KTH Computer Science and Communication

DT2119 Lab2: Hidden Markov Models with Gaussian Emissions

1 Objective

After this exercise you should be able to:

- implement the algorithms for the evaluation and decoding of Hidden Markov Models (HMMs),
- use your implementation to perform isolated word recognition,
- implement the algorithms for training Gaussian Hidden Markov Models (G-HMMs),
- explain the meaning of the forward, backward and state posterior probabilities evaluated on speech utterances,

The lab is designed in Python, but the same functions can be obtained in Matlab/Octave.

2 Task

The overall task is to implement and test methods for isolated word recognition:

- combine phonetic HMMs into word HMMs using a lexicon
- implement the *forward-backward* algorithm,
- use it compute the log likelihood of spoken utterances given a Gaussian HMM
- perform isolated word recognition
- implement the *Viterbi algorithm*, and use it to compute Viterbi path and likelihood
- compare and comment Viterbi and Forward likelihoods
- implement the *Baum-Welch algorithm* to update the parameters of the emission probability distributions

In order to pass the lab, you will need to follow the steps described in this document, and present your results to a teaching assistant in the designated time slots.

3 Data and model set

The speech data used in this lab is similar but not the same as in Lab 1. You can load the array containing speech utterances with the commands:

```
>>> import numpy as np
>>> data = np.load('lab2_data.npz')['data']
```

The data contains also MFCC features (`lmfcc` key), but you are welcome to test how the algorithms perform on the MFCCs computed with your own code from Lab 1. Refer to the instructions to Lab 1 for more information about the data structures.

Additionally, the `lab2_models_onespkr.npz` and `lab2_models_all.npz` files contain the parameters of the models you are going to use to test your functions and the `lab2_example.npz` file contains an example that can be used for debugging.

3.1 The phonetic models

The models you will use in the exercise were trained on the TIDIGITS database with 13 MFCC feature vectors computed as in Lab 1. The lab package contains two versions of the models that you will have to compare:

- `lab2_models_all.npz` was trained on the full training set,
- `lab2_models_onespkr.npz` was trained on a single speaker, that happens to be female.

Load one of the model files with:

```
>>> phoneHMMs = np.load('lab2_models_onespkr.npz')['phoneHMMs'].item()
```

or

```
>>> phoneHMMs = np.load('lab2_models_all.npz')['phoneHMMs'].item()
```

The resulting variable `phoneHMMs` is a dictionary with 21 keys each corresponding to a phonetic model. You can list the model names with:

```
>>> list(sorted(phoneHMMs.keys()))
['ah', 'ao', 'ay', 'eh', 'ey', 'f', 'ih', 'iy', 'k', 'n', 'ow', 'r', 's',
 'sil', 'sp', 't', 'th', 'uw', 'v', 'w', 'z']
```

Special cases are `sil` and `sp` that model silence and short pauses. For this exercise you can ignore the `sp` model, that will become important only in Lab 3. Note that the list is a subset of the phonemes used in the English language limited to the pronunciation of the 11 digits.

Each model is an HMM with a single Gaussian emission probability distribution per state and diagonal covariance matrices stored as vectors.

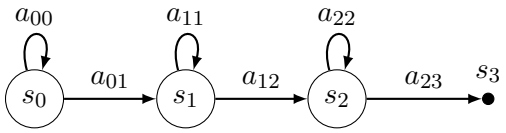
Each model contains the following keys:

```
>>> phoneHMMs['ah'].keys()
['name', 'startprob', 'transmat', 'means', 'covars']
```

key	symbol	description
`name`		phonetic symbol, `sil` or `sp`
`startprob`	$\pi_i = P(z_0 = s_i)$	probability to start in state i
`transmat`:	$a_{ij} = P(z_n = s_j \| z_{n-1} = s_i)$	transition probability from state i to j
`means`:	μ_{id}	array of mean vectors (rows correspond to different states)
`covars`:	σ^2_{id}	array of variance vectors (rows correspond to different states)

If you ignore the `sp` model, all models have three emitting states. Consequently, the `means` and `covars` arrays will be both 3×13 in size. Note, however, that both the `startprob` and `transmat` arrays have sizes corresponding to four states (`startprob` is length 4 and `transmat` is 4×4). The reason for this is that we will concatenate these models to form word models, and we therefore want to be able to express the probability a_{i3} to leave the model from state s_i.

