Building a Hidden Markov model (HMM) in three weeks

John Goldsmith

January 5, 2014

This is a problem set in three parts, done over three weeks, in which you will build an HMM. HMMs are widely used in computational linguistics and other fields (such as bioinformatics), and many other models grew out of HMMs. The first week involves writing code to build a functioning HMM which does not know how to learn parameters. The second week involves adding code so that parameters can be learned from the data—this is the Baum-Welch algorithm. The third part adds some additional functionality.

In the first part of this assignment, you will set up a 2-state HMM, but you should write your code so that any number of states is possible, so that you can use the same code for data in the future when more than two states will be desirable.

The particular application we will use the HMM for is to separate phonemes into two sets, which will empirically turn out to be vowels and consonants. We would like to apply this to the Voynich manuscript later. Therefore you must write your code so that the number of symbols emitted by the HMM (which is our "alphabet"), and the set of symbols itself, is only determined at run-time, by reading through the data.

1 Part 1, Week 4

Write a program that analyzes letters as produced by two states, S_1 and S_2 in an state-emision hidden Markov model. Each of the 2 states will produce each of the symbols in the alphabet with a non-zero probability. From the point of view of grading, there will be a total of 13 points given for this week's assignment. That is more than usual, because there is more work than usual.

Note that the chapter from Manning and Schütze describes a model in which the probability of emission is conditioned by both the state transitioning from *and* the state transitioning to. I am asking you to do a slightly simpler task, one in which the model is that the probability of emission is conditioned only by the state transitioning from.

For your data, begin with a very simple data set: two words, *babibabi#* and *didadida#*, is good to use while you are writing your code and eliminating bugs. Then use the set of approximately 1000 English words (in standard orthography) that you are given by us. Please assume that each word ends with #. For example, if you have the word "the" in your corpus, treat it as if it were "the#". If you have N words in your corpus, you must run N strings through your HMM, but you will end up with (expected) counts of each of the approximately 28 letters and the states the HMM was in when it emitted them for the whole corpus: *your counts will sum over all of the words you run through the HMM.*

1. **Initialization (3 points)**: Start building the basic structure of the program, including a function to read in the corpus, set up the alphabet of symbols used by the corpus, and to create two states. Each state must have two sets of probabilities associated with it: a set of transition probabilities to the set of states, and a set of emission probabilities. Write a function to assign initial probabilities to these variables in such a way that all numbers that should add up to 1 (i.e., form a distribution) do so. Assign probabilities for the state transitions that form a well-formed distribution. Also assign a distribution to the choice of which state is entered initially.

   In this assignment, you will output various working variables to a text file so that you can be sure you are calculating the right quantities. Once the program is working correctly, you will want to turn off most of this output. Create a variable called "VerboseFlag" which can be set to True or False, and be sure to make most of your output statements be of the form "if Verbose Flag…".

Output: Print the initial values for A, B, and Π; it should look like this.[1]

```
---------------------------------
-  Initialization               -
---------------------------------
Creating State 0
Transitions
     To state   0  0.3763
     To state   1  0.6237

Emission probabilities
     Letter    b  0.4095
     Letter    n  0.2112
     Letter    #  0.1724
     Letter    a  0.1050
     Letter    i  0.1019
     Total: 1.0

Creating State 1
Transitions
     To state   0  0.0381
     To state   1  0.9619

Emission probabilities
     Letter    n  0.3602
     Letter    a  0.2421
     Letter    i  0.2010
     Letter    b  0.1513
     Letter    #  0.0454
     Total: 1.0


---------------------------
Pi:
     State   0       0.8057
     State   1       0.1943
```

2. **Forward and Backward (3 points)**: Write a function that calculates the forward probability for each α_i, as defined in equation (9.10). Write a function that calculates the backward probability for each β_i, as in (9.11).

   Output: Print the alpha and beta values for each state and each time, for each of the Keywords above.

