

HW2: Incomplete data

Note: Please submit the written aspects of assignments in postscript or PDF format only. I *highly* recommend you use \LaTeX to prepare the assignments. A solution will be posted here after the due date. See <http://www.cs.utah.edu/classes/cs5964/handin.html> for handin instructions.

Part A

(due 3 Oct)

In this part, we will consider a unsupervised model for *morphology*. In order to do this, we will work with single words at a time and the elements in our models will be characters.

The generative story for a single word is as follows. First, generate a sequence of affix-types. Second, map each affix-type to a character. There will be three affix-types in our model: PREFIX, SUFFIX, WORD (which I'll write as P, S and W, respectively). The sequence of $\{P,S,W\}$ will be generated according to the regular expression "PW+S". In other words, we have some number (possibly zero) of prefix characters, followed by at least one word character, followed by some number (possibly zero) of suffix characters. Here are some examples for some real words:

1. unhappy ; \iff ; PPWWWWWW
2. ignore ; \iff ; WWWWWWW
3. writes ; \iff ; WWWWWWS
4. unfriendly ; \iff ; PPWWWWWWSS

Note that there will necessarily be some ambiguity. For instance, in the word "happiness," should the "i" be a W or a S? (That's a rhetorical question – we won't actually worry about this.)

Our model will be trained in an unsupervised fashion: that is, the only data we have access to is a list of English words w_1, \dots, w_N . We will treat the affix-type sequences as *latent variables* in this model.

In the noisy channel framework, we'll have a *source* model that generates sequences of $\{P,S,W\}$ (subject to the constraints mentioned before) and a *channel* model that maps affix-types to Latin characters.

As usual, we will make our source model a bigram model over affix-type sequences. So, for instance, in the case of the sequence associated with the word "unfriendly" above, we would have:

$$p_S(\text{PPWWWWWWSS}) = p(P | \langle s \rangle) p(P | P) p(W | P) p(W | W)^5 p(S | W) p(S | S) p(\langle /s \rangle | S)$$

(Note the fifth power in $p(W | W)$ to account for the sequence of W characters in the middle.)

So far, this is very much like the cryptanalysis model, except for the fact that we have a structural constraint on the possible source sequences.

To make things more interesting (and more realistic), we will make the channel model slightly more complex. In particular, we will use a *bigram* channel model, so that the i th Latin character depends not only on the i th affix-type, but also on the $i - 1$ st affix-type. So, continuing the "unfriendly" example, according to the *channel* we would have:

$$p_C(\text{unfriendly} | \text{PPWWWWWWSS}) = p(u | P, \langle s \rangle) p(n | P, P) p(f | W, P) p(r | W, W) p(i | W, W) \\ p(e | W, W) p(n | W, W) p(d | W, W) p(l | S, W) p(y | S, S)$$

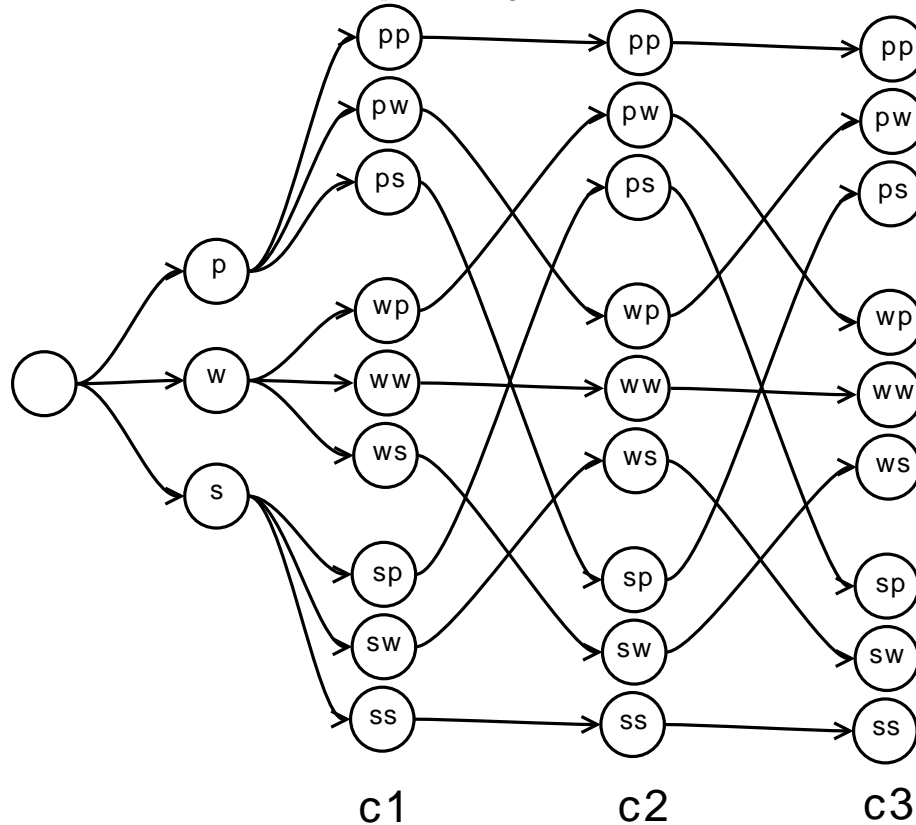
The hope is that by having a bigram model in the channel, we can account for things like "y \rightarrow i" in the case of "happiness."

