KTH Computer Science
and Communication

DT2118 Lab2: Hidden Markov Models with Gaussian Emissions

1 Objective

The objective is to implement the algorithms for evaluation and decoding of Hidden Markov Models (HMMs) combined with Gaussian emission probability distributions. The lab is designed in Python, but the same functions can be obtained in Matlab/Octave or using the Hidden Markov Toolkit (HTK).

2 Task

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

- implement the *forward* algorithm,
- use it compute the log likelihood of spoken utterances given a Gaussian HMM
- perform isolated word recognition
- compare the Gaussian HMM to a GMM (Gaussian Mixture Model)
- implement the *Viterbi algorithm*, and use it to compute Viterbi path and likelihood

In order to pass the lab, you will need to follow the steps described in this document, and produce a report where you describe your work and answer the questions asked here. The report should be submitted in the Assignment section in the course Web on KTH Social `https://www.kth.se/social/course/DT2118/`. One submission should be done for each group, clearly stating all the members of that group.

3 Data

The speech data used in this lab is the same as in Lab 1. This lab assumes that you have loaded the `tidigits` array containing speech utterances, and that you have computed MFCC features on each utterance. Refer to the instructions of Lab 1 to know how to load and process the data.

Additionally, the `lab2_models.npz` file contains 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 models

Load the model file with:

```
import numpy as np
models = np.load('lab2_models.npz')['models']
```

The `models` array has length 11 and contains for each element the following keys:

```
models[0].keys()
['digit', 'pron', 'hmm', 'gmm']
```

key	description
`digit`:	the digit the model refers to
`pron`:	pronunciation of the digit in terms of phonemes (not needed in this exercise)
`hmm`:	model parameters of a Gaussian HMM trained with Baum-Welch
`gmm`:	model parameters of a GMM trained with EM

Both the HMMs and the GMMs were trained on the same data (different from the data in the `tidigits` array) and were initialized the same way. Both use diagonal covariance matrices for the Gaussian distributions. The size of the model (number of states in HMM and number of Gaussians in GMM) depends on the digit and was obtained using three states for each phoneme in `models[i]['pron']` plus three states for the initial and final silence in the utterance. The HMM model contains the following fields:

```
models[0]['hmm'].keys()
['startprob', 'transmat', 'means', 'covars']
```

key	symbol	description
`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)

The GMM models contain the following fields:

```
models[0]['gmm'].keys()
['weights', 'means', 'covars']
```

key	symbol	description
`weights`:	w_i	weight of each Gaussian in the mixture
`means`:	μ_{id}	array of mean vectors (rows correspond to different Gaussians)
`covars`:	σ^2_{id}	array of variance vectors (rows correspond to different Gaussians)

3.2 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 Lab 1, plus the following additional fields:

```
example.keys()
[..., 'gmm_obsloglik', 'gmm_loglik', 'hmm_obsloglik', 'hmm_logalpha',
 'hmm_loglik', 'hmm_vloglik']
```

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 by using the HMM model in `models[0]['hmm']` and the GMM model in `models[0]['gmm']` on the sequence of MFCC vectors in `example['mfcc']`:

key	idxs	symbol	description
`gmm_obsloglik`	`[i,j]`	$\log \phi_j(x_i)$	observation log likelihood for each Gaussian in `models[0][gmm]`, shape: (n_timesteps, n_gaussians)
`gmm_loglik`	-	$\log P(X\|\theta_{\text{GMM}})$	log likelihood of the observations sequence X given the full GMM model, scalar
`hmm_obsloglik`	`[i,j]`	$\log \phi_j(x_i)$	observation log likelihood for each Gaussians in `models[0][hmm]`, shape: (n_timesteps, n_states)
`hmm_logalpha`	`[i,j]`	$\log \alpha_j(x_i)$	alpha log probabilities, see definition later, shape: (n_timesteps, n_states)
`hmm_loglik`	-	$\log P(X\|\theta_{\text{HMM}})$	log likelihood of the observations sequence X given the HMM model, scalar
`hmm_vloglik`	-	$\log P(X\|\theta_{\text{HMM}}, S_{\text{opt}})$	Viterbi log likelihood of the observations sequence X given the HMM model and the best path, scalar

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

4 Multivariate Gaussian Density

The function `log_multivariate_normal_density_diag` in `tools2.py` can be used to compute

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

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. In case of a GMM, ϕ_j corresponds to a single Gaussian in the model, without considering the weights.

Verify that you get the same results as in `example['hmm_obsloglik']` and `example['gmm_obsloglik']` when you apply the `log_multivariate_normal_density_diag` function to the `example['mfcc']` data using the Gaussian distributions defined respectively in `models[0]['hmm']` and `models[0]['gmm']`. Note that in these cases we are using neither the weights of the GMM, nor the time dependency structure of the HMM, but only the Gaussian distributions.