   Suppose the word were "babi#". I have printed out the calculation of the alpha in gorey detail, and then summarized the alphas and betas at each moment below:

[1] Note that the letters are sorted by emission probability for each state. That does not matter now, but later you will want to quickly see which letters each state assigns high probabilities to.

```
*** word: babi#***

Forward
Pi of state      0      0.8057
Pi of state      1      0.1943
     time 2: 'b'
           to state:0
               from state   0   Alpha: 0.1242
               from state   1   Alpha: 0.1253
           to state:1
               from state   0   Alpha: 0.2058
               from state   1   Alpha: 0.2341
          Total at this time: 0.3594    (sum of last entry in each of the to-states)
     time 3: 'a'
           to state:0
               from state   0   Alpha: 0.004949
               from state   1   Alpha: 0.00711
           to state:1
               from state   0   Alpha: 0.008203
               from state   1   Alpha: 0.06271
          Total at this time: 0.06982    (sum of last entry in each of the to-states)
     time 4: 'b'
           to state:0
               from state   0   Alpha: 0.001096
               from state   1   Alpha: 0.001458
           to state:1
               from state   0   Alpha: 0.001816
               from state   1   Alpha: 0.01094
          Total at this time: 0.0124    (sum of last entry in each of the to-states)
     time 5: 'i'
           to state:0
               from state   0   Alpha: 5.591e-05
               from state   1   Alpha: 0.0001398
           to state:1
               from state   0   Alpha: 9.267e-05
               from state   1   Alpha: 0.002209
          Total at this time: 0.002348    (sum of last entry in each of the to-states)
     time 6: '#'
           to state:0
               from state   0   Alpha: 9.067e-06
               from state   1   Alpha: 1.289e-05
           to state:1
               from state   0   Alpha: 1.503e-05
               from state   1   Alpha: 0.0001115
          Total at this time: 0.0001244    (sum of last entry in each of the to-states)
Alpha:
     Time  1 State   0:      0.8057      State   1:      0.1943
     Time  2 State   0:      0.1253      State   1:      0.2341
     Time  3 State   0:     0.00711      State   1:     0.06271
     Time  4 State   0:    0.001458      State   1:     0.01094
     Time  5 State   0:   0.0001398      State   1:    0.002209
     Time  6 State   0:   1.289e-05      State   1:   0.0001115
```

```
Beta:
     Time  1 State   0:   0.0001403      State   1:   5.863e-05
     Time  2 State   0:   0.0002597      State   1:   0.0003925
     Time  3 State   0:    0.004045      State   1:    0.001525
     Time  4 State   0:    0.009499      State   1:      0.0101
     Time  5 State   0:      0.1724      State   1:     0.04543
     Time  6 State   0:           1      State   1:           1
```

3. **Total probability (1 point):** Calculate and print the sum of the probabilities assigned to the strings of the corpus by the current model.

4. **Expected counts of state-production of letters (3 points)**: For each occurrence of each letter in the corpus, calculate the expected count of its production from State S_1 and its production from state S_2. This is the most delicate part of the calculation. These soft counts should sum to 1.0 over all of the state transitions for each letter generated.

```
---------------------------
Soft counts
-------------------------

String probability from Alphas: 0.00012442
String probability from Betas:  0.00012442

     Letter: b
          From state: 0
               to state:    0       0.2591;
               to state:    1       0.6493;

          From state: 1
               to state:    0       0.0023;
               to state:    1       0.0892;

     Letter: a
          From state: 0
               to state:    0       0.1609;
               to state:    1       0.1006;

          From state: 1
               to state:    0       0.0703;
               to state:    1       0.6683;

     Letter: b
          From state: 0
               to state:    0       0.0837;
               to state:    1       0.1475;

          From state: 1
               to state:    0       0.0276;
               to state:    1       0.7412;

     Letter: i
          From state: 0
               to state:    0       0.0774;
```

```
          to state:    1       0.0338;

     From state: 1
          to state:    0       0.1162;
          to state:    1       0.7725;

Letter: #
     From state: 0
          to state:    0       0.0729;
          to state:    1       0.1208;

     From state: 1
          to state:    0       0.0308;
          to state:    1       0.7756;
```