In the left-to-right models you will use in this exercise the model can only be left from the last (third) emitting state in each phonetic model s_2 (with probability a_{23}). This is illustrated in the following figure and will be clearer in Section 3.2.



Note that the last state s_3 is non-emitting, meaning that it does not have any state to output probability distribution associated with it.

3.2 Pronunciation dictionary and utterance models

The mapping between digits (words) and phonetic models can be obtained with the pronunciation dictionary that you can find in `prondict.py`, and that looks like this:

```
prondict = {}
prondict['o'] = ['ow']
prondict['z'] = ['z', 'iy', 'r', 'ow']
prondict['1'] = ['w', 'ah', 'n']
...
```

Because we are working with recordings of isolated digits, a model of each utterance should also contain initial and final silence:

```
>>> isolated = {}
>>> for digit in prondict.keys():
>>>    isolated[digit] = ['sil'] + prondict[digit] + ['sil']
```

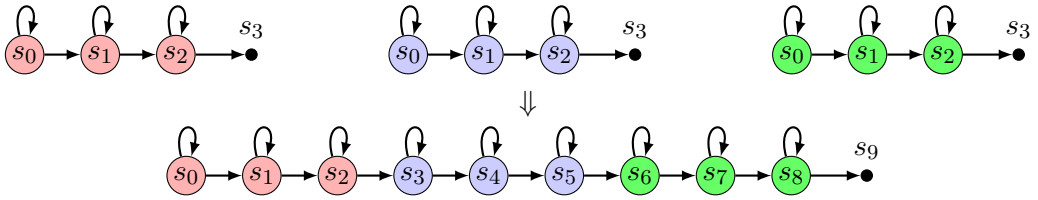
4 Concatenating HMMs

Write the `concatTwoHMMs` function in `lab2_proto.py` that takes two HMMs in input and returns an HMM that is the concatenation of the two. Refer to the included `concatenating_hmms.pdf` document for a detailed description on how to combine the state priors, transition matrices and emission probability distributions from each model.

Then use the already defined `concatHMMs` in `lab2_proto.py` to concatenate any number of HMMs, given a set of HMM models and a list of model names. For example:

```
>>> wordHMMs = {}
>>> wordHMMs['o'] = concatHMMs(phoneHMMs, isolated['o'])
```

In this case, `isolated['o']=['sil', 'ow', 'sil]`, and the resulting word model will be a concatenation of three models with three emitting states each as illustrated below:



Remember that each model in `phoneHMMs` has an extra state to simplify the concatenation. All the extra states but the last are dropped when doing the concatenation.

4.1 The example

Load the example file with:

```
example = np.load('lab2_example.npz')['example'].item()
```

This is a dictionary containing all the fields as in the `data` array, plus the following additional fields:

```
list(example.keys())
[..., 'obsloglik', 'logalpha', 'loglik', 'vloglik', 'loggamma', 'logxi']
```

Here is a description of each field. You will see how to use this information in the reminder of these instructions. All the probabilities described below were obtained using the HMM model `wordHMMs['o']` (that is, the models for digit `'o'` with phonetic models taken from `lab2_models_onespkr.npz`) on the sequence of MFCC vectors in `example['lmfcc']`. Also with `n_states` we intend the number of emitting states in the model, that is the length of `startprob` minus one:

key	idxs	symbol	description
`obsloglik`	`[i,j]`	$\log \phi_j(x_i)$	observation log likelihood for each Gaussian in `wordHMMs['o']`, shape: `(n_timesteps, n_states)`
`logalpha`	`[i,j]`	$\log \alpha_i(j)$	alpha log probabilities, see definition later, shape: `(n_timesteps, n_states)`
`logbeta`	`[i,j]`	$\log \beta_i(j)$	beta log probabilities, see definition later, shape: `(n_timesteps, n_states)`
`loglik`	-	$\log P(X\|\theta_{\text{HMM}})$	log likelihood of the observations sequence X given the HMM model, scalar
`vloglik`	-	$\log P(X, S_{\text{opt}}\|\theta_{\text{HMM}})$	Viterbi log likelihood of the observations sequence X and the best path given the HMM model, scalar
`loggamma`	`[i,j]`	$\log \gamma_i(j)$	gamma log probabilities, see definition later, shape: `(n_timesteps, n_states)`
`logxi`	`[i,j]`	$\log \xi_i(j)$	xi log probabilities, see definition later, shape: `(n_timesteps, n_states)`

Figure 1 shows some of the relevant fields in `example`.

