DIGITAL SPEECH PROCESSING HOMEWORK #1

DISCRETE HIDDEN MARKOV MODEL IMPLEMENTATION

Date: March, 28 2018

Revised by Ju-Chieh Chou

Outline

- HMM in Speech Recognition
- Problems of HMM
  - Training
  - Testing
- File Format
- Submit Requirement

HMM IN SPEECH RECOGNITION

Speech Recognition

- In acoustic model,
  - each word consists of syllables
  - each syllable consists of phonemes
  - each phoneme consists of some (hypothetical) states.

“青色” → “青(ㄑㄧㄥ)色(ㄙㄜˋ)” → ”ㄑ” → $\{s_1, s_2, \ldots\}$

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.

Speech Recognition

- Hence, there are state transition probabilities (a_{ij}) and observation distribution ($b_j[o_t]$) in each phoneme acoustic model(HMM).

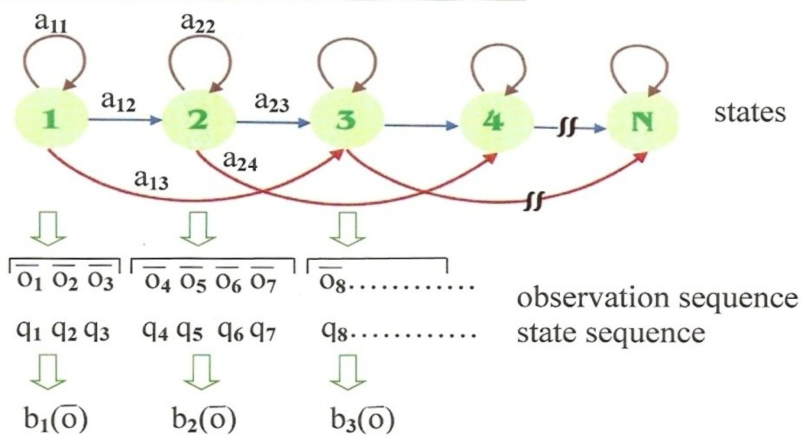
- Usually in speech recognition we restrict the HMM to be a left-to-right model, and the observation distribution are assumed to be a continuous Gaussian mixture model.

Review

- **left-to-right**
- observation distribution are a **continuous** Gaussian mixture model

2.0 Fundamentals of Speech Recognition

Hidden Markov Models (HMM)



