HMM

Alexander Resare, Nikita Suprun

March 27, 2024

Q1-Q6

See attached handwritten pages in the Appendix

Q7

Does the algorithm converge?: The algorithm converges for the given initial guesses. We will discuss the definition of convergence in this question later, here we will just state that we see that the values returned by the algorithm differ somewhat from the HMM dynamics used to generate the observations, yet most entries are close.

How many observations do you need for the algorithm to converge?: The algorithm converges independently of how many observations we are using.
Here we are using the following definition of convergence: An iterative algorithm is said to converge when, as the iterations proceed, the output gets closer and closer to some specific value.
The convergence is obvious as if we only use 3 observations, then we will adjust all entries in matrices A, B and π to the values that the observations hint on and eventually no further adjustments will be made. This result is independent of the number of observations.
We also state that the fewer observations that are used the faster the convergence, however the converged matrices are further away from the true A, B and π.

How can you define convergence?: See the definition above. Convergence for the algorithm can be for instance defined as: all the distances are less than 10^{-3} or some other power of 10. There are many ways to measure distance between matrices! To measure the distance between the previous iteration and the current one, we could take the absolute value of the difference of each element in the matrix, for each respective matrix A, B and π, from the past and the current iteration.

Q8

For simplicity we will initialize our initial guesses as was suggested in Q7. The following result is obtained using all available observations and 1000 iterations:

$$A_c = \begin{pmatrix} 0.7 & 0.01 & 0.29 \\ 0.1 & 0.81 & 0.09 \\ 0.19 & 0.3 & 0.51 \end{pmatrix}$$
$$B_c = \begin{pmatrix} 0.69 & 0.23 & 0.08 & 0.01 \\ 0.07 & 0.41 & 0.28 & 0.24 \\ 0.0 & 0.0 & 0.35 & 0.65 \end{pmatrix}$$
$$\pi_c = \begin{pmatrix} 1.0 & 0.0 & 0.0 \end{pmatrix}$$

The problem of measuring distance between matrices is that there are many different metrics one could use (Not sure what the question is referring to actually!?). Let's give some examples:

$d_1(\mathbf{AB}) = \sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij} - b_{ij}|$

$d_2(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij} - b_{ij})^2}$

$d_\infty(\mathbf{A}, \mathbf{B}) = \max_{1 \le i \le n} \max_{1 \le j \le n} |a_{ij} - b_{ij}|$

$d_m(\mathbf{A}, \mathbf{B}) = \max\{\|(\mathbf{A} - \mathbf{B})\mathbf{x}\| : \mathbf{x} \in \mathbb{R}^n, \|\mathbf{x}\| = 1\}.$

Since we are interesting in finding the entries of A, B and π, I would argue d_1 is a simple and intuitive metric one could use to measure the difference. This would overcome the problem of not having a well-defined distance we had initially.

In our case the distances are given as follows:

$d_1(A_c, A) = 0.04 + 0.04 + 0.01 + 0.01 + 0.01 + 0.01 = 0.12$

$d_1(B_c, B) = 0.01 + 0.03 + 0.02 + 0.01 + 0.03 + 0.01 + 0.02 + 0.04 + 0.1 + 0.15 + 0.05 = 0.47$

$d_1(\pi_c, \pi) = 0$

Q9

Let's start by training an HMM using 2 hidden states instead of 3. A becomes a 2x2 matrix, B becomes a 2x4 matrix and π is a 1x2 matrix (row vector). We initialized the model as following:

A_guess_flat = [0.54, 0.46, 0.47, 0.53, 0.22, 0.78]

B_guess_flat = [0.5, 0.2, 0.11, 0.19, 0.22, 0.28, 0.23, 0.27]

pi_guess = [0.3, 0.7]

The following matrices were obtained:

$$A_c = \begin{pmatrix} 0.64 & 0.36 \\ 0.19 & 0.81 \end{pmatrix}$$
$$B_c = \begin{pmatrix} 0.7 & 0.17 & 0.08 & 0.06 \\ 0.0 & 0.29 & 0.32 & 0.39 \end{pmatrix}$$
$$\pi_c = \begin{pmatrix} 1.0 & 0.0 \end{pmatrix}$$

We observe that some entries get a good approximation, while others don't. In particular diagonal entries of A get the best approximation, i.e. when a hidden state has no transition (the same at next timestamp). For the true B, we see that in hidden state 4 observations 1 and 2 have probabilities 0 and 0.1, while observation 4 has probability 0.7. Consequently, we see see how it impacted the trained HMM: The probabilities of observations 1 and 2 are underestimated while probability of observation 4 is overestimated for hidden states 1 and 2.