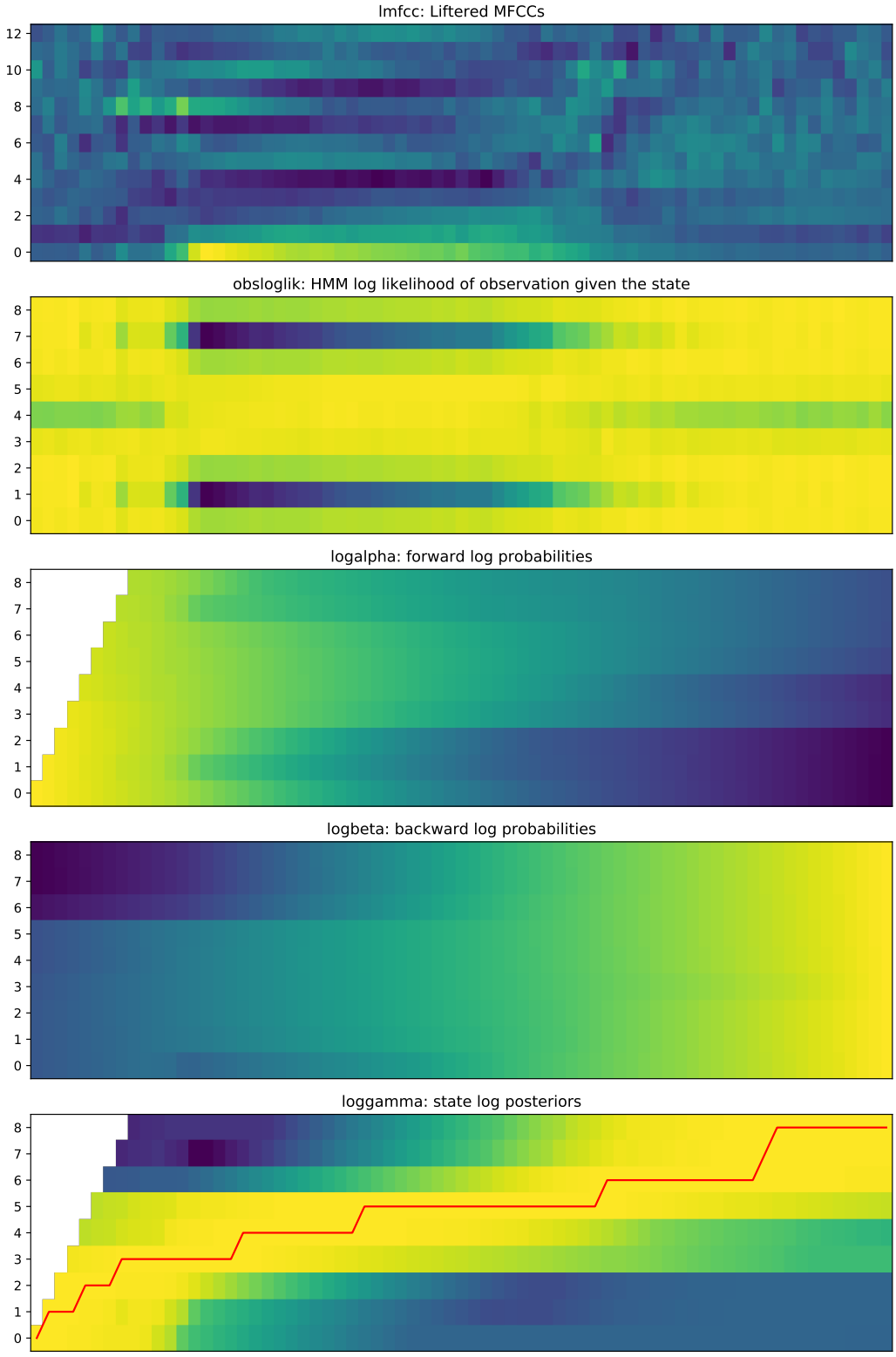


Figure 1. Step-by-step probability calculations for the utterance in the example. The utterance contains the digit "oh" spoken by a female speaker. The model parameters are obtained by concatenating the HMM models for `sil`, `ow` and `sil`.