- **Formulation**

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

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

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

state transition probability

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

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

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

M : total number of mixtures

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

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

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

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

General Discrete HMM

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

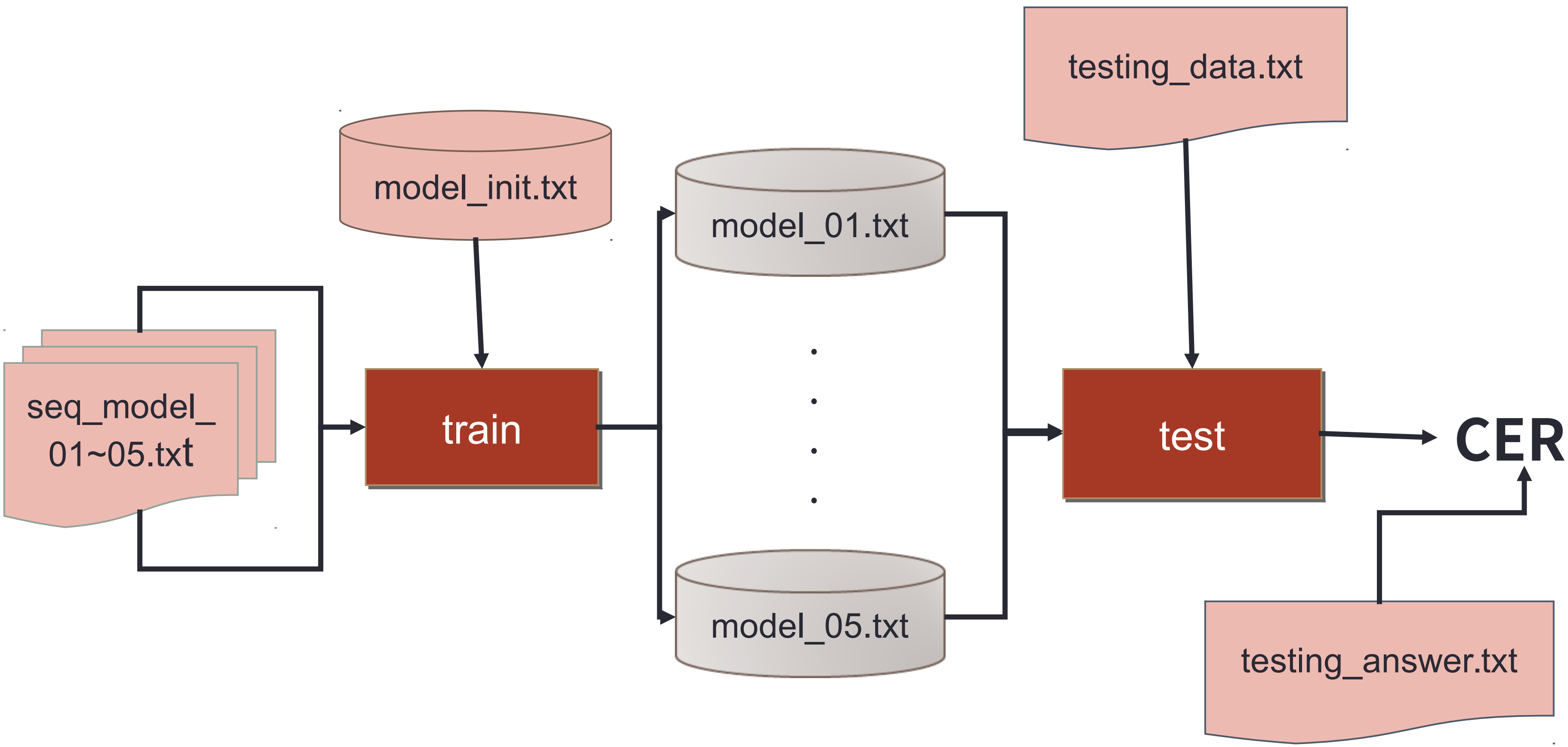
Given q_t , the probability distributions of q_{t+1} and o_t are completely determined.
(independent of other states or observation)

HW1 v.s. Speech Recognition

	Homework #1	Speech Recognition
λ set	5 Models	Initial-Final
λ	model_01~05	“ ㄑ ”
$\{o_t\}$	*A, B, C, D, E, F*	39dim MFCC
unit	an alphabet	a time frame
observation	sequence	voice wave

Homework Of HMM

Flowchart


model_init.txt
seq_model_
01~05.txt
train
model_01.txt
model_05.txt
testing_data.txt
test
CER
testing_answer.txt

Problems of HMM

- **Training**
  - Basic Problem 3 in Lecture 4.0
    - Give O and an initial model $\lambda = (A, B, \pi)$, adjust λ to maximize $P(O|\lambda)$
      $\pi_i = P(q_1 = i)$, $A_{ij} = a_{ij}$, $B_{jt} = b_j [o_t]$
  - Baum-Welch algorithm
- **Testing**
  - Basic Problem 2 in Lecture 4.0
    - Given model λ and O, find the best state sequences to maximize $P(O|\lambda, q)$.
  - Viterbi algorithm

Training

- Basic Problem 3:
  - Give O and an initial model $\lambda = (A, B, \pi)$, adjust λ to maximize $P(O|\lambda)$

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

- Baum-Welch algorithm

- A generalized expectation-maximization (EM) algorithm.

