Homework #3
CS662, Fall 2017
Due **Tuesday, October 10, midnight**
Late homework 30% off (up to one week late)
PLEASE PUT YOUR NAME AT THE TOP OF writeup.pdf

==========

NOTES ON CARMEL SWITCHES

==========

```
-t D M          train weights of WFST M on data D (alternating string pairs)
-! 15           execute 15 random restarts, return best scoring model
-M 200          do at most 200 iterations of EM
-X 0.99999      quit EM if P(observed-data) didn't change much from last iteration
-?              faster EM, more memory (caches derivations)
-:              even faster EM, even more memory
--train-cascade D M1 M2 …     train weights of machines M1, M2, … on data D
-HJ             maintain the state names of machines when training
(A (B "x" "y" 0.275!))        fix parameter value at 0.275 (EM won't change it)
```

0. Resource

Work through **carmel-training.pdf**, which describes EM training of WFSAs and WFSTs (and cascades of them) using Carmel.

1. Transliteration (35 points)

In the part of the assignment, you will process a set of *unaligned* Japanese/English sound sequences (**epron-jpron-unsupervised.data**). You will automatically align the data and learn a WFST (**epron-jpron-unsupervised.wfst**).

Use the following generative story that probabilistically generates Japanese sequences from English ones:

> Given: English sound sequence e_1 ... e_r.
> Process: for i = 1 to r
> output J (a particular sequence of *one or more* Japanese sounds)
> with probability P(J | e_i)

An alignment of a word pair consists of connections between the English sounds and Japanese sounds. As implied by the generative story, (1) each English sound may be connected to one or more (contiguous) Japanese sounds, (2) each Japanese sound must be connected to exactly one English sound, and (3) connections may not cross each other.

These are legal alignments:

These are illegal alignments:

```
AH L ER   T        AH L  ER   T
| / /     |          X    |   |
A R A A T O        A R A A T O
```

Note that the generative story requires that the English side have no more sounds than the Japanese side.

Complete data, incomplete model

Given the following manually aligned word pairs:

```
("boat")          ("archer")              ("armor")        ("test")
B  OW   T      AA R CH   ER          AA R M ER      T EH S   T
|  /\  /\       / /  /\  /\           / / / /\       | | /\  /\
B O O T O      A A CH Y A A          A A M A A      T E S U T O
```

- *Question 1.* Calculate maximum likelihood estimates for:

P(A A | AA)	= ?	P(T | T)	= ?
P(M | AA)	= ?	P(T O | T)	= ?
P(A | AA)	= ?	P(S | T)	= ?

Complete model, incomplete data

Given the following model parameters:

P(A | AH)	= 0.8	P(A T O | T)	= 0.0
P(A R | AH)	= 0.1	P(T O | T)	= 0.5
P(A R A | AH)	= 0.1	P(O | T)	= 0.1
P(R | L)	= 0.6	P(A | ER)	= 0.1
P(R A | L)	= 0.2	P(A A | ER)	= 0.6
P(R A A | L)	= 0.1	P(A A T | ER)	= 0.0
P(A | L)	= 0.0	P(A T | ER)	= 0.0
P(A A | L)	= 0.0	P(T | ER)	= 0.0

- *Question 2.* Draw all of the legal alignments for the pair AH L ER T / A R A A T O. Indicate which one is the most probable alignment, according to the model parameters above.

Incomplete model, incomplete data

Write a program to generate all legal alignments for any word pair, such as those found in **epron-jpron-unsupervised.data**.

- *Question 3.* Show all legal alignments for the first five word pairs in **epron-jpron-unsupervised.data**, as printed automatically by your program.

Then implement an EM algorithm for computing parameter estimates and best sequence-pair alignments for all word pairs in **epron-jpron-unsupervised.data**.

- *Question 4.* Show the highest-scoring alignments for each of the first five word pairs, after each of the first five EM iterations.

Create an alignment file (**epron-jpron.alignment**) in the same integer-sequence format as **epron-jpron.data** from the previous homework.

- *Question 5.* What is the accuracy of your automatic alignment? (Compare all integers in **epron-jpron.alignment** and **epron-jpron.data**).

Put your source code in a directory called **align/**, including a **README** file telling us how to run your code.

Format your learned parameter estimates into a Carmel weighted finite-state transducer called **epron-jpron-unsupervised.wfst**. Only include transitions with probability bigger than 0.01.