5 HMM Likelihood and Recognition

5.1 Gaussian emission probabilities

The function `log_multivariate_normal_density_diag()` from `lab2_tools.py` can be used to compute

$$\text{obsloglik}[i,j] = \log \phi_j(x_i) = \log N(x_i, \mu_j, \Sigma_j) = \log P(x_i|\mu_j, \Sigma_j),$$

that is the log likelihood for each observation x_i and each term in a multivariate Gaussian density function with means μ_j and diagonal covariance matrices Σ_j. In the case of Gaussian HMMs, ϕ_j corresponds to the emission probability model for a single state j.

Verify that you get the same results as in `example['obsloglik']` when you apply the `log_multivariate_normal_density_diag` function to the `example['lmfcc']` data using the Gaussian distributions defined by the `wordHMMs['o']` that you have created with your `concatHMMs` function. Note that in these cases we are not using the time dependency structure of the HMM, but only the Gaussian distributions.

Plot the log likelihood for Gaussians from HMMs models corresponding to the same digit on a test utterance of your choice. What can you say about the figure? Which Gaussian components give the high likelihood in which time steps? Why? Remember that each utterance starts and ends with silence.

5.2 Forward Algorithm

Write the function `forward` following the prototypes in `lab2_proto.py`. The function should take as input a lattice of emission probabilities as in the previous section, that is an array containing:

$$\text{obsloglik}[i,j] \quad = \quad \log \phi_j(x_i)$$

and the model parameters a_{ij} and π_i. When you compute the log of a_{ij} and π_i, some of the values will become negative infinity ($\log(0)$). This should not be a problem, because all the formulas should remain consistent and you should only get a warning from `numpy`[1].

The output is the array:

$$\text{logalpha}[i,j] \quad = \quad \log \alpha_i(j)$$

where i corresponds to time steps and j corresponds to states in the model.

Remember that the forward probability is defined as:

$$\alpha_n(j) \quad = \quad P(x_0, \ldots, x_n, z_n = s_j|\theta), \tag{1}$$

where $\theta = \{\Pi, A, \Phi\}$ are the model parameters. There is a recursion formula (see Appendix A) to obtain $\alpha_n(j)$ from $\alpha_{n-1}(i)$ for all the previous states. In the recursion formulae for the forward (and backward) probabilities there is an expression involving the log of a sum of exponents ($\log \sum \exp(.)$). Make use of the `logsumexp` function from `lab2_tools.py` to calculate those cases.

Apply your function to the `example['obsloglik']` utterance using the model parameters in your `wordHMMs['o']` model and verify that you obtain the same $\log \alpha$ array as in `example['logalpha']`.

Remembering the definition of α in Eqn. 1, derive the formula to compute the likelihood $P(X|\theta)$ of the whole sequence $X = \{x_0, \ldots, x_{N-1}\}$, given the model parameters θ in terms of $\alpha_n(j)$.

[1] An alternative would be to use the function `numpy.ma.log()` which will mask log(0) elements, but at the time of writing I could not get this to work properly, so I do not recommend it at this point.

Hint: if the events $B_j, j \in [0, M)$ form a disjoint partition of the sure event, that is:

$$\begin{aligned} P(B_i, B_j) &= 0, \quad \forall i, j, \quad i \neq j, \quad \text{and} \\ \sum_{j=0}^{M-1} P(B_j) &= 1, \end{aligned}$$

then it is true that $P(A) = \sum_{j=0}^{M-1} P(A, B_j)$, that is, we can *marginalize out* the variable B_j.

Convert the formula you have derived into log domain. Verify that the log likelihood obtained this way using the model parameters in `wordHMMs['o']` and the observation sequence in `example['lmfcc']` is the same as `example['loglik']`.

Using your formula, score all the 44 utterances in the `data` array with each of the 11 HMM models in `wordHMMs`. Do you see any mistakes if you take the maximum likelihood model as winner? Compare the results obtained from the models trained on a single speaker or all the training speakers.

5.3 Viterbi Approximation

Here you will compute the log likelihood of the observation X given a HMM model and the best sequence of states. This is called the Viterbi approximation. Implement the function `viterbi` in `lab2_proto.py`.

