Assignment 1
CSE 517: Natural Language Processing

University of Washington

Spring 2018
Due: April 11, 2018, 1 pm
Latest update: April 4, 2018

For this assignment, you will submit code that the course staff will run on the server `umnak.cs.washington.edu`. Make sure you have credentials to log in to this server; if you do not have a CSE account, fill out the quiz on Canvas called "Request for account to use umnak" and we'll get an account set up for you.

Your first assignment is nearly identical to the project; unlike the project, you must complete it on your own. Read carefully to make sure you notice the differences between this assignment and the project.

You are to build a probabilistic language model. 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 assignment, the alphabet ($\mathcal{V}$) is the set of all valid UTF-8 encodings of Unicode version 10.0 in the "basic multilingual plane." Hence $V = 65{,}424$. You can read about Unicode at `http://unicode.org/` and the basic multilingual plane at `https://en.wikipedia.org/wiki/Plane_(Unicode)#Basic_Multilingual_Plane`. This is a significantly smaller alphabet than is expected for your project. (This paragraph was updated on 4/4 to use Unicode 10.0, not Unicode 8.0.

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

- `o`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 currently-stored 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.

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

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. We provide example input and output files (on Canvas); please use these but also write your own tests. **It is your responsibility to make sure your submission runs exactly to the specification; we will not troubleshoot your code, and any problems will result in a major loss of points.**

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 you and 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.)

You have two 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* students' programs that it is, in fact, natural.

Note that we are not providing training data; we assume that you will construct your own, since natural language text is in such rich supply. This means you must decide how much effort to devote to (i) implementing your model, (ii) gathering data, and (iii) training your model. We encourage you to build your own development set to track perplexity as you proceed.

When you submit your program, you will upload a gzipped tarball. Once untarred, there should be a file named `run_language_model.sh`. 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:

```
ohoeolooq.gx
```

which might generate output such as:

(a series of newline characters)
`-1.301123`
`!`

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 `!`.

Submit your program as a gzipped tarball (`A1.tgz`) via Canvas. Because we'll be executing your program, you are responsible for making sure that it will run on our server, and that the tarball is completely self-contained. We strongly advise that you use a widely-used programming language that is already installed on `umnak.cs.washington.edu`. If you have specific requests for additional installations, we will consider them, but you should ask right away so no time is wasted.

Include in your submission a file with the name `README` that includes a paragraph of natural language text detailing (i) how your language model works, (ii) a description of 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). Clarification on 4/4: a big part of your grade is for writing a clear and concise description of what you did, so please take this part very seriously. This is more important than the quantitative performance of your method, for this assignment.