5 GMM Likelihood and Recognition

Based on the output of the function in the previous section, write the `gmmloglik` function (`proto2.py`) that computes the log likelihood of an observation sequence $X = \{x_0, \dots, x_{N-1}\}$ given the full GMM model, that is, given the weights of each Gaussian density in the GMM model. Remember that the assumption in the GMM model is that the x_n vectors are independent for

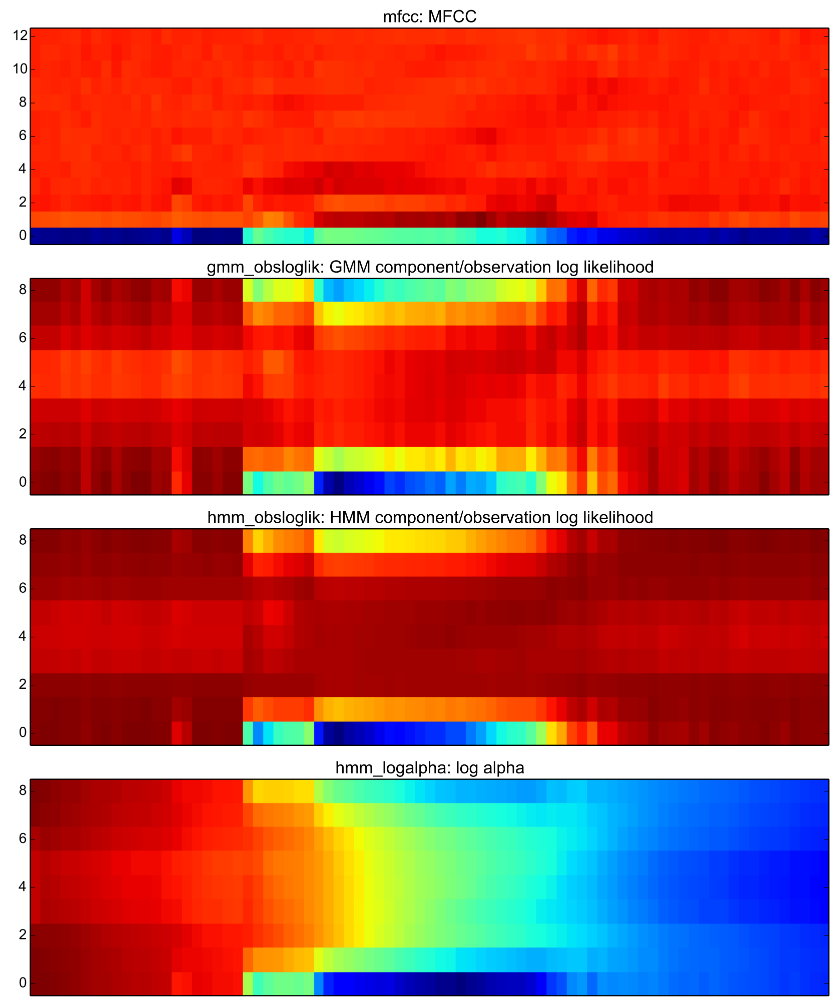
mfcc: MFCC
12
10
8
6
4
2
0
gmm_obsloglik: GMM component/observation log likelihood
8
6
4
2
0
hmm_obsloglik: HMM component/observation log likelihood
8
6
4
2
0
hmm_logalpha: log alpha
8
6
4
2
0

Figure 1.

each $n = [0, N)$[1]. Whenever there is an expression involving the log of a sum of exponents ($\log \sum \exp(.)$), make use of the `logsumexp` function in `tools2.py`.

The output of the `gmmloglik` function, using the model parameters in `models[0]['gmm']` and the observation sequence from `example['mfcc']`, should correspond to `example['gmm_loglik']`.

Use this function to score each of the 44 utterances in the `tidigits` array with each of the 11 GMM models in `models[i]['gmm']`. If we select for each utterance the model with the highest log likelihood, how many digits are misrecognized?

6 HMM Likelihood and Recognition

6.1 Forward Algorithm

Write the function `forward` following the prototypes in `proto2.py`. The function should take as input a lattice of emission probabilities as in the previous section, that is an array containing:

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

The output is the array:

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

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

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

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 `tools2.py`.

Apply your function to the `example['mfcc']` utterance using the model parameters in `models[0]['hmm']` and verify that you obtain the same $\log \alpha$ array as in `example['hmm_logalpha']`.

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

$$\alpha_n(j) = P(x_0, \dots, x_n, z_n = s_j | \theta)$$

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

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

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

[1]We are using the convention to express intervals with square brackets "[" if the extreme is included and parentheses ")" if it is excluded. This corresponds to the `range` function in Python that includes the start value but not the end value.

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 `models[0]['hmm']` and the observation sequence in `example['mfcc']` is the same as `example['hmm_loglik']`.

Using your formula, score all the 44 utterances in `tidigits` with each of the 11 HMM models in `models`. How many mistakes can you count if you take the maximum likelihood model as winner? Compare these results with the GMM results obtained in the previous session.

Repeat the recognition using the Gaussian distributions in the HMM models as if they were a GMM model with equal weights for each Gaussian. What can you say about the results you obtain this way? Can you say something about the influence of the HMM transition model in this particular task?

6.2 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 `proto2.py`. The Viterbi recursion formulas are as follows:

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

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):

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

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 `models[0]['hmm']` and `example['mfcc']`, and verify that you obtain the same result as in `example['hmm_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 `tidigits` with each of the 11 HMM models in `models`. How many mistakes can you count if you take the maximum likelihood model as winner? Compare these results with the results obtained in previous sections.

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.