In order to recover the best path, you will also need an array storing the best previous path for each time step and state (this needs to be defined only for $n \in [1, N)$, that is not for the first time step):

$$B_n(j) = \arg\max_{i=0}^{M-1} \left(\log V_{n-1}(i) + \log a_{ij} \right)$$

Consult the course book [1], Section 8.2.3, to see the details of the implementation (note that the book uses indices from 1, instead of 0).

Compute the Viterbi log likelihood for `wordHMMs['o']` and `example['lmfcc']`, and verify that you obtain the same result as in `example['vloglik']`.

Plot the alpha array that you obtained in the previous Section and overlay the best path obtained by Viterbi decoding. Can you explain the reason why the path looks this way?

Using the Viterbi algorithm, score all the 44 utterances in the `data` with each of the 11 HMM models in `wordHMMs`. How many mistakes can you count if you take as winner the model with the maximum viterbi score? Are these the same mistakes obtained in previous section? Can you say something about the complexity of the viterbi scoring compared to the forward scoring (for example by measuring the CPU usage?)

5.4 Backward Algorithm

Write the function `backward` following the prototypes in `lab2_proto.py`. Similarly to the function `forward` in the previous section, the function should take as input a lattice of emission probabilities as in the previous section, that is an array containing:

$$\text{obsloglik}[i, j] = \log \phi_j(x_i)$$

The output is the array:

$$\text{logbeta}[i,j] \quad = \quad \log \beta_i(j),$$

where i corresponds to time steps and j corresponds to states in the model. See Appendix A for the recursion formulae.

In all the cases where there is an expression involving the log of a sum of exponents ($\log \sum \exp(.)$), make use of the `logsumexp` function in `lab2_tools.py`.

Apply your function to the `example['lmfcc']` utterance using the model parameters in `wordHMMs['o']` and verify that you obtain the $\log \beta$ arrays as in `example['logbeta']`.

The definitions of $\beta_n(j)$ in terms of probabilities of events are defined below (where $\theta = \{\Pi, A, \Phi\}$ are the model parameters):

$$\beta_n(i) \quad = \quad P(x_{n+1}, \ldots, x_{N-1} | z_n = s_i, \theta)$$

Optional: Derive the formula that computes $P(X|\theta)$ using the betas $\beta_n(i)$ instead of the alphas.

Hint 1: only the $\beta_0(i)$ are needed for the calculation.

Hint 2: note that the definition of $\alpha_n(j)$ is a *joint* probability of observations *and* state at time step n whereas $\beta_n(i)$ is a *conditional* probability of the observations *given* the state at time step n.

Hint 3: the calculation will not only involve the betas, but also some of the observation likelihoods $\phi_j(x_n)$ and some of the model parameters, π_i or a_{ij}.

Verify that also this method gives you the same log likelihood and that they both are equal to `example['loglik']`.

6 HMM Retraining (emission probability distributions)

6.1 State posterior probabilities

Implement the `statePosteriors` function in `lab2_proto.py` that calculates the state posteriors $\gamma_n(i) = P(z_n = s_i | X, \theta)$ given the observation sequence, also called gamma probabilities. See Appendix A for the formulas in log domain. Calculate the state posteriors for the utterance in the example. Verify that for each time step the state posteriors sum to one (in linear domain).

Consider the emission probabilities in the model as a mixture of Gaussians with equal a priori probabilities of each term (and no transition model). Using this GMM model, compute the state posteriors $\gamma_n^{\text{GMM}}(i) = P(z_n = s_i | x_n, \phi)$ for each time step in example utterance. Compare the HMM and GMM posteriors. What is the difference? Why?

Now sum the HMM posteriors (in linear domain) for each state along the time axis. What is the meaning the the values you obtain? What about summing over both states and time steps? Compare this number to the length of the observation sequence.

6.2 Retraining the emission probability distributions

Write the function `updateMeanAndVar` in `lab2_proto.py` that, given a sequence of feature vectors and the array of state posteriors probabilities, estimates the new mean and variance vectors for each state. Note that this function has an input argument called `varianceFloor` with default

value 5.0. The reason is that, the more we tend to maximise the likelihood, the narrower the Gaussian distributions will become (variance that tends to zero), especially if a Gaussian component is associated with very few data points. To prevent this, after the update, you should substitute any value of the variances that falls below `varianceFloor`, with the value of `varianceFloor`. In theory the variance floor could be different for each element in the feature vector. In this exercise, for simplicity we use a single variance floor.

