CS 539-001 Natural Language Processing, Fall 2019
EX 2: Language Models and Entropy

Instructions:

- Submit **individually** by Monday Oct 14, 11:59pm on **Canvas**. No late submission will be accepted.
- Only `ex2.zip` (which contains `report.pdf`) should be submitted.
- Worth **5%** of the final grade. Unlike EX1, this EX will be graded by performance.
- This EX also teaches you a little bit about the very useful tools `sed` and `awk`.

This Exercise asks you to train and use char-based n-gram language models. Please download `http://classes.engr.oregonstate.edu/eecs/fall2019/cs539-001/ex2/ex2-data.tgz` which contains

1. `train.txt`: training data for collecting statistics;
2. `dev.txt`: held-out data for fiddling smoothing parameters;
3. `test.txt`: test data for evaluation.
4. two example WFSAs: `ab.wfsa` and `uni.wfsa`.

There is also a **blind test** corpus which we do not reveal to you.

1 Training Language Models

1. Train unigram, bigram, and trigram character language models. These models must assign non-zero probability to any sequence. Store these models in Carmel's WFSA format. Evaluate your models on `text.txt`. Try improve your models. **Your grade will depend in large part on the entropies of your models on the blind test data.**

   Name your models: `unigram.wfsa`, `bigram.wfsa`, `trigram.wfsa`.

2. What are the corpus probabilities your three models assign to the test data? What are the respective (cross) entropies?

   (Hint: use

   ```
   cat <text> | sed -e 's/ /_/g;s/\(.\)/\1 /g' | awk '{printf("<s> %s </s>\n", $0)}' \
              | carmel -sribI <your_wfsa>
   ```

   for corpus probability. The `sed` command inserts a space between letters and replaces the original space by _, i.e., `it is` becomes `i t _ i s`.)

3. What are the final sizes of your WFSAs in states and transitions.

   (Hint: use `carmel -c <your_wfsa>`).

4. Include a description of your smoothing method. Your description should include the algorithms, how you used the held-out data, and what (if any) experiments you did before settling on your solution.

5. Include a sketch drawing of your 3-gram model in sufficient detail that someone could replicate it. Please consider this carefully – examine your drawing after you have drawn it and evaluate whether someone (not you) could build the same WFSA you have built. We will consider it in the same light.

Important:

- There is a special start symbol `<s>` and special end symbol `</s>` around every sentence, so that we can model for example the fact that letter `x` normally does not start a sentence, and the letter `z` normally does not end a sentence (we omit punctuations).

  Here is an example unigram WFSA on two characters (`ab.wfsa`):

  ```
  2
  (0 (1 <s> 1))
  (1 (1 a 0.8))
  (1 (1 b 0.1))
  (1 (2 </s> 0.1))
  ```

  Note that `<s>` is **not** a token in the language, while `</s>` is. That is why the probability of `<s>` is always 1, and the probability of `</s>` is normalized with other characters. We've also provided an example `uni.wfsa`.

- To ensure that your WFSA represents a legitimate probability distribution, you should normalize it with `carmel -Hjn wfsa > wfsa.norm` (`-j` for joint prob. model, i.e., WFSA-style instead of the conditional WFST style). We will do this (before testing your models) in any case. You can try this on the above sample and it will give you the same WFSA since it's already normalized.

- To ensure that your WFSA represents a legitimate probability distribution, you should have a single final state with no exiting transitions. The above sample also demonstrates this.

2 Using Language Models

Language models are often used for generation, prediction, and decoding in a noisy-channel framework.

1. Random generation from n-gram models.

   Use `carmel -GI 20 <your_wfsa>` to stochastically generate character sequences. Show the results. Do these results make sense? (For example, you can do the same on the above sample WFSA, and you will see the proportion of `a`:`b` is indeed about 8:1.)

2. Restoring vowels.

   Just as in HW1, we can decode text without vowels. Try doing that experiment on `test.txt` with the language models you trained. What's your command line? What are your accuracy results? Include the input file `test.txt.novowels` and the result files `test.txt.vowel_restored.{uni,bi,tri}`. (Hint: you can use `sed -e 's/[aeiou]//g'` to remove vowels).

3. Restoring spaces.

   Similarly, we can remove the spaces and try to restore them with the help of language models.

   Try doing that experiment on `test.txt` with the language models you trained. What's your command line? What are your accuracy results? Include the input file `test.txt.nospaces` and the result files `test.txt.space_restored.{uni,bi,tri}`. (Hint: you can use `sed -e 's/ //g'` to remove spaces).

   Finally, decode the following two sentences with your models:

   `therestcanbeatotalmessandyoucanstillreaditwithoutaproblem`

   `thisisbecausethehumanminddoesnotreadeveryletterbyitselfbutthewordasawhole.`

4. Which of the two decoding problems is easier? Write a paragraph of observations you made in these experiments.