CS-5340/6340, Programming Assignment #1
Due: Friday, September 15, 2017 by 11:00pm

Your task for this assignment is to build N-gram language models. Your language modeling program should accept the following command-line arguments:

ngrams <training_file> -test <test_file>

For example, we should be able to run your program like this:

ngrams train.txt -test test.txt

Given these arguments, your program should create language models from the training file and apply the language models to the sentences in the test file, as described below.

Input Files

The *training_file* will consist of sentences, one sentence per line. For example, a training file might look like this:

I love natural language processing .
This assignment looks like fun !

You should divide each sentence into unigrams based solely on white space. Note that this can produce isolated punctuation marks (when white space separates a punctuation mark from adjacent