1. Calculate α (forward probabilities) and β (backward probabilities) by the observations.
2. Find ε and γ from α and β
3. Recalculate parameters $\lambda' = (A', B', \pi')$

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

Forward Procedure

- **Forward Procedure(Forward Algorithm): defining a forward variable $\alpha_t(i)$**

$$\alpha_t(i) = P(o_1 o_2 \ldots\ldots o_t, q_t = i|\lambda)$$

$$= \text{Prob}[\text{observing } o_1 o_2 \ldots o_t, \text{ state } i \text{ at time } t| \lambda]$$

  - Initialization

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

  - Induction

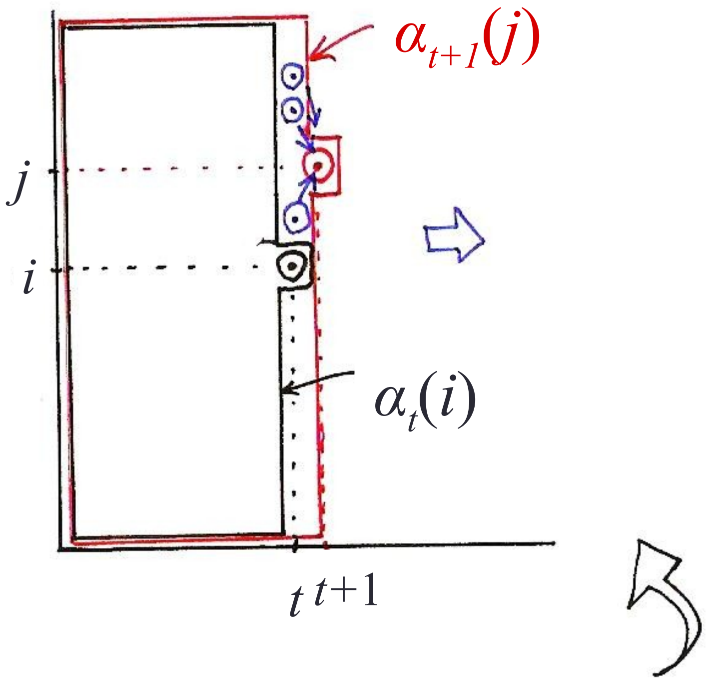
$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}]\, b_j(o_{t+1})$$

$$1 \le t \le T-1$$

$$1 \le j \le N$$

  - Termination

$$P(\overline{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$



Forward Algorithm

Forward Procedure by matrix

- Calculate β by backward procedure is similar.
- **Backward Algorithm :** defining a backward variable $\beta_t(i)$

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

$$= \text{Prob}[\text{observing } o_{t+1}, o_{t+2}, \ldots, o_T \mid \text{state } i \text{ at time } t, \lambda]$$

- Initialization

$$\beta_T(i) = 1, \quad 1 \le i \le N$$

- Induction

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1}) \beta_{t+1}(j)$$

$$t = T-1, T-2, \ldots, 2, 1, \qquad 1 \le i \le N$$

See Fig. 6.6 of Rabiner and Juang

Calculate γ

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

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

N * T matrix

Calculate ε

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

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

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

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

Totally (T-1) N*N matrices.

Accumulate ε and γ

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

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

$$= \text{expected number of transitions from state i in } \overline{O}$$

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

Re-estimate Model Parameters

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

$$\pi_i = \frac{\sum \gamma_1(i)}{N}, \text{where N is number of samples}$$

$$a_{ij} = \frac{\sum \varepsilon(i,j)}{\sum \gamma(i)} = \frac{E[\text{Number of Transition from i to j}]}{E[\text{Number of Visiting state i}]}$$

$$b_i(k) = \frac{\sum_{O=k} \gamma(i)}{\sum \gamma(i)} = \frac{E[\text{Number of Observation } O = k \text{ in state i}]}{E[\text{Number of Visiting state i}]}$$

Accumulate ε and γ through all samples!!
Not just all observations in one sample!!