Our model depends on two sets of parameters, both of which I'll call θ . In particular, let $\theta_{t,u}^{(s)} = p_s(u | t)$ be the bigram (source) probabilities and let $\theta_{t,u,c}^{(c)} = p_c(c | t, u)$ be the bigram (channel) probabilities.

Question 1: Suppose that we have explicitly enumerated all possible affix-type sequences $\bar{s} = \langle s_1, \dots, s_N \rangle$ corresponding to some word w of length N . Suppose we have also computed $p(\bar{s} | w)$ for each of these. Write down the equations for the re-estimated values of both types of θ parameters.

The result of composing these models will produce a lattice—as before—but it will be slightly more complex. In particular, each state in the lattice will have to correspond to a *pair* of affix-type tags. This is because the generation of a single Latin character depends on pairs on affix tags. Thus, each state will represent both the *previous* and *current* tag for a given character. Note that the lattice also won't be fully connected from one time state to the next. This is because it isn't possible to transition, for instance from “prev=P cur=W” to “prev=S cur=?” because the “prev” at time i has to match the “cur” at time $i - 1$.

This is drawn for three time steps in the figure below:



Here, state “sp” for instance means “previous tag was S and current tag is P”. (Note that actually this transition shouldn't be possible because we can transition from S to P, but ignore that for now.)

We will refer to a single state in this lattice as “ $S_{t,u,n}$ ”, where t is the previous tag, u is the current tag and n is the time step. For instance, the top-right state in the figure would be $S_{p,p,3}$ and the one below it would be $S_{p,w,3}$.

Question 2: Forward probabilities α_S and backward probabilities β_S can be computed recursively in this lattice. Write down the recursive computation for all α s and all β s. This should look something like $\alpha_{S_{p,p,i}} = \dots$; $\alpha_{S_{p,w,i}} = \dots$; and so on, for each possible combination of t and u . (There will be nine.) Some of these will be zero because t, u is not a valid state according to the prefix-word-suffix model. Do the same thing for the backward probabilities. These should all depend on previous α s (or β s) and the current θ s. You can ignore the “time=1” and “time=N-1” cases because these depend on boundary conditions.

Part B**(due 17 Oct)**

In this assignment, we will build a transliteration system that maps between English words and Japanese words. So that you guys can all read the outputs, we'll use *romanji* to denote the Japanese words (this basically just means writing Japanese characters in Latin (or "Roman") characters...in Japanese, *ji* means "character"). So, for example, the romanji for the English word "computer" is "ko-n-pi-yu-u-ta-a", the name "Hal Daume" becomes "ha-ru da-u-me" and the name "Bill Clinton" becomes "bi-ru ku-ri-n-to-n".

We'll work in an "indirect model." That is, one that goes through phonemes. This process will be directed from English to Japanese, so that English is the source and Japanese is observation. (This means that we will be trying to *back-transliterate* from Japanese to English.) Our model will work as follows:

1. First, generate an English word/phrase w according to $p(w)$
example: knight
2. Next, map w into a phoneme sequence e with probability $p(e | w)$
example: knight \Rightarrow N AY T
3. Next, map the English phoneme sequence e to a Japanese phoneme sequence j with probability $p(j | e)$
example: N AY T \Rightarrow N A I T O
4. Finally, render the Japanese phoneme sequence as a romanji sequence r with probability $p(r | j)$
example: N A I T O \Rightarrow NA-I-TO

This gives $p(r | w) = \sum_e \sum_j p(e | w)p(j | e)p(r | j)$, which looks nasty because of the sums. But we'll implement all of these models in finite state machines and this will just be composition.

We will ignore step 4. Also, to make life easier on you (and harder on me!), I've already solved parts 1 and 2 for you! These are stored as transducers `w.pfsa` and `w-e.pfst` in the project 2 directory on Cade! The whole point of this project is for you to develop a transducer for stage 3.

