Course Project
CSE 517: Natural Language Processing

University of Washington

Winter 2016
Due: March 9, 2016, 1:30 pm
Last update: February 3, 2016

1 Overview

Your project is to build a probabilistic language model. As discussed in lectures, a language model maps sequences of symbols to valid probabilities; using the chain rule, this equates to mapping every arbitrarily-long prefix (history) to a discrete probability distribution over symbols that can come next.

In this project, the alphabet ($\mathcal{V}$) is the set of all valid UTF-8 encodings of Unicode version 8.0 (i.e., $V = 120,737$). You can read about Unicode at `http://unicode.org/`. Note that this is a significantly larger alphabet than you had in the first assignment.

2 Teams

This project is meant to be completed in teams of three: one CSE Ph.D. student expecting to do a Ph.D. on an NLP topic, one CSE Ph.D. student expecting to do a Ph.D. on a different topic, and one student who's not doing a Ph.D. in CSE.

3 Specs

On execution, your command-line program should do whatever is necessary to load up your language model, then iteratively process a series of commands from standard input. Your system should only print anything in response to receiving a command. For example, it should not output instructions or other messages upon starting. This will break our grading scripts and you won't receive points.

Each command is a single-character code, in some cases followed by a single-character argument.

- $\circ c$: observe the next character c; i.e., append it to the history. If c is the stop symbol (U+0003), clear the history. Output anything you like that doesn't include a newline character, then a newline character. For diagnostic purposes, you might want to output the log probability of c given the history (before c was added to the history), or the characters your model thought were most probable before it observed c. It's up to you.

- `q`c: write to standard output the base-2 log-probability[1] of character c given the history, followed by a newline. The history does not change (do not assume this character is observed—it's just a query).
- `g`: randomly generate a character from the conditional distribution over the next character given the history, write it to standard output (followed, optionally, by anything you like that does not include a newline character, followed by a single newline character), and append the generated character to the history.
- `x`: exit.

If the text your system writes to standard out doesn't precisely meet these formatting instructions, it won't be parsed by our scripts and we will be unable to evaluate your submission. Please make sure your system outputs the correct number of newlines, characters, and numbers.

Note that $\mathcal{V}$ needs to include a stop symbol, so that your program can guess that the passage has ended. For this, we use U+0003 (end of text).

This interface allows us to check, at any point, that your probabilities sum to a value no greater than one. If they sum to something greater than one, you are cheating, and your model will suffer an infinite penalty.

Your program should take on the command line an integer, which should be used to seed the random number generator used for the `g` command. (We do not care which random number generator you use, but running the program twice with the same seed and set of commands should always give the same output.)

An even mixture of the different commands should be processed at a rate of at least 10 commands per second.

You have three goals in building your language model:

1. Assign high probability to naturally occuring text. We will evaluate your program by running real text through it (with a combination of the `o` and `q` commands) and calculating perplexity:

$$perplexity(\boldsymbol{c}_{1:I}) = 2^{-\frac{1}{I}\sum_{i=1}^{I}\log_2 p(c_i|\boldsymbol{c}_{1:i-1})}$$

   where I is the length of the text in characters (including the stop symbol at the end). The text we test your program on could be in any natural language encoded in $\mathcal{V}$.

2. Generate, with high probability, text that looks natural. We will evaluate your program based on its ability to convince *other* teams' programs that it is, in fact, natural.[2]

3. We will use your model to classify texts as "naturally occurring" or not. The naturally occurring texts could be any text in any natural language. The "unnatural" texts will come from other teams' language models. Your model will be used to rank a collection (half natural, half "unnatural") by perplexity scores, and we will measure how well your model separates natural from unnatural texts. (Showing that $p(c_{1:I} \mid natural)$ is monotonic in $p(natural \mid c_{1:I})$, under reasonable assumptions, is left as an exercise.) Note that this wasn't part of the evaluation of the first assignment.

4 Submitting

When you submit your program, you will upload a gzipped tarball named `project.tar.gz`. Once untarred, there should be a file named `run_language_model.sh` in the same directory as `project.tar.gz`

[1] Why base 2? This makes things easier for us. If your language provides $\log_b$, you can calculate $\log_2 x = \frac{\log_b x}{\log_b 2}$.

[2] You may be tempted to try to generate memorized data. If you do this, your score will suffer if we decide to mix up `o` and `g` commands!

(not a subdirectory). This file should be a bash script, allowing us to run (for example):

```
bash run_language_model.sh 999
```

(which seeds the random number generator with 999). An example of input to your program is:

```
ohoeolololooq.gx
```

which might generate output such as:

```
-1.301123
!
```

(*Update on Feburary 3*: the above output should be preceded by a series of newline characters, not shown. Please see the official input/output example for testing.)

That is, the probability that `.` is the symbol following `hello` is $2^{-1.301123} \approx 0.406$, and a randomly sampled symbol following `hello`, according to your distribution, is `!`.

We will issue instructions on how to submit your program before the deadline. Because we'll be executing your program, you are responsible for making sure that it will run on `attu`. We strongly advise that you use a widely-used programming language. Feel free to send questions.

Include a file with the name `README` that includes a natural language explanation of (i) how your language model works, (ii) the data you trained on, and (iii) any external libraries or tools you used. This is also the place to mention anyone who helped you (per the academic integrity policy). Be complete in your description, but concise. We expect your model to incorporate intuitions of natural language that you learned from the course, and support your design choices on model and data with experiments.

5 Using Existing Software

We have no ban on software you can package with your submission. The only restrictions are those imposed by `attu`, the server we'll be running your system on. We will allow five minutes from the time we run the bash script in your submission until your system is required to start responding to the queries, which will allow for a little set up. Please also note, as described on the discussion board, that you will have to assume we will not have a connection to the internet when running your submission, so having your code download and install packages is probably a bad idea.