

Digital Speech Processing

Homework #1

Implementing Discrete Hidden Markov Model

林子權 (modified from 馮子軒 2021DSP ver.)

October 05, 2022

Due on 23:59, November 04, 2022

Outline

1. HMM in Speech Recognition
2. Homework of HMM
 - 2.1 Training
 - 2.2 Testing
3. Requirements
 - 3.1 File Format
 - 3.2 Submission Requirement
4. Grading
5. Contact TAs

HMM in Speech Recognition

In acoustic model

- each word consists of syllables
- each syllable consists of phonemes
- each phoneme consists of some (hypothetical) states

“語音” → “語 (ㄌㄩˇ) 音 (ㄣ)” → “ㄌ” → $\{s_1, s_2, \dots\}$

Each phoneme can be described by a HMM (acoustic model). Given a sequence of observation (MFCC vectors), each of them can be mapped to a corresponding state.

Hence, there are

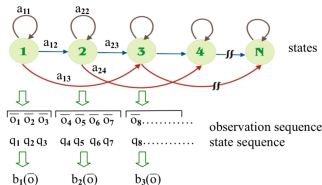
- state transition probabilities (a_{ij}) and
- observation distribution ($b_j[o_t]$)

in each phoneme acoustic model (HMM).

Usually in speech recognition we restrict the HMM to be a *left-to-right model*, and the observation distribution is assumed to be a continuous Gaussian mixture model.

- Left-to-right
- The observation distribution is a continuous Gaussian mixture model.

Hidden Markov Models (HMM)



Formulation

$\bar{o}_t = [x_1, x_2, \dots, x_D]^T$ feature vectors for a frame at time t

$q_t \in \{1, 2, 3, \dots, N\}$ state number for feature vector \bar{o}_t

$A = [a_{ij}]$, $a_{ij} = \text{Prob}[q_t = j \mid q_{t-1} = i]$
state transition probability

$B = [b_j(\bar{o}), j = 1, 2, \dots, N]$ observation (emission) probability

$b_j(\bar{o}) = \sum_{k=1}^M c_{jk} b_{jk}(\bar{o})$ Gaussian Mixture Model (GMM)

$b_{jk}(\bar{o})$: multi-variate Gaussian distribution
for the k -th mixture (Gaussian) of the j -th state

M : total number of mixtures

$\sum_{k=1}^M c_{jk} = 1$

$\pi = [\pi_1, \pi_2, \dots, \pi_N]$ initial probabilities

$\pi_i = \text{Prob}[q_1 = i]$

HMM: $(A, B, \pi) = \lambda$

Figure 1: HMM from lecture 2.0

$$a_{ij} = P(q_{t+1} = j \mid q_t = i), \forall t, i, j \quad (1)$$

$$b_j(A) = P(o_t = A \mid q_t = j), \forall t, A, j \quad (2)$$

Given q_t , the probability distributions of q_{t+1} and o_t are completely determined. (independent of other states or observation)

HW1 v.s. Speech Recognition

	<i>Homework</i>	<i>Speech Recognition</i>
λ set	5 models	initial-final
λ	model_01-05	"□"
$\{o_t\}$	A, B, C, D, E, F	39-dim MFCC
unit	an alphabet	a time frame
observation	sequence	voice wave

Homework of HMM

Flowchart

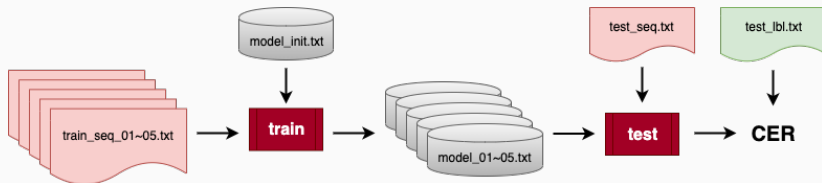


Figure 2: Training and testing models

Problems of HMM

Training

- Basic problem 3 in lecture 4.0
 - Given observations O and an initial model $\lambda = (A, B, \pi)$, adjust λ to maximize $P(O \mid \lambda)$.

$$A_{ij} = a_{ij}, B_{jt} = b_j[o_t], \pi_i = P(q_1 = i)$$

- Baum-Welch algorithm

Testing

- Basic problem 2 in lecture 4.0
 - Given λ and O , find the best state sequences to maximize $P(O \mid \lambda, q)$.
- Viterbi algorithm

Homework of HMM

Training

- Basic problem 3
- *Baum-Welch algorithm*: A generalized expectation-maximization (EM) algorithm¹
 - Calculate α (forward probabilities) and β (backward probabilities) given the observations
 - Find temporary variables ϵ and γ from α and β
 - Update model parameters $\lambda' = (A', B', \pi')$