Testing

- Basic Problem 2:
  - Given model λ and O, find the best state sequences to maximize $P(O|\lambda, q)$.
- Calculate $P(O|\lambda) \doteqdot \max P(O|\lambda, q)$ 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 Algorithm

- Initialization

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

- Recursion

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] \bullet b_j(o_t)$$

$$2 \le t \le T,\quad 1 \le j \le N$$

- Termination

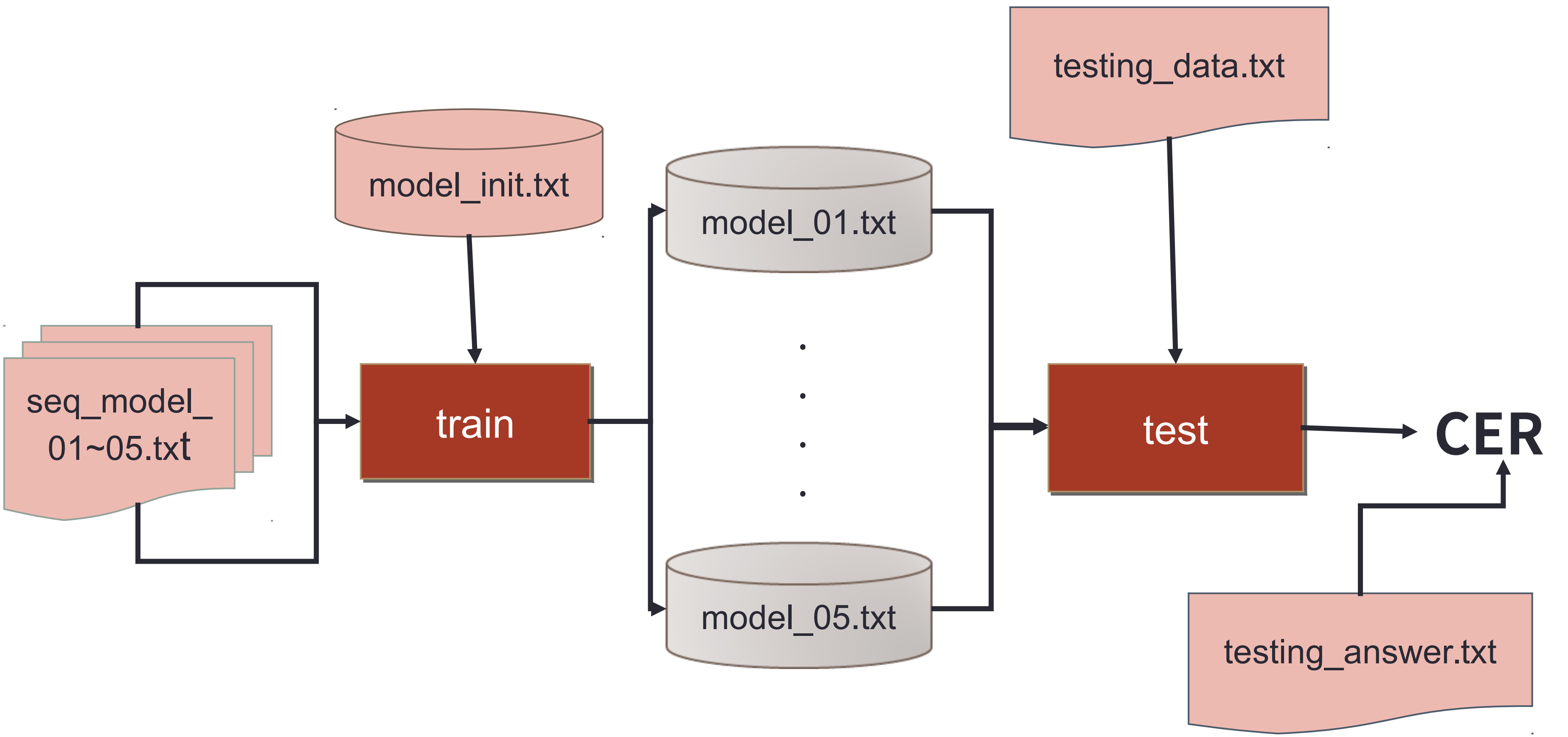
$$P^* = \max_{1 \le i \le N} [\delta_T(i)]$$

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

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

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

Flowchart


testing_data.txt
model_init.txt
model_01.txt
seq_model_
01~05.txt
train
test
CER
model_05.txt
testing_answer.txt

