Natural Language Processing, Spring 2017, HW 2

Prof. Liang Huang

Due Monday May 1st at 11:59pm on Canvas (each group only submits one copy)

In Japanese text, foreign words borrowed in the modern era (such as Western names and technical terms) are *transliterated* into special symbols called *katakana*. Katakana is a **syllabary**,[1] in which most symbols stand for syllable sounds (such as *ko* or *ra*). Because spoken Japanese consists largely of consonant-vowel syllables, most katakana symbols stand for two Japanese phonemes. In this assignment, among other things, we'll decode katakana words of English origin back into English. Refer to the slides for this section and the tutorial on writing systems and transliteration for more information (available from the course website).

Download `http://classes.engr.oregonstate.edu/eecs/spring2017/cs519-001/hw2/hw2-data.tgz` which includes:

`eword.wfsa`	a unigram WFSA of English word sequences
`epron.wfsa`	a trigram WFSA of English phoneme sequences
`eword-epron.data`	an online dictionary of English words and their phoneme sequences
`eword-epron.wfst`	a WFST from English words to English phoneme sequences
`epron-eword.wfst`	inverse transducer; the result of `carmel -v eword-epron.wfst`
`epron-espell.wfst`	a WFST from English phoneme sequences to English letter sequences
`epron-jpron.data`	a database of aligned English/Japanese phoneme sequence pairs
`jprons.txt`	a short list of Japanese Katakana sounds to decode
`epron.probs`	a human-readable version of `epron.wfsa`.

1 Part-of-Speech Tagging as WFST Composition (10 pts)

Recall that in the lectures we did POS tagging for "I hope that this works" and "They can fish" by composing a chain (word sequence), a flower (word/tag lexicon), and a tag bigram. Now you need to

1. build a Carmel WFSA `bigram.wfsa` for the tag bigram model (similar to the one from the slides, but add `PREP` and `AUX` for prepositions and aux. verbs); you can assign probabilities in any reasonable way.
2. build a Carmel WFST `lexicon.wfst` for the word/tag lexicon (big enough to cover the examples below); again, you can assign probabilities in any reasonable way, but make sure your pipeline is mathematically sound!
3. compose them to solve POS tagging for the above two sentences plus the following examples (show the resulting tag sequences from Carmel output):
   (a) A panda eats shoots and leaves
   (b) They can can a can
   (c) Time flies like an arrow
4. Why is your composition pipeline mathematically sound? Write the equations.
5. What funny interpretations did you observe? What are possible ways to fix them? (no need to implement them)

[1] See `https://en.wikipedia.org/wiki/Katakana` and `https://en.wikipedia.org/wiki/Syllabary`.

2 Pronouncing and Spelling English (30 pts)

1. Pick some English words and pronounce them with `eword-epron.wfst`. This transducer is essentially a giant lookup table built from `eword-epron.data`. Turn in five (5) examples. Here is one:

   ```
   echo HELLO | carmel -sliOEQk 5 eword-epron.wfst
   ```

2. Now let's try to pronounce character sequences without whole-word lookup (since we may face unknown words, for example). You can send a character string backwards through `epron-espell.wfst`. Try several words and turn in your examples; here is a suggested command:

   ```
   echo 'H E L L O' | carmel -sriIEQk 5 epron-espell.wfst
   ```

   Did you get some non-sense output? Why? Explain it in terms of probabilitic modeling.

3. Let's fix the nonsense by adding a language model of likely pronunciations, `epron.wfsa`. Try several and turn in your examples.

   ```
   echo 'H E L L O' | carmel -sriIEQk 5 epron.wfsa epron-espell.wfst
   ```

4. Now spell out some phoneme sequences by using `epron-espell.wfst` in the forward direction. Try several and turn in your examples.

   ```
   echo 'HH EH L OW' | carmel -sliOEQk 50 epron-espell.wfst
   ```

5. How to improve the above results? Well, you can try to take advantage of the language model `eword.wfsa` (as a filter). But you need a "bridge" from espell to eword. That's trivial, isn't it? Write a simple Python program `gen_espell_eword.py` to generate `espell-eword.wfst` from the word list in `eword-epron.data`. And now you can:

   ```
   echo 'HH EH L OW' | carmel -sliOEQk 50 epron-espell.wfst espell-eword.wfst eword.wfsa
   ```

   Try several examples and turn them in. Include `gen_espell_eword.py` and `espell-eword.wfst` in the submission also. (this only works for words in both `espell-eword.wfst` and `eword.wfsa`.)

6. Well, alternatively, you can go directly from epron to eword:

   ```
   echo 'HH EH L OW' | carmel -sliOEQk 50 epron-eword.wfst eword.wfsa
   ```

   Is this better than 5)? Try the same set of examples can compare the results with those from 5).

7. Try to pronounce some words that are not in the data file `eword-espell.data`. Try several examples and turn them in.

   ```
   echo WHALEBONES | carmel -sliOEQk 5 eword-epron.wfst
   ```

8. Now try pronouncing letter sequences of those same words. Try several examples and turn them in.

   ```
   echo 'W H A L E B O N E S' | carmel -sriIEQk 5 epron.wfsa epron-espell.wfst
   ```

9. How about phoneme sequences of new words? What happens when you use `eword-epron.wfst` versus `epron-espell.wfst`? Try several examples and turn them in.