Consider the utterance in `data[10]` containing the digit "four" spoken by a male speaker. Consider also the model in `wordHMMs['4']`, with the model parameters for the same digit trained on utterances from all the training speakers. First of all estimate the log likelihood of the data given the current model ($\log P(X|\theta)$), where the model parameters are, as usual:

- π_i a priori probability of state s_i,
- a_{ij} transition probabilities from state s_i to state s_j,
- μ_{ik} Gaussian mean parameter for state s_j and feature component k
- σ^2_{ik} Gaussian variance parameter for state s_j and feature component k

Keeping π_j and a_{ij} constant, repeat the following steps until termination (the increase in log likelihood falls below a threshold):

1. Expectation: compute the alpha, beta and gamma probabilities for the utterance, given the current model parameters (using your functions `forward()`, `backward()` and `statePosteriors()`)
2. Maximization: update the means μ_{jk} and variances σ^2_{ik} given the sequence of feature vectors and the gamma probabilities (using `updateMeanAndVar()`)
3. estimate the likelihood of the data given the new model, if the likelihood has increased, go back to 1

You can use a max number of iterations, for example 20, and a threshold of 1.0 on the increase in log likelihood, as stopping criterion.

Repeat the same procedure on the same utterance `data[10]`, but starting from different models in `wordHMMs`. Can you say anything about the log likelihood at each iteration and the number of iterations that are required to terminate the algorithm?

A Recursion Formulae in Log Domain

A.1 Forward probability

The recursion formulae for the forward probabilities in *log domain* are given here, where we have used Python convention with array indices going from 0 to len-1:

$$\begin{aligned}\log \alpha_0(j) &= \log \pi_j + \log \phi_j(x_0) \\ \log \alpha_n(j) &= \log \left(\sum_{i=0}^{M-1} \exp \Big(\log \alpha_{n-1}(i) + \log a_{ij} \Big) \right) + \log \phi_j(x_n)\end{aligned}$$

A.2 Backward probability

The recursion formulae for the backward probabilities in *log domain* are given here, where we have used Python convention with array indices going from 0 to len-1:

$$\begin{aligned}\log \beta_{N-1}(i) &= 0 \\ \log \beta_n(i) &= \log \left(\sum_{j=0}^{M-1} \exp \Big(\log a_{ij} + \log \phi_j(x_{n+1}) + \log \beta_{n+1}(j) \Big) \right)\end{aligned}$$

Note also that the initialization of the $\beta_{N-1}(i)$ is different from the one defined in the course book [1] (Section 8.2.4) but corresponds to the one used in [2].

A.3 Viterbi approximation

The Viterbi recursion formulas are as follows:

$$\begin{aligned}\log V_0(j) &= \log \pi_j + \log \phi_j(x_0) \\ \log V_n(j) &= \max_{i=0}^{M-1} \Big(\log V_{n-1}(i) + \log a_{ij} \Big) + \log \phi_j(x_n)\end{aligned}$$

A.4 State posteriors (gamma)

The gamma probability is the posterior of the state given the whole observation sequence and the model parameters: $\gamma_n(i) = P(z_n = s_i | X, \theta)$ This can be easily computed from the forward and backward probabilities. In log domain:

$$\log \gamma_n(i) = \log \alpha_n(i) + \log \beta_n(i) - \log \sum_i \exp \left(\log \alpha_{N-1}(i) \right)$$

where we have used the Python convention of starting indices from 0.

A.5 Pair of states posteriors (xi, not used in this exercise)

The xi probabilities are the posteriors of being in a two subsequent states given the whole observation sequence and the model parameters: $\xi_n(i,j) = P(z_n = s_j, z_{n-1} = s_i | X, \theta)$. In order to compute them you will need the forward and backward probabilities, but also the emission probabilities $\phi_j(x_n)$ for each state and feature vector and the state transition probabilities a_{ij}. In log domain:

$$\log \xi_n(i,j) = \log \alpha_{n-1}(i) + \log a_{ij} + \log \phi_j(x_n) + \log \beta_n(j) - \log \sum_i \exp \left(\log \alpha_{N-1}(i) \right)$$

Note that you can only compute this from the second time step ($n = 1$) because you need the alpha probabilities for the previous time step.

References

[1] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice Hall PTR, 2001.

[2] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.