In order to do so, I have provided a list of about 2500 English/Japanese word pairs in phonetic format in the file `ej.data`. An example would be:

A H L E R T \Leftrightarrow A R A A T O

Here, the left-hand side is the English phoneme sequence and the right-hand side is the Japanese phoneme sequence.

In this assignment, we will use a simple generative story for stage 3 (how English sound sequences get converted into Japanese sound sequences):

1. Given an English sound sequence e_1, \dots, e_N :
2. For $n = 1 \dots N$,
 - (a) Output $j \dots j$ (a particular sequence of *one or more* Japanese sounds with probability $p(j \dots j | e_n)$)

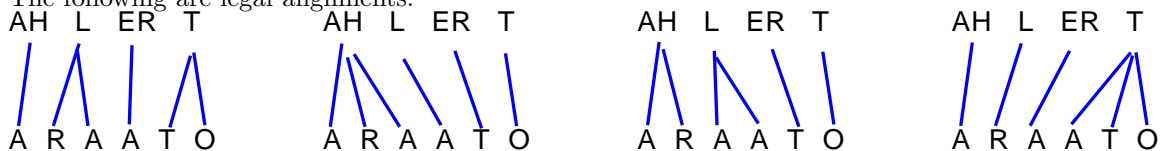
A sample entry in the $p(j \dots j | e_n)$ might be:

$$p(\text{T O} | \text{T}) = 0.42 \quad \text{probability that English "T" is translated to Japanese "T" "O"}$$

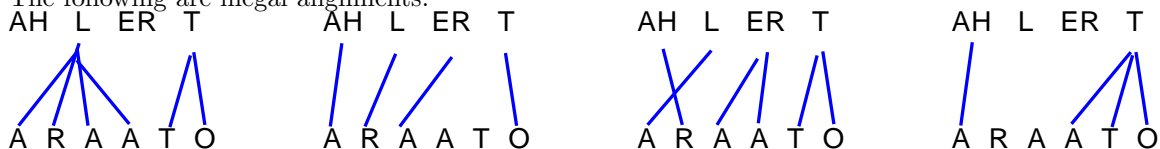
An *alignment* of a word pair consists of connections between the English sounds and Japanese sounds. Following the generative story:

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,
3. connections may not cross.

The following are legal alignments:

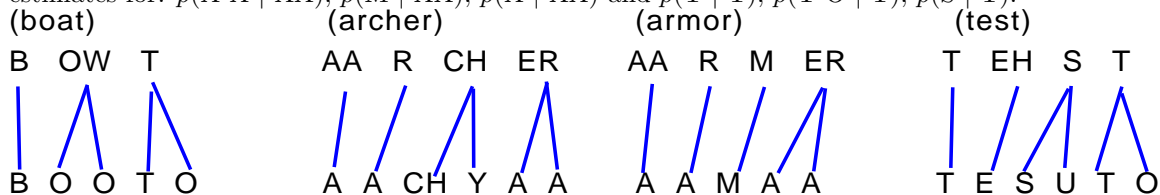


The following are illegal alignments:



Note that the generative story requires that the Japanese side have at least as many sounds as the English side.

Question 1: Given the manually aligned word pairs shown below, calculate and hand in maximum likelihood estimates for: $p(A A | AA)$, $p(M | AA)$, $p(A | AA)$ and $p(T | T)$, $p(T O | T)$, $p(S | T)$.



Question 2: Given the model parameters shown below, draw and turn in *all* possible alignments (hint: there are ten) for the pair “AH L ER T / A R A A T O”. Note which one is the most probable alignment according to the model parameters.

$$\begin{aligned}
 p(A | AH) &= 0.8 & p(A T O | T) &= 0.0 \\
 p(A R | AH) &= 0.1 & p(T O | T) &= 0.5 \\
 p(A R A | AH) &= 0.1 & p(O | T) &= 0.1 \\
 \\
 p(R | L) &= 0.6 & p(A | ER) &= 0.1 \\
 p(R A | L) &= 0.2 & p(A A | ER) &= 0.6 \\
 p(R A A | L) &= 0.1 & p(A A T | ER) &= 0.0 \\
 p(A | L) &= 0.0 & p(A T | ER) &= 0.0 \\
 p(A A | L) &= 0.0 & p(T | ER) &= 0.0
 \end{aligned}$$