FILE FORMAT

test_hmm.c

- An example of using hmm.h and Makefile(a script to compile your program).
- Type ”make” to compile, type ”make clean” to remove excutable.
- Please use provided hmm.h.
- If C++11 is used, add the flag -std=c++11 in your makefile.

test_hmm.c

```
r98922053@linux12:~/hw1 $ cd c_cpp/
r98922053@linux12:~/hw1/c_cpp $ ls
hmm.h  Makefile  model_init.txt  modellist.txt  test_hmm.c
r98922053@linux12:~/hw1/c_cpp $ make
cc   -lm       test_hmm.c   -o test_hmm
r98922053@linux12:~/hw1/c_cpp $ ./test_hmm
initial: 6
0.20000 0.10000 0.20000 0.20000 0.20000 0.10000

transition: 6
0.30000 0.30000 0.10000 0.10000 0.10000 0.10000
0.10000 0.30000 0.30000 0.10000 0.10000 0.10000
0.10000 0.10000 0.30000 0.30000 0.10000 0.10000
0.10000 0.10000 0.10000 0.30000 0.30000 0.10000
0.10000 0.10000 0.10000 0.10000 0.30000 0.30000
0.30000 0.10000 0.10000 0.10000 0.10000 0.30000

observation: 6
0.20000 0.20000 0.10000 0.10000 0.10000 0.10000
0.20000 0.20000 0.20000 0.20000 0.10000 0.10000
0.20000 0.20000 0.20000 0.20000 0.20000 0.20000
0.20000 0.20000 0.20000 0.20000 0.20000 0.20000
0.10000 0.10000 0.20000 0.20000 0.20000 0.20000
0.10000 0.10000 0.10000 0.10000 0.20000 0.20000
0.405465
r98922053@linux12:~/hw1/c_cpp $ make clean
rm -f test_hmm    # type make clean to remove the compiled file
r98922053@linux12:~/hw1/c_cpp $
```

Input and Output of your programs

- Training algorithm
  - input
    - number of iterations
    - initial model (model_init.txt)
    - observed sequences (seq_model_01~05.txt)
  - output
    - $\lambda = (A, B, \pi)$ for 5 trained models

      5 files of parameters for 5 models (model_01~05.txt)

- Testing algorithm
  - input
    - trained models in the previous step
    - modellist.txt (file saving model name)
    - Observed sequences (testing_data1.txt & testing_data2.txt)
  - output
    - best answer labels and $P(O|\lambda)$ (result1.txt & result2.txt)

Program Format Example

```
./train iteration model_init.txt seq_model_01.txt model_01.txt

./test modellist.txt testing_data.txt result.txt
```

- The arguments need to be variable path(it is not necessary to be in the directory the program executed).
- Use argv in main function to pass the arguments.

Input Files

```
+- dsp_hw1/
   +-  c_cpp/
   |     +-
   +-  modellist.txt     //the list of models to be trained
   +-  model_init.txt   //HMM initial models
   +-  seq_model_01~05.txt //training data observation
   +-  testing_data1.txt   //testing data observation
   +-  testing_answer.txt //answer for "testing_data1.txt"
   +-  testing_data2.txt   //testing data without answer
```

Observation Sequence Format

seq_model_01~05.txt / testing_data1.txt

```
ACCDDDDFFCCCCBCFFFCCCCCEDADCCAEFCCCACDDFFCCDDFFCCD
CABACCAFCCFFCCCDFFCCCCCDFFCDDDDFCDDCCFCCCEFFCCCCBC
ABACCCDDCCCDDDDFBCCCCCDDAACFBCCBCCCCCCCFFFCCCCCDBF
AAABBBCCFFBDCDDFFACDCDFCDDFFFFFCDFFFCCCDCFFFFCCCCD
AACCDCCCCCCCDCEDCBFFFCDCDCDAFBCDCFFCCDCCCEACDBAFFF
CBCCCCDCFFCCCFFFFFBCCACCDCFCBCDDDCDCCDDBAADCCBFFCC
CABCAFFFCCADCDCDDFCDFFCDDFFFCCCDDFCACCCCDCDFFCCAFF
BAFFFFFFFCCCCDDDFFCCACACCCDDDFFFCBDDCBEADDCCDDACCF
BACFFCCACEDCFCCEFCCCFCBDDDDFFFCCDDDFCCCDCCCADFCCBB
......
```

Model Format

- model parameters.