Using **eword.wfsa**, **eword-epron.wfst**, **epron-jpron-unsupervised.wfst**, and Carmel, produce the top-5 decodings of the following Japanese sound sequences (**japanese.txt**) into English.

```
"A" "N" "J" "I" "R" "A" "N" "A" "I" "T" "O"
"S" "U" "CH" "I" "I" "B" "E" "N" "R" "A" "R" "U" "Z" "U"
"D" "O" "N" "A" "R" "U" "D" "O" "T" "O" "R" "A" "N" "P" "U"
"SH" "Y" "E" "R" "I" "R" "U" "S" "A" "N" "D" "O" "B" "A" "A" "G" "U"
```

- *Question 6.* Show your decodings.

Hint. An EM algorithm sketch

1. iteration = 0
2. for each word pair $<e_1...e_r, j_1...j_m>$ in the corpus:
   - 2a. enumerate all legal alignments $a_1...a_n$
   - 2b. compute alignment probabilities $P(a_1)...P(a_n)$ as follows:
     - if (iteration == 0) then $P(a_i) = 1/n$ (for all i)
     - else:

```
for i = 1 to n
   P(a_i) = 1
   for k = 1 to r
      let J be the Japanese sound sequence produced by English sound e_k in a_i
      P(a_i) *= P(J | e_k)
/* normalize to yield P(a_i) estimates */
total = 0
for i = 1 to n
   total += P(a_i)
for i = 1 to n
   P(a_i) = P(a_i) / total
/* good place to print best alignment for this word pair */
```

   - 2c. collect fractional counts over all legal alignments as follows:

```
for i = 1 to n
   for k = 1 to r
      let J be the Japanese sound sequence produced by English sound e_k in a_i
      count(J | e_k) += P(a_i)     /* perhaps use a hash table? */
```

3. normalize sound-translation counts to yield P(J | e) estimates as follows:

```
for each distinct English sound e
   total = 0
   for each distinct sequence j...j such that count(J | e) > 0   /* perhaps map over hash table? */
```

```
        total += count(J| e)
      for each distinct sequence j...j such that count(J | e) > 0
        P(J | e) = count(J | e) / total
4. clear counts as follows:
    for each J and e such that count(J | e) > 0,
      count(J | e) = 0
5. iteration += 1
6. if (iteration < max-iterations), then go to step 2.
```

2. Part of speech tagging (15 points)

*This part of the homework uses Carmel **exclusively**. You do not need to implement EM yourself.*

Here, you will automatically tag sequences of words with their parts of speech. But this time, the training data (**tagging.data**) is an *untagged* sequence of words.

We will follow this generative story of English word sequences:

```
Given:  Nothing.
Process:  i=0
          while (true) do
                i += 1
                choose tag t_i with probability P(t_i | t_i-1)        /* tagging.fsa */
                if (t_i == STOP) then break
                output word w_i with probability P(w_i | t_i)        /* tagging.fst */
```

Look at the tag-bigram FSA **tagging.fsa**. This FSA is fully connected. Any tag follows any other tag with uniform probability. Look also at the tag-to-word FST **tagging.fst**. This FST contains connections between words and their legal tags, according to a dictionary. Probabilities are also uniform. When cascaded, the FSA and FST implement the above generative story.

We now investigate which parameter values for **tagging.fsa** and **tagging.fst** optimize the probability of the observed word sequence **tagging.data**.

Run Carmel EM to find out:

```
% carmel --train-cascade -HJ -X 0.9999 -M 5 -! 0 tagging.data tagging.fsa tagging.fst
% carmel --project-right --project-identity-fsa -HJ tagging.fsa.trained > tagging.fsa.trained.noe
% awk 'NF>0' tagging.data > tagging.data.noe
% cat tagging.data.noe | carmel -qbsriWIEk 1 tagging.fsa.trained.noe tagging.fst.trained
```

- *Question 7.* What tagging accuracy is achieved? (Compare **the output of the last command above** to the correct **tagging.key**).
- *Question 8.* Can you achieve better results with more iterations and/or random restarts?

3. Code-breaking (50 points)

*This part of the homework **does not** involve Carmel. You may implement a Carmel model as a sanity check, but do not turn in results of any Carmel runs.*