5. **Maximization, part 1 (2 points)**: Now you must calculate, for each state, what the total soft counts are for each letter generated by that state over the whole corpus. And then normalize that into a distribution over the alphabet, which gives you the probability distribution for the alphabet for that state.

```
Emission

     From State: 0
          letter: a
               to state  0       0.193   running total 0.193
               to state  1       0.386   running total 0.386

          letter: i
               to state  0       0.114   running total 0.500
               to state  1       0.164   running total 0.550

          letter: #
               to state  0       0.133   running total 0.683
               to state  1       0.354   running total 0.904

          letter: b
               to state  0       0.343   running total 1.247
               to state  1       1.140   running total 2.044

          letter: n
               to state  0       0.138   running total 2.182
               to state  1       0.707   running total 2.751
          letter:  a  probability: 0.1404
          letter:  i  probability: 0.0596
          letter:  #  probability: 0.1288
          letter:  b  probability: 0.4143
          letter:  n  probability: 0.2570

     From State: 1
          letter: a
               to state  0       0.090   running total 0.090
```

```
          to state  1       1.614   running total 1.614

     letter: i
          to state  0       0.240   running total 1.854
          to state  1       1.836   running total 3.450

     letter: #
          to state  0       0.063   running total 3.513
          to state  1       1.646   running total 5.096

     letter: b
          to state  0       0.030   running total 5.126
          to state  1       0.860   running total 5.956

     letter: n
          to state  0       0.039   running total 5.995
          to state  1       1.293   running total 7.249
     letter:  a  probability: 0.2226
     letter:  i  probability: 0.2533
     letter:  #  probability: 0.2270
     letter:  b  probability: 0.1187
     letter:  n  probability: 0.1784
```

Maximization, part 2 (1 point): In similar fashion, you will recalculate the transition probabilities, and the Pi probabilities; you should produce an output like this:

```
      From_State: 0
          To state:   0 prob: 0.3350 (0.921 over 2.751)
          To state:   1 prob: 0.6650 (1.829 over 2.751)

     From_State: 1
          To state:   0 prob: 0.0637 (0.462 over 7.249)
          To state:   1 prob: 0.9363 (6.788 over 7.249)
---------------------------
Pi:
     State   0       0.8057
     State   1       0.1943
```

2 Part 2, Week 5

In this section, you will use put together the Expectation and Maximization functions that you created in the first part, to arrive at an HMM that finds local optima for parameter values.

Then we will use the HMM to classify letters. You will see that you can determine if your HMM is learning well by seeing the probability that it assigns to the data—the higher the probability, the better it is discovery structure in the data. And you will also see that not all initial assumptions about the parameters will lead to finding the highest probabilities: the system can be trapped in a local optimum that is not at all a global optimum. This is a good lesson to learn!

1. **Expectation-Maximization (2 points)**: You will create a loop with the Expectation and Maximization functions that you have already written. You will need to set a stopping condition for the Expectation portion. Give your program the ability to stop either (i) based on the number of iterations, or (ii) because the sum of the probabilities of all of the words is not increasing significantly.

2. Now you will have gotten your HMM to work correctly, and you will want to turn the VerboseFlag off, and only to output the values of interest, which are computed in the final iteration. What you want to know is the values of the distributions that have been learned: Pi, A, and B. We can compute the log ratio of the probability assigned to each letter by the two states, and sorting the letters by that value (and separating at zero) should divide the letters into vowels and consonants.

3. **Local maxima (3 points)**: Run your program 20 times with the same set of data, assigning initial values for the parameters randomly each time. You will find that at the end of its run, the total probability of the data is not always the same. You will find that looking at the state-transition parameter values at the end of the learning process allows you to determine whether the total probability is as large as it can be. Where should the parameters be at the end? How does the initial assignment of these values affect the ability of the system to end up in the spot where the state transition parameters are optimized?