(model_init.txt /model_01~05.txt)

initial: 6

0	1	2	3	4	5
0.22805	0.02915	0.12379	0.18420	0.00000	0.43481

Prob(q_1=3|HMM) = 0.18420

transition: 6

0	0.36670	0.51269	0.08114	0.00217	0.02003	0.01727
1	0.17125	0.53161	0.26536	0.02538	0.00068	0.00572
2	0.31537	0.08201	0.06787	0.49395	0.00913	0.03167
3	0.24777	0.06364	0.06607	0.48348	0.01540	0.12364
4	0.09149	0.05842	0.00141	0.00303	0.59082	0.25483
5	0.29564	0.06203	0.00153	0.00017	0.38311	0.25753

Prob(q_{t+1}=4|q_t=2, HMM) = 0.00913

observation: 6

A	0.34292	0.55389	0.18097	0.06694	0.01863	0.09414
B	0.08053	0.16186	0.42137	0.02412	0.09857	0.06969
C	0.13727	0.10949	0.28189	0.15020	0.12050	0.37143
D	0.45833	0.19536	0.01585	0.01016	0.07078	0.36145
E	0.00147	0.00072	0.12113	0.76911	0.02559	0.07438
F	0.00002	0.00000	0.00001	0.00001	0.68433	0.04579

Prob(o_t=B|q_t=3, HMM) = 0.02412

Model List Format

- Model list: modellist.txt testing_answer.txt

modellist.txt	testing_answer.txt
model_01.txt	model_01.txt
model_02.txt	model_05.txt
model_03.txt	model_01.txt
model_04.txt	model_02.txt
model_05.txt	model_02.txt
	model_04.txt
	model_03.txt
	model_05.txt
	model_04.txt
	…….

Output Format

- result.txt
  - Hypothesis model and it likelihood

```
model_01.txt        1.0004988e-40
model_05.txt        6.3458389e-34
model_03.txt        1.6022463e-41
…….
```

result1.txt

```
94 model_03.txt 2.019640e-34
95 model_03.txt 1.349792e-39
96 model_02.txt 3.839207e-39
97 model_05.txt 1.641065e-41
98 model_02.txt 7.878113e-41
```

- acc.txt
  - Calculate the classification accuracy.
  - ex.0.8566
  - Only the highest accuracy!!!
  - Only number!!!
  - Don’t need to submit the code for calculating accuracy.