Question 3: We want to generate a PFST to transduce from Japanese sounds to English sounds. In order to keep the size of the transducer down, we only want to include transitions that actually occur in the data; namely `ej.data`. Remember that you need to include Japanese sounds sequences of arbitrary length. In

order to do this, I would construct the PFST as follows. For each Japanese sound sequence jjj , generate a state called jjj . Each of these can transition with English sound e and a Japanese epsilon to the START state with the appropriate probability. Similarly, for each Japanese sound sequences jjj_1 and jjj_2 such that jjj_2 can be obtained by adding a *single* Japanese sound s to the end (i.e., $jjj_1 = jjj_2s$), create an arc from the state jjj_1 to the state jjj_2 with an epsilon on the English side and sound s on the Japanese side with probability one. You should initialize the probabilities uniformly (that means that $\sum_{jjj} p(jjj | e) = 1$ but are otherwise equal. (Hint: this doesn't mean that every probability in the model will be equal – just those with the same e .) Based on the data in `ej.data`, create such a PFST and call it `e-j1.pfst`. To keep it smaller, you can remove any English/Japanese pair that occurs strictly fewer than five times, provided that the Japanese pair has length 2 or more. For me, after doing this, I wound up with 3784 E/J pairs.

Question 4: Using this created PFST (with uniform weights), we'll use `carmel` to generate the top five outputs for each line in the test data. Write down the results. Do these seem reasonable? If not, why do you think the machine chose the outputs it did. We can do this by executing:

```
% carmel -brIEQk 5 w.pfsa w-e.pfst e-j1.pfst test-data
```

Question 5: Now, we will use `carmel` to estimate the probabilities in `e-j1.pfst`. The switch `-t` will do this for us. We can have `carmel` train the weights based on the `ej.data`. This command would normally print the new transducer to `stdout`, so we'll redirect to `e-j2.pfst`.

```
% carmel -tjp 1e-6 ej.data e-j1.pfst > e-j2.pfst
```

(`-t` means train, `-j` means normalize joint probabilities and `-p 1e-6` means to not print arcs with probability less than 10^{-6}). As this is running (warning: it will probably take a few minutes and be warned that it will give you some warnings as it goes...you can ignore them), it will print out perplexities at each iteration. Write these down.

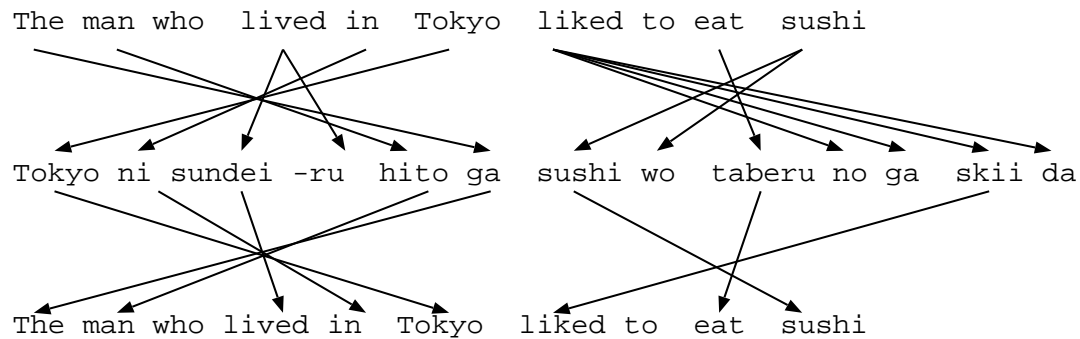
Question 6: Finally, we'll run the new new trained transducer on our data to see what it says (the command to do this is below). Write down the top five outputs for each input and guess which of them is right (if any). If none is right, can you tell why it didn't find the correct output? (You may have to look in the transducers I provided in order to figure this out). You can run:

```
% carmel -brIEQk 5 w.pfsa w-e.pfst e-j2.pfst test-data
```

Part C

(due 7 Nov)

Question 1: Consider the E-to-J alignment and J-to-E alignment below. Draw an alignment matrix for (a) intersection, (b) union and (c) grow-diag-final.



Question 2: Using the grow-diag-final alignments produced for question 1, list all of the phrases that would be extracted. For each phrase, mark whether you think it is a good (i.e., useful) phrase or a bad phrase.

Question 3: Suppose that we build an MT systems that produced the translations on the right hand side of the table below. The reference translations are on the left-hand side. Compute the BLEU-4 score for this system, by counting (and writing down) unigram through 4-gram matches and brevity penalty.

yesterday the man ate a big sandwich the man consumed a huge sandwich yesterday	yesterday the man ate huge sandwich
today he had a very upset stomach the sandwich upset his stomach today	his stomach was very upset today