You intercept a piece of enciphered English, given in the file **cipher.data**:

```
SGDN DNFMOH DN M KQWX NDOTCFMW EOQ DS IEONDNSN EY FDSSFQ QFNQ SGMO
SGQ NQM NMOH MOH DN MLECS SGWQQ PDFQN FEOT DSN LWQMHSG MS OE REDOS
QUIQQHN M ZCMWSQW EY M PDFQ DS DN NQRMWMSQH YWEP SGQ PMDO FMOH LX M
NIMWIQFX RQWIQRSDLFQ IWQQV EEJDOT DSN AMX SGWECTG M ADFHQWOQNN EY
WQQHN MOH NFDPQ M YMKEWDSQ WQNEWS EY SGQ PMWNGGQO SGQ KQTQSMSDEO MN
PDTGS LQ NCRRENQH DN NIMOS EW MS FQMNS HAMWYDNG OE SWQQN EY MOX
PMTODSCHQ MWQ SE LQ NQQO OQMW SGQ AQNSQWO QUSWQPDSX AGQWQ YEWS
PECFSWDQ NSMOHN MOH AGQWQ MWQ NEPQ PDNQWMLFQ YWMPQ LCDFHDOTN
SQOMOSQH HCWDOT NCPPQW LX SGQ YCTDSDKQN YWEP IGMWFQNSEO HCNS MOH
YQKQW PMX LQ YECOH DOHQQH SGQ LWDNSFX RMFPQSSE LCS SGQ AGEFQ DNFMOH
ADSG SGQ QUIQRSDEO EY SGDN AQNSQWO REDOS MOH M FDOQ EY GMWH AGDSQ
LQMIG EO SGQ NQMIEMNS DN IEKQWQH ADSG M HQONQ COHQWTWEASG EY SGQ
NAQQS PXWSFQ NE PCIG RWDJQH LX SGQ GEWSDICFSCWDNSN EY QOTFMOH SGQ
NGWCL GQWQ EYSQO MSSMDON SGQ GQDTGS EY YDYSQQO EW SAQOSX YQQS MOH
YEWPN MO MFPENS DPRQOQSWMLFQ IERRDIQ LCWSGQODOT SGQ MDW ADSG DSN
YWMTWMOIQ
```

The plaintext is unknown. Use the following generative model for ciphertexts:

Given: Nothing.
Process: i=0
  while (true) do
    i += 1
    choose English letter e_i with probability $P(e_i \mid e_{i-1})$ /* English bigrams */
    if (e_i = STOP) then break;
    if (e_i = SPACE) then output cipher letter SPACE
    else output cipher letter c_i with probability $P(c_i \mid e_i)$ /* Substitutions */

Estimate English letter-bigram probabilities from **english.data**. Include only the letters A-Z plus SPACE. This text is unrelated to the cipher or its solution.

Write an EM algorithm to attack **cipher.data**. You will need to implement the forward-backward algorithm, and you may need to experiment with different learning parameters. An English bigram model is very weak, so you will not get a perfect decipherment. You should work with a portion of the cipher first -- this will make your testing much faster.

- *Question 9.* Make a graph with P(cipher) on the y-axis and the number of EM iterations on the x-axis. P(cipher) is computed by the forward algorithm. Your graph should be monotonically increasing.
- *Question 10.* Show the Viterbi decoding (best plaintext) after each of the first five EM iterations.
- *Question 11.* What is the best decoding from your last iteration? Show the text.
- *Question 12.* What are your final substitution probabilities? Show the substitution table. You may suppress any values less than 0.01.

Submit your source code under a directory called **decipher/**, and include a **README**.

Note #1: if multiplying many small probabilities causes numerical underflow, you may choose to store/manipulate log probabilities instead of regular probabilities. Instead of storing values like x and y, you store values for log(x) and log(y). Then, when you want to multiply x * y, you instead add log(x) + log(y). But what if you want to add x + y (as in the forward algorithm)? Here is a trick to do that without converting the stored log(x) and log(y) values back into regular probabilities:

```
add(log x, log y)
    if log x = -infinity, then output log y
    if log y = -infinity, then output log x
    if log x - log y > 16 (for floats, 32 double) then output log x      /* y is minor anyway */
    if log x > log y then output log x + log(1 + exp(log y - log x))
    if log y - log x > 16 (for floats, 32 double) then output log y      /* x is minor anyway */
    if log y > log x then output log y + log(1 + exp(log x - log y))
```

Note #2: Advanced techniques to get better results: (1) letter-trigram model, (2) cube all channel model probabilities after EM is finished, but before final Viterbi decoding.

==========

WHAT TO TURN IN ON BLACKBOARD

==========

We provide:

carmel-training.pdf
epron-jpron-unsupervised.data
tagging.data
tagging.fsa
tagging.fst
tagging.key
cipher.data
english.data

Make a tar file called FirstName_LastName_Assignment3.tar. It should unpack into a directory called FirstName_LastName_Assignment3/. That directory should include:

epron-jpron.alignment
epron-to-jpron-unsupervised.wfst
align/ (including README)
decipher/ (including README)

writeup.pdf (or .txt)

- Your name.
- Answers to all numbered questions above.
- Any strategies, observations, or experiments you want to share.