4. **Viterbi (2 points):** Write a function to implement the Viterbi algorithm, finding the best path through the HMM to generate any of the words in the training data. You will apply this function to each word after all the learning is done—and we will start the HMM learning in the next step. But for now, you want to be able to compute the Viterbi path so you can see the effects of the learning. Your output should look essentially like this, for each word (the word that is analyzed below is "nani#"):

```
----------------------------
      Viterbi path
----------------------------
nani#

Delta[1] of stateno  0   0.3955
Delta[1] of stateno  1   0.6045

Time t+1:  2   n
at state  0:
          from-state  0: 0.02382
          from-state  1: 0.1457
best state to come from is   1 (at  0.1457)

at state  1:
          from-state  0: 0.04371
          from-state  1: 0.09209
best state to come from is   1 (at 0.09209)

Time t+1:  3   a
at state  0:
          from-state  0: 0.005743
          from-state  1: 0.001021
best state to come from is   0 (at 0.005743)

at state  1:
          from-state  0: 0.01054
          from-state  1: 0.0006454
best state to come from is   0 (at 0.01054)

Time t+1:  4   n
at state  0:
          from-state  0: 0.000346
          from-state  1: 0.00254
best state to come from is   1 (at 0.00254)
```

```
at state  1:
          from-state  0: 0.0006348
          from-state  1: 0.001605
best state to come from is   1 (at 0.001605)

Time t+1:  5   i
at state  0:
          from-state  0: 0.0001977
          from-state  1: 0.000308
best state to come from is   1 (at 0.000308)

at state  1:
          from-state  0: 0.0003628
          from-state  1: 0.0001947
best state to come from is   0 (at 0.0003628)

Time t+1:  6   #
at state  0:
          from-state  0: 3.515e-05
          from-state  1: 1.521e-05
best state to come from is   0 (at 3.515e-05)

at state  1:
          from-state  0: 6.45e-05
          from-state  1: 9.614e-06
best state to come from is   0 (at 6.45e-05)
Path readout
Xhat:      1 at t+1   6
Xhat:      1 at t+1   5
Xhat:      0 at t+1   4
Xhat:      1 at t+1   3
Xhat:      0 at t+1   2
Xhat:      1 at t+1   1

Viterbi path:
time:      1       2       3       4       5       6
state:     1       0       1       0       1       1
```

Bonus question: Are there any letters of the alphabet which are sometimes generated by one state, and sometimes by the other, in the Viterbi parse of two different words? Why would this be the case?

3 Part 3, Week 6

This week, you will produce some graphical output so that you can better see what the algorithm is doing. If you use Python or some version of C, you can use (Py)GraphViz to produce useful output without much trouble.

1. It is extremely convenient to look at a graph of the evolution of the values during the learning phase. One easy way to do this is with the python pyx graphics package. Here is a simple function to output a pdf file with a graph of the data in a list called *data*, where each entry in the list is a pair of (x,y) values. Please feel free to use a wider range of interesting colors and options.

```
def     Plot2D(data, mytitle, xaxistitle="", yaxistitle="",mycolor=(0,0,0)):
          g = graph.graphxy(width=8,x=graph.axis.linear(min=-0, max=1,title=xaxistitle),
                y=graph.axis.linear(min=-0, max=1,title=yaxistitle))
          g.plot(graph.data.points(data, x=1,y=2),styles=
                [graph.style.line([ style.linestyle.solid]) ] )

          g.text(g.width/2,g.height +0.2,mytitle,[text.halign.center,
                text.valign.bottom,text.size.large])

          g.writePDFfile(mytitle)
```

If you are not using Python, it is also convenient to make graphs with R. See, for example, `http://www.statmethods.net/graphs/` and `http://www.statmethods.net/graphs/dot.html` (I have not used the information at those websites, though I have used R to create eps versions of data of this sort).

2. **Plots (3 points)**: Produce a plot of a sequence of points (one for each EM cycle) (x,y), where x = pr (transition from state 0 to state 1) and y = pr (transition from state 1 to state 0). For each, indicate what the final total probability is that was assigned to the data at the end of the learning process.

3. **Bonus project: Phonemic transcription (optional 2 points)**

   From `http://svn.code.sf.net/p/cmusphinx/code/trunk/cmudict/` you can download the CMU dictionary, which gives a phonemic transcription of a large English vocabulary. Convert your wordlist into a phonemic representation, and analyze the data with your HMM. Explain the significance of the differences you find between the structure that the HMM has learned from orthographic form and from phonemic form.