithm has converged. Finally, lP is the matrix used to store all the values of *logProb* calculated in each iteration.

As an example, the initialisation of variables is:

```
Pi = rand(N,1); %random initialisation
Pi = Pi/sum(Pi); %ensuring all values sum to 1

a = rand(N,N); %random initialisation
a = a./repmat(sum(a,2), [1 N]); %ensuring every row sums to 1

b = rand(N,M); %random initialisation
b = b./repmat(sum(b,2), [1 M]); %ensuring every row sums to 1

iters = 0;
maxIters = 100;
oldLogProb = -Inf;
lP = zeros(0,1);

Alpha = ForwardAlgo(Pi, a, b, Ob, O);
Betaa = BackwardAlgo(Pi, a, b, Ob, O);
```

The function call is:

```
[PiNew, aNew, bNew, AlphaNew, BetaNew, logProb, lP] = BaumWelsh(Pi, a, b, Ob, O, Alpha, Betaa, iters,
maxIters, oldLogProb, lP);
```

Viterbi Algorithm

To determine the most probable state sequence *Q*, given *Pi* : the Initial Probability matrix, *a* : Transition matrix, *b* : Emission matrix, *Ob* : matrix of Possible Observations, and *O* : matrix of observations (one observation set). The function call is:

```
Q = ViterbiAlgo(Pi, a, b, Ob, O)
```

Finger-Tip Gesture Recognition [Triangle, Square, Diamond]

Using the MATLAB codes for Hidden Markov Models, I have developed a Gesture Recognition code which tracks the tip of a finger and recognises the gesture made by it as either a *Triangle* or a *Square* or a *Diamond* (or that it doesn't know). A detailed explanation of each step is explained in the code itself.

I have used two cascaded HMM's for this purpose. The first HMM recognises the sequence of directional lines made, the possible states are being *Right*, *Down*, *Left*, *Up*, *Down-Right*, *Down-Left*, *Up-Left* and *Up-Right*. The second HMM recognises the shape made bases on the sequence of directional lines, the possible states being *Triangle*, *Square* and *Diamond*.

This code was made only to illustrate the correctness of the HMM algorithms. There are simpler ways to accomplish the same task, the first being to use only one HMM, but his would require a lot of training data. The two sets of values of *Pi*, *a* and *b* used in this code were statistically determined by observing the chance of occurrence of the respective states in the three shapes drawn. Therefore, there could be errors in recognition, but it serves its purpose of demonstrating that the HMM codes work.