   ```
   echo 'W EY L B OW N Z' | carmel -sriIEQk 5 eword-epron.wfst
   echo 'W EY L B OW N Z' | carmel -sliOEQk 5 epron-espell.wfst
   ```

10. You can hook up these transducers in unexpected ways. What is the following command trying to accomplish?

```
echo 'BEAR' | carmel -sliOEQk 10 eword-epron.wfst epron-eword.wfst eword.wfsa
```

11. Write up a half-page or so of observations from your experiments. What did you learn about these automata and what they can (or can't) do? Please take time to express your thoughts clearly using complete sentences.

3 Decoding English Words from Japanese Katakana (80 pts)

1. Look at `epron-jpron.data`. Each pair in this file consists of an English phoneme sequence paired with a Japanese phoneme sequence. For each pair, Japanese phonemes are assigned integers telling which English phonemes map to them. For example:

```
AE K T ER         ;; English phoneme sequence for 'actor'
A K U T A A       ;; Same word, loaned into Japanese
1 2 2 3 4 4       ;; e.g., Japanese T maps to the 3rd English sound
```

   From the data, we can see that each English phoneme maps to *one or more* Japanese phonemes. Estimate the channel probabilities $p(\mathbf{j} \mid s)$ where $\mathbf{j}$ represents one or more Japanese phoneme(s), and s an English phoneme, using the simplest maximum likelihood estimation (MLE). Write a Python program `estimate.py` that generates the file `epron-jpron.probs` like this (each line represents one $p(\mathbf{j} \mid s)$, and you can omit those with probability < 0.01 and you can omit those where one English phoneme maps to more than three (3) Japanese phonemes since they are mostly noise):

```
AE : A # 0.95
AE : Y A # 0.05
T : T # 0.4
T : T O # 0.4
T : TT O # 0.1
T : TT # 0.1
R : A A # 0.8
R : E R # 0.1
R : E R U # 0.1
...
```

   Transitions for each English phoneme should be consecutive (see above). Include `estimate.py` and `epron-jpron.probs`.

2. Now, based on this, you can create a WFST called `epron-jpron.wfst`. It should probabilistically map English phoneme sequences onto Japanese ones, behaving something like this in the forward direction:

```
echo 'L AE M P' | carmel -sliOEQk 5 epron-jpron.wfst
```

   and you'll get a list like

```
R A M P
R U A M P
R A M U P
R A M P U
R A M PP U
...
```

   Most transducers will consist of 300 transitions or so.

3. Using your knowledge about Japanese syllable structure (see slides), do these results (including their rankings) make sense? If not, why (briefly explain)?
4. How could you improve the results to sound more "Japanese like"? (no need to implement it)
5. Now consider the following Japanese katakana sequences, from a Japanese newspaper (actually these are the phoneme sequences that are easily read off from the katakana characters):

```
H I R A R I K U R I N T O N
D O N A R U D O T O R A N P U
B I D E O T E E P U
H O M A A SH I N P U S O N
R A PP U T O PP U
SH E E B I N G U K U R I I M U
CH A I R U D O SH I I T O
SH I I T O B E R U T O
SH I N G U R U R U U M U
G A A R U H U R E N D O
T O R A B E R A A Z U TCH E KK U
B E B I I SH I TT A A
S U K O TT O R A N D O
B A I A R I N K O N TCH E R U T O
A PP U R U M A KK U B U KK U P U R O
K O N P I U U T A S A I E N S U
H I J I K A R U T O R E E N I N G U
H I J I K A R U E K U S A S A I S U
A I S U K U R I I M U
H O TT O M I R U K U
T O R I P U R U R U U M U
K U R A U N P U R A Z A H O T E R U
H E E S U B U KK U R I S A A TCH I S A I E N T I S U T O
W O R U H U G A N G U M O TS U A R U T O
```

   (For your convenience I have included a `jprons.txt` for the above sequences.)

   How would you decode them into English *phoneme sequences* using carmel (with the WFST you just built and with help of `epron.wfsa`)? Show the command-line and results (with probs, with `-k 5` option). Do they make sense (as English words)?
6. Now redo the above exercise, but this time into English *words* by assembling `eword.wfsa`, `eword-epron.wfst`, and `epron-jpron.wfst` with carmel. Show the command-line and results (with probs, with `-k 5`). Do you think the results make more sense this time? Briefly explain.
7. Some desired outputs were not ranked top. Why? How would you improve the results? (No need to implement any of these)
8. Try search for at least five (5) other Japanese Katakana examples not covered in this HW or in slides, and try to decode them back into English using the above approach. Try to be creative. They should preferably be a two word phrase or a compound word, like "shopping center" or "piano sonata". You can use Google Translate (from English to Katakana) to verify your results.
9. Are there other ways to decode into English words given all these existing WFSAs and WFSTs (with some tweaking)? What are the differences of these methods (speed, accuracy, etc.)?
10. If you can make some new WFSAs/WFSTs such as `espell.wfsa` and `espell-epron.wfst`, what else can you do to decode `jprons.txt` (not necessarily to English words)? (No need to implement these)

4 Shannon Game and Entropy of English (optional)

1. Each team member plays the game once.[2] Report the entropy from each time (take a screenshot).
2. Explain how the entropy was calculated in this game (with equations).

Submit two files: an `hw2.zip` which includes `*.{wfst,wfsa,probs,py}`, and a separate `report.pdf`.

[2] Try `http://classes.engr.oregonstate.edu/eecs/spring2017/cs519-001/shannon.tgz` and run `appletviewer index.html`.