Natural Language Processing, Fall 2019, HW 3

Prof. Liang Huang

Due Wednesday Oct 30 at 11:59pm on Canvas (to HW3 Groups)

In HW2, we did POS tagging, English pronunciation and spelling, and Katakana-to-English backtransliteration, all using Carmel. In this HW, you will implement the famous Viterbi algorithm used by Carmel for all these decodings; in doing so, you will implicitly implement "composition" as well. In other words, you can view this HW as the "under-the-hood" version of HW2.

You will need again all the files we provided for HW2:

`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`.

In addition, you will also need `epron-jpron.probs` and `epron-jpron.wfst` from your HW2 submission. But in case you didn't get everything correctly, we have provided our reference solutions in

`http://classes.engr.oregonstate.edu/eecs/fall2019/cs539-001/hw3/hw3-data.tgz`.

(Note that all expected outputs are in the 5-bests, and only in a few cases the 1-bests are confused with similar-sounding words e.g., `SHEET` vs. `SEAT`, `SAVING` vs. `SHAVING`, `MAKE BOOK` vs. `MAC BOOK`, etc.).

1 Part-of-Speech Tagging (10 pts)

Redo the POS tagging examples by implementing a bigram Viterbi algorithm (name it `tagging.py`) that would output the most probable POS tag sequence (and its probability) for an input sentence. Verify your results with Carmel outputs from HW2. After finishing bigram tagging, please work on trigram tagging, which paves the way for Part 2. *This question is graded by completeness, not by correctness.*

2 Decoding Katakana to English Phonemes (35 pts)

Now implement your own Viterbi decoding in Python for HW2 Question 3.5 (jprons to eprons). Your command-line should be:

```
echo -e 'P I A N O\nN A I T O' | ./decode.py epron.probs epron-jpron.probs
```

with the result like: (note the input represents the Japanese katakana `PI A NO`, not the English word!)

```
P IY AA N OW # 1.489806e-08
N AY T # 8.824983e-06
```

For your convenience, we have converted `epron.wfsa` into a human-readable `epron.probs`, in the same format as `epron-jpron.probs` above, e.g.:

```
T S : </s> # 0.774355
T S : AH # 0.0333625
T S : Z # 2.92651e-07
```

which correspond to our intuition that sounds `T S` (words with `-ts`) are very likely to end a sentence or a word, and sounds `T S AH` are much less likely (but still noticeable in words like `WATSON` and `BOTSON`), but `T S Z` is (almost) completely unheard of (with a tiny prob here due to smoothing).

You only need to print the 1-best solution and its probability. Please

1. Describe your algorithm in English first.
2. Define the subproblem and recurrence relations.
3. Analyse its complexity.
4. Implement it in Python. Your grade will depend on both correctness and efficiency.

Note that each English phoneme corresponds to **one to three** Japanese phonemes (the PIANO example is 1-1 and is too simple, and the NIGHT/KNIGHT example is more interesting), so the Viterbi algorithm from the slides needs to be extended a little bit.

Compare your 1-best results with carmel's. Try your best to match them. Include a side-by-side comparison (hint: `diff -y`).

Hint: if you want to make sure your program is 100% correct with respect to Carmel, you can compare their results by:

```
cat jprons.txt | ./decode.py epron.probs epron-jpron.probs > results.my
cat jprons.txt | carmel -bsriIEQk 1 epron.wfsa epron-jpron.wfst 2>/dev/null \
               | awk '{for (i=1;i<NF;i++) printf("%s ", $i); printf("# %e\n", $NF)}' \
               >  results.carmel
diff -b results.my results.carmel
```

You should see no output if you did everything correctly. To generate a side-by-side comparison, replace `diff -b` by `diff -by`.

FYI, my not-very-optimized implementation is 60 lines of Python, and uses about ~~4x~~ **2x** time compared to Carmel. Your implementation will be graded in terms of both correctness and efficiency.

3 K-Best Output (15 pts) (`kbest.py`)

Now extend your Viterbi algorithm to output *k*-best sequences (and their corresponding probabilities). Implement it on ~~POS tagging and~~ Katakana-to-English. Verify the results with Carmel using the `diff` method above (`kbest.py` should agree with Carmel output for *k*=1..10). Efficiency is part of the grade. **Hint:** see Huang and Chiang (2005). Algorithm 2 is enough to receive full credit for this part.

4 Extra Credit: Decode Katakana to English WORDS (30 pts)

(*Do NOT attempt this problem until you finish everything else. This is the hardest problem in this course.*)

Now if you are to decode with HW2 approach using `eword.wfsa` and `eword-epron.wfst` and `epron-jpron.wfst`.

1. Describe your algorithm in English first.
2. Define the subproblem and recurrence relations.
3. Analyse its complexity.
4. Implement `decode_word.py`.

```
echo -e 'P I A N O N A I T O' | ./decode_word.py eword.probs eword-epron.data epron-jpron.probs
PIANO NIGHT # 1.856048e-11
```

Here `eword.probs` is just a human-readable format of `eword.wfsa`. Verify your results with Carmel on `jprons.txt`. Just 1-best is more than enough! **UPDATE:** My code is orders of magnitude faster than Carmel, thanks to the trie. (one diff: `BABYSITTER` is in `eword-epron.data` but not in `eword-epron.wfst`)