acc.txt

```
1 0.869600
2
```

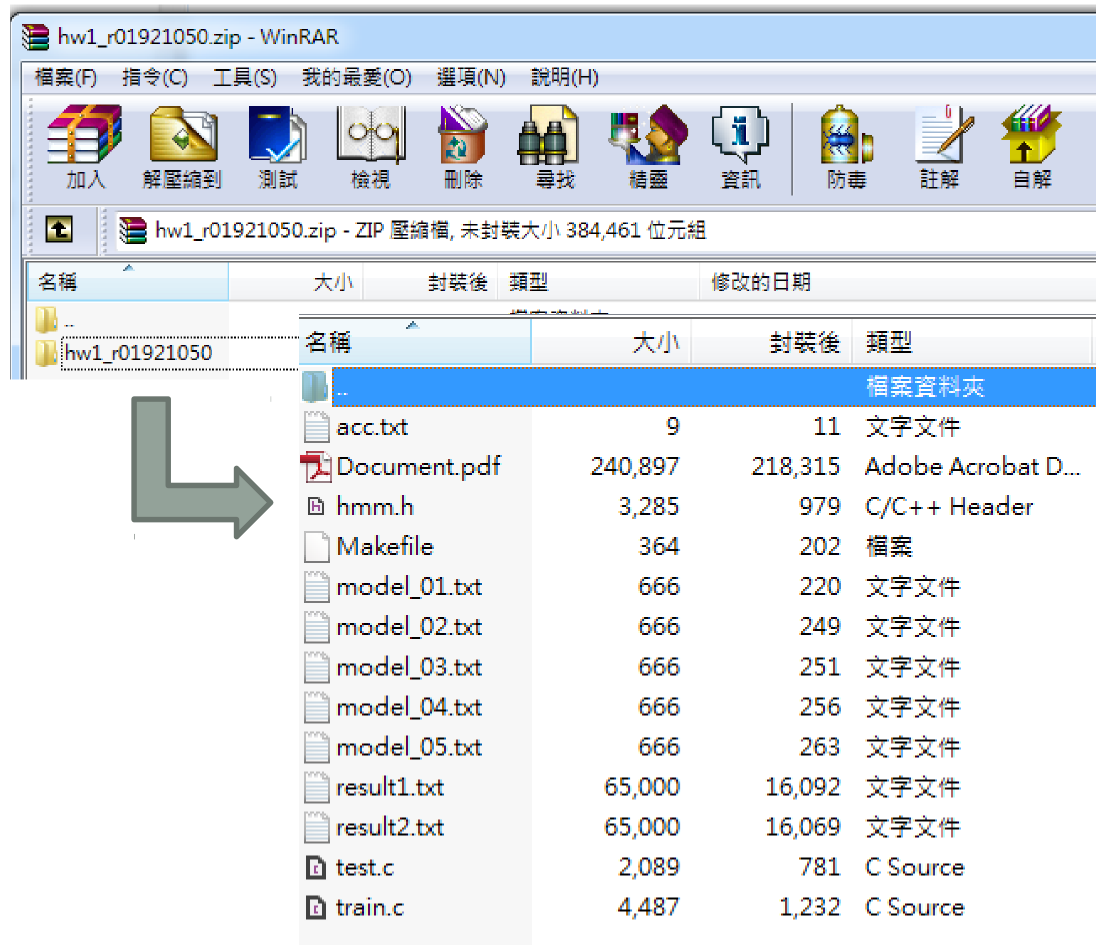
Submit Requirement

- Upload to CEIBA
- Your program
  - train.c, test.c, hmm.h, Makefile
- Your 5 Models After Training
  - model_01~05.txt
- Testing result and and accuracy
  - result1~2.txt (for testing_data1~2.txt)
  - acc.txt (for testing_data1.txt)
- Document (pdf) **(No more than 2 pages)**
  - Name, student ID, summary of your results
  - Specify your environment and how to execute.

Submit Requirement

Compress your hw1 into “hw1_[學號].zip”

+- hw1_[學號]/
- +- train.c /.cpp
- +- test.c /.cpp
- +- hmm.h
- +- Makefile
- +- model_01~05.txt
- +- result1~2.txt
- +- acc.txt
- +- Document.pdf (pdf)



Remark

- Testing environment: CSIE workstation(gcc 7.3).
- If C++11 is used, add -std=c++11 in your makefile.
- You have to make sure **your program is able to compile**(hmm.h should be submitted).
- The arguments of your program have to be given **in the runtime**(provided by argv in main function).
- **Do not** compress the directory by **RAR/TAR.**
- The **testing program** should run in **10 minute.**
- FAQ

Grading Policy

- **Accuracy 30%**
- **Program 35%**
- **Report 10%**
  - Environment + how to execute + summary of your program.
- **File Format 25%**
  - zip & fold name
  - result1~2.txt
  - model_01~05.txt
  - acc.txt
  - makefile
  - Command line (train & test) *(see page. 25)*

You may get zero point in file format if the format is wrong.

- **Bonus 5%**
  - Impressive analysis in report.

Do Not Cheat!

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

- We will compare your code with others.
  (including students who has enrolled this course)

Contact TA

- ntudigitalspeechprocessingta@gmail.com 周儒杰

Office Hour: Tuesday 13:00-14:00 電二 531

Please let me know you‘re coming by email, thanks!