Since A, B and π are stochastic matrices each row always sums up to 1 and therefore some entries get skewed values when there are fewer entries. Also we observe that the learning rate is higher as there are fewer entries that have to be updated!

Let's now train an HMM using 4 hidden states instead of 3. A becomes a 4x4 matrix, B becomes a 4x4 matrix and π is a 1x4 matrix (row vector). We initialized the model as following:
A_guess_flat = [0.54, 0.26, 0.10, 0.10, 0.19, 0.53, 0.18, 0.10, 0.22, 0.18, 0.4, 0.2, 0.22, 0.18, 0.4, 0.2]
B_guess_flat = [0.5, 0.2, 0.11, 0.19, 0.22, 0.28, 0.23, 0.27, 0.19, 0.21, 0.15, 0.45, 0.19, 0.21, 0.15, 0.45]
pi_guess = [0.3, 0.2, 0.3, 0.2]
The following matrices were obtained:

$$A_c = \begin{pmatrix} 0.78 & 0.01 & 0.0 & 0.21 \\ 0.06 & 0.44 & 0.11 & 0.39 \\ 0.01 & 0.33 & 0.64 & 0.02 \\ 0.16 & 0.0 & 0.36 & 0.48 \end{pmatrix}$$

$$B_c = \begin{pmatrix} 0.67 & 0.17 & 0.09 & 0.06 \\ 0.4 & 0.58 & 0.03 & 0.0 \\ 0.0 & 0.32 & 0.36 & 0.32 \\ 0.0 & 0.0 & 0.37 & 0.63 \end{pmatrix}$$

$$\pi_c = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

We see that there are more entries that have a value of 0.0 or close to that, i.e. the matrix becomes more sparse. This is logical as a stochastic matrix has row sum of 1 and the probabilities are not uniformly distributed. However, the main reason is that some hidden states are redundant and after enough iterations these converge to 0. The rate of learning has become slower, i.e. the conversion is slower than with 3 hidden states. This is due to the fact we have to update more entries in the matrices.

I would argue that if we have a lot of data, it is preferable to choose 4 hidden states for 4 observations, as the future predictions will be more accurate by virtue of including more features. The redundant hidden states (if there are any) will just converge to 0 or close to that value probability. However, in case we have less data, given the learning rate is faster for a smaller number of hidden states (i.e. we need less data points for matrices to get a good approximation), we might want to choose 3 hidden states. The matrix having 4 hidden states might not get enough data to be well-trained and perform more poorly.

Considering the above, one might determine the optimal setting by studying at which number of hidden states we get redundant states, i.e. the matrices start to become sparse. Otherwise, we have to optimize for the number of data points.

Q10

Initialize your Baum-Welch algorithm with a uniform distribution. How does this affect the learning?: If we set the probabilities exactly uniform, the algorithm fails and does not update any values. If the probabilities are not strictly uniform the algorithm succeeds but the learning rate is slower as the probabilities stay close to uniform for a few iterations.

Initialize your Baum-Welch algorithm with a diagonal A matrix and $\pi = [0, 0, 1]$. How does this affect the learning?: The algorithm fails, as the algorithm fails to update the 0 entries of A: digamma together with gamma become 0, due to division by 0 we get an exception. Hence, in our initial guess we want all entries to be non-zero.

Initialize your Baum-Welch algorithm with a matrices that are close to the solution. How does this affect the learning?: An initial guess close to the solution contributes to fast convergence, as the algorithm is able to skip many iterations in order to arrive at the approximation.