¹http://en.wikipedia.org/wiki/Baum-Welch_algorithm

Forward Procedure

Forward algorithm: define a forward variable $\alpha_t(i)$

$$\alpha_t(i) = P(o_1, o_2, \dots, o_t, q_t = i \mid \lambda) \quad (3)$$

$$= \text{Prob}[\text{observing } o_1, o_2, \dots, o_t, \text{ state } i \text{ at time } t \mid \lambda] \quad (4)$$

Initialization

$$\alpha_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N \quad (5)$$

Induction

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^N \alpha_t(i) a_{ij} \right] \cdot b_j(o_{t+1}),$$
$$1 \leq t \leq T-1, \quad 1 \leq j \leq N \quad (6)$$

Termination

$$P(\bar{O} \mid \lambda) = \sum_{i=1}^N \alpha_T(i) \quad (7)$$

Backward Procedure

Backward algorithm: define a backward variable $\beta_t(i)$

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \dots, o_T \mid q_t = i, \lambda) \quad (8)$$

$$= \text{Prob}[\text{observing } o_{t+1}, o_{t+2}, \dots, o_T \mid \text{state } i \text{ at time } t, \lambda] \quad (9)$$

Initialization

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \quad (10)$$

Induction

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j),$$
$$t = \{T-1, T-2, \dots, 1\}, \quad 1 \leq i \leq N \quad (11)$$

Calculate γ

Define a temporary variable $\gamma_t(i) = P(q_t = i \mid \bar{O}, \lambda)$

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} = \frac{P(\bar{O}, q_t = i \mid \lambda)}{P(\bar{O} \mid \lambda)} \quad (12)$$

It should be a $T \times N$ matrix!

The probability of transition from state i to state j given observation and model.

$$\epsilon_t(i, j) = P(q_t = i, q_{t+1} = j \mid \bar{O}, \lambda) \quad (13)$$

$$= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)} \quad (14)$$

$$= \frac{\text{Prob}[\bar{O}, q_t = i, q_{t+1} = j \mid \lambda]}{P(\bar{O} \mid \lambda)} \quad (15)$$

In total $T - 1$ matrices (each $N \times N$)

Recall $\gamma_t(i) = P(q_t = i \mid \bar{O}, \lambda)$

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of times that state } i \\ \text{is visited in } \bar{O} \text{ from } t = 1 \text{ to } t = T - 1 \quad (16)$$

$$\sum_{t=1}^{T-1} \epsilon_t(i, j) = \text{expected number of transitions from} \\ \text{state } i \text{ to state } j \text{ in } \bar{O} \quad (17)$$

Re-estimate Model Parameters

Now we could update the parameters with ϵ and γ

$$\lambda' = (A', B', \pi') \quad (18)$$

$$\pi'_i = \gamma_1(i) = p(\text{the first state is } i) \quad (19)$$

$$a'_{ij} = \frac{\sum_{t=1}^{T-1} \epsilon_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} = \frac{\mathbb{E} [\text{Number of transition from } i \text{ to } j]}{\mathbb{E} [\text{Number of visiting state } i]} \quad (20)$$

$$b'_i(k) = \frac{\sum_{O_t=k} \gamma_t(i)}{\sum_{t=1}^T \gamma_t(i)} = \frac{\mathbb{E} [\text{Number of observation } O = k \text{ in state } i]}{\mathbb{E} [\text{Number of visiting state } i]} \quad (21)$$

Re-estimate Model Parameters with Multiple Sequences

With N training sequences, we should update parameters from all those sequences

$$\pi'_i = \frac{\sum_{n=1}^N \gamma_{n,1}(i)}{N} \quad (22)$$

$$a'_{ij} = \frac{\sum_{n=1}^N \sum_{t=1}^{T-1} \epsilon_{n,t}(i,j)}{\sum_{n=1}^N \sum_{t=1}^{T-1} \gamma_{n,t}(i)} \quad (23)$$

$$b'_i(k) = \frac{\sum_{n=1}^N \sum_{O_{n,t}=k} \gamma_{n,t}(i)}{\sum_{n=1}^N \sum_{t=1}^T \gamma_{n,t}(i)} \quad (24)$$

Homework of HMM

Testing

- Basic problem 2
 - Given λ and O , find the best state sequences to maximize $P(O \mid \lambda, q)$.
- Calculate $P(O \mid \lambda) \approx \max P(O \mid \lambda, q)$ for each of the five models
- The model with the highest probability for the most probable path usually also has the highest probability for all possible paths.

Viterbi Algorithm

Complete procedure for Viterbi algorithm²

Initialization

$$\delta_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N \quad (25)$$

Recursion

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \cdot b_j(o_t), 2 \leq t \leq T, 1 \leq j \leq N \quad (26)$$

Termination

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)] \quad (27)$$

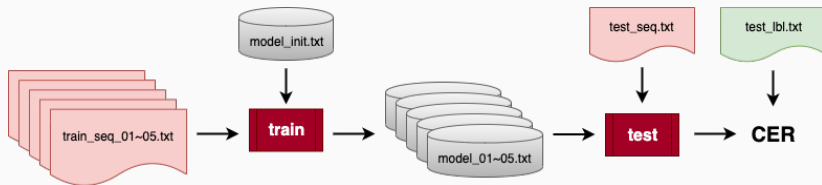
$$\delta_t(i) = \max_{q_1, \dots, q_{t-1}} P[q_1, q_2, \dots, q_{t-1}, q_t = i, o_1, o_2, \dots, o_t \mid \lambda] \quad (28)$$

= the highest probability along a certain single path ending
at state i at time t for the first t observations, given λ (29)

²http://en.wikipedia.org/wiki/Viterbi_algorithm

Requirements

Recall: Flowchart



Provided Files

data/train_seq_0X.txt

- Training data (10K observation sequences)

data/test_lbl.txt

- Testing labels

data/test_seq.txt

- Testing data (2.5K observation sequences)

inc/hmm.h

- Provided by TA, please work with it!
- You can load/dump models with functions within.

model_init.txt

- Initial model parameters

modellist.txt

- Paths to model files

src/test_hmm.c

- A showcase of the usage of *hmm.h*

```
dsp_hw1
├── data
│   ├── test_lbl.txt
│   ├── test_seq.txt
│   ├── train_seq_01.txt
│   ├── train_seq_02.txt
│   ├── train_seq_03.txt
│   ├── train_seq_04.txt
│   └── train_seq_05.txt
├── inc
│   └── hmm.h
├── Makefile
├── model_init.txt
├── modellist.txt
└── src
    └── test_hmm.c
```

Input and Output of Your Program

Training

- Input**
1. number of iterations
 2. initial model (*model_init.txt*)
 3. observation sequences (*train_seq_01~05.txt*)

Output Five files of parameters for 5 models, each contains $\lambda = (A, B, \pi)$ (e.g. *model_01~05.txt*)

Testing

- Input**
1. a file of paths to the models trained in the previous step (*modellist.txt*)
 2. observation sequences (*test_seq.txt*)

Output best answer labels and $P(O | \lambda)$ (e.g. *result.txt*)

Training Details

```
./train <iter> <model_init_path> <seq_path> <output_model_path>
```

<i>iter</i>	# of iterations
<i>model_init_path</i>	path to the initial model params
<i>seq_path</i>	path to sequence data
<i>output_model_path</i>	path to dump trained models

```
./test <models_list_path> <seq_path> <output_result_path>
```

<i>models_list_path</i>	path to the model list file
<i>seq_path</i>	path to sequence data
<i>output_result_path</i>	path to output testing result

Program Execution Example

Compiling

make # type this in the root directory of the project

Training

./train 100 model_init.txt data/train_seq_01.txt model_01.txt

Testing

./test modellist.txt data/test_seq.txt result.txt

Notice!

Command-line arguments are not fixed, read them during runtime.
(e.g. Use **argv** in main function to pass the arguments.)

Also the paths in arguments need to be variable path.

Requirements

File Format

Observation Sequence Format

The given *data/train_seq_01~05.txt* and *data/test_seq.txt* look like this.

1	ACCDDDDFFCCCCBCFFFCCCCCEDADCCAEFCCCACDDFFCCDDFFCCD
2	CABACCAFCCFFCCCDFFCCCCCDFFCDDDDFCDDCCFCCCEFFCCCCBC
3	ABACCCDDCCDDDDFBCCCCDDAACFBCCBCCCCCCCCFFCCCCCDBF
4	AAABBBCCFFBDCDDFFACDCDFCDDFFFFFCDDFFCCCDCCFFFCCCD
5	AACCDCCCCCCCCDCEDCBFFFCD CDCDAFBCDCFFCCDCCCEACDBAFFF
6	...

Each of the former has 10000 sequences and the latter has 2500 sequences.

Model Format 1/2

Please make sure the format is the same as *model_init.txt*, namely, do not swap the row and column.

initial: 6

$$\pi = [\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6]$$

transition: 6³

$$A = \begin{bmatrix} a_{11} & \dots & a_{16} \\ \vdots & \ddots & \vdots \\ a_{61} & \dots & a_{66} \end{bmatrix}$$

observation: 6⁴

$$B = \begin{bmatrix} b_1(o_1) & \dots & b_6(o_1) \\ \vdots & \ddots & \vdots \\ b_1(o_6) & \dots & b_6(o_6) \end{bmatrix}$$

³The sum of row is 1 here.

⁴The sum of column is 1 here.

Model Format 2/2

A model file (e.g. *model_0X.txt*) should look like this and be able to be loaded by 'loadHMM' in *inc/hmm.h*.

```
1  initial: 6
2  0.2      0.1      0.2      0.2      0.2      0.1
3
4  transition: 6
5  0.3      0.3      0.1      0.1      0.1      0.1
6  0.1      0.3      0.3      0.1      0.1      0.1
7  0.1      0.1      0.3      0.3      0.1      0.1
8  0.1      0.1      0.1      0.3      0.3      0.1
9  0.1      0.1      0.1      0.1      0.3      0.3
10 0.3      0.1      0.1      0.1      0.1      0.3
11
12 observation: 6
13 0.2      0.2      0.1      0.1      0.1      0.1
14 0.2      0.2      0.2      0.2      0.1      0.1
15 0.2      0.2      0.2      0.2      0.2      0.2
16 0.2      0.2      0.2      0.2      0.2      0.2
17 0.1      0.1      0.2      0.2      0.2      0.2
18 0.1      0.1      0.1      0.1      0.2      0.2
```

Model List Format

The given *modellist.txt* looks like this.

```
1  model_01.txt
2  model_02.txt
3  model_03.txt
4  model_04.txt
5  model_05.txt
```

Your testing program should be able to read a list like this and load models from the specified paths for testing. (Don't worry! If you use *hmm.h*, all of these are done by calling function *load_models()*. For more details please refer to *hmm.h*.)

Output Format

Your testing program should output these to the specific path (e.g. *result.txt*) given as a command-line argument while executing the program.

```
1 model_01.txt 7.822367e-34
2 model_05.txt 1.094896e-40
3 model_01.txt 7.928724e-33
4 model_02.txt 4.262100e-37
5 model_02.txt 5.914689e-42
6 ...
```

Each line consists of the hypothesis model and its likelihood. They should be separated by a **space**.

The first few lines of the given *data/test_lbl.txt* looks like this.

```
1  model_01.txt
2  model_05.txt
3  model_01.txt
4  model_02.txt
5  model_02.txt
6  ...
```

Makefile Format

The Makefile you submit should be capable to compile your program using *make*. The provided one can compile *train.c* and *test.c* in directory *src* into two executables *train* and *test*.

```
1  .PHONY: all clean run
2  CC=gcc
3  CFLAGS=-std=c99 -O2
4  LDFLAGS=-lm
5  TARGET=train test
6
7  all: $(TARGET)
8
9  train: src/train.c
10     $(CC) -o $@ $^ $(CFLAGS) $(LDFLAGS) -Iinc
11
12  test: src/test.c
13     $(CC) -o $@ $^ $(CFLAGS) $(LDFLAGS) -Iinc
14
15  clean:
16     rm -f $(TARGET)
```

Please write a **one-page** report in **PDF** format, name it *report.pdf* and submit with your source code.

State your name (the name on NTU cool, we have no way know your English name or any nickname), student ID and any challenges you encounter or attempts you try.

If your report contains your name and student ID, you will get all points of report. A good report may grant you bonus of extra 5%.

File Structure

All of your source code files must be placed under `inc/` and `src/`.

Let's say you only have two implementation files and use the functions provided in `hmm.h`. You should put your source code under `src/` and leave `hmm.h` in `inc/`.

```
.
├── inc
│   ├── hmm.h
│   └── [other *.h or *.hpp]
├── Makefile
├── model_init.txt
├── report.pdf
└── src
    ├── test.c
    ├── train.c
    └── [other *.c, *.cc or *.cpp]
```


Requirements

Submission Requirement

Submission Requirement 1/2

1. Create a directory named *hw1_[STUDENT_ID]*⁵.
2. Put
 - *inc/*
 - *Makefile*
 - *model_init.txt*
 - *report.pdf*
 - *src/*into the directory.⁶
3. Compress the directory into a **ZIP** file named *hw1_[STUDENT_ID].zip*.
4. Upload this ZIP file to NTUCOOL.

⁵lowercase

⁶Put every source code file in *inc/* and *src/*.

Submission Requirement 2/2

Let's say your student ID is *r01234567*.

hw1_r01234567.zip

└─ *hw1_r01234567*

└─ *inc*

└─ *└─ [*h or *.hpp]*

└─ *Makefile*

└─ *model_init.txt*

└─ *report.pdf*

└─ *src*

└─ *└─ [*c, *.cc or *.cpp]*

Grading

Your training and testing program will be tested respectively. We will specify **100** as the number of iterations while testing your training program. And each of your program is allowed to run for **1 min**.

Here's TA's environment.

Kernel Linux 5.11.0-46-generic

Processors AMD Ryzen 5 3600 6-Core Processor

RAM 32 GB

GCC Version 9.3.0

Grading Policy

File Format 20%

- ZIP file name
- directory name
- separated header and implementation files
- *Makefile*
- *model_init.txt*

Program 20%

- compiled and executed without error (including the correctness of test)
- output files generated after execution

Report 10%

and bonus of extra 5% for the impressive ones

Accuracy 50%

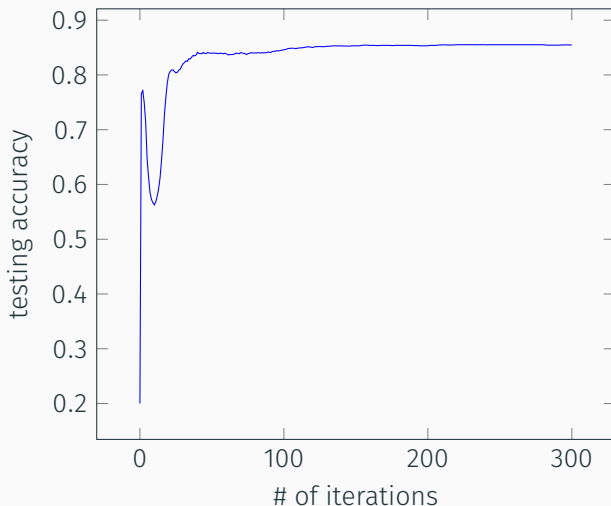
- only for the training part, not for the testing part
- 30% for provided test set, 20% for private test set⁷

⁷details are provided at the next page

Training Accuracy vs Credit

Accuracy	Credit
$80\% \leq \text{accuracy}$	100%
$60\% \leq \text{accuracy} < 80\%$	75%
$40\% \leq \text{accuracy} < 60\%$	50%
$\text{accuracy} < 40\%$	0%

Test Accuracy v.s. # of Training Iterations



This figure is only for demo, and is not from the given public data.

Late Submission

Due on 23:59, November 04, 2022

You are still allowed to submit after the due date. The penalty for late submission is an exponential decay with decay rate 1.5%⁸ of the maximum grade applicable for the assignment, for each hour that the assignment is late.

An assignment submitted more than 3 days after the deadline will have a grade of zero recorded for that assignment.

$$\text{SCORE}_{\text{final}}(\text{hr}) = \begin{cases} \text{SCORE}_{\text{original}} \times 0.985^{\text{hr}} & , \text{hr} \leq 72 \\ 0 & , \text{hr} > 72 \end{cases}$$

⁸less than 70% after 24 hrs, 48% for 48 hrs and 33% for 72 hrs

Please Note...

File Format

- All of your source code files should be placed under *inc/* and *src/*.
- None of your object file, executable files should present.
- *model_init.txt* must be submitted, even if it's not needed for your program.

Program

- Only *C/C++* is allowed.
- Make sure your program can be compiled with the *Makefile* you submit.
- The paths in command-line arguments have to support both absolute and relative path.
- Each of your program is allowed to run for 1 min. (each model have 1 min to train)

Accuracy

- Make sure your training program saves models within time limit.
- Public and private test set may sample from *different* distributions.

If you have any questions, please read the FAQ⁹ first.

⁹<http://speech.ee.ntu.edu.tw/DSP2022Autumn/hw1/FAQ.html>

DO NOT CHEAT

Any form of cheating, lying, or plagiarism will not be tolerated.

- *Discussion is encouraged, while sharing your code is not*
- *A public Github repo is not equal to an open source code*
- *Even is open source, it still couldn't be your homework*
- *The simplest guideline: do homework by yourself*

Please refer to this file for more detail about plagiarism.¹⁰

¹⁰<https://speech.ee.ntu.edu.tw/DSP2022Autumn/Pdfs/Plagiarism.pdf>

Questions?

Contact TAs

If you have any question or need help,

- send email to *ntudsp2022ta@gmail.com*
- and use “[HW1]” as the subject line prefix

Or come to my online TA hour, and don't forget to inform me by email, thanks!

林子權 Mon. 10:00 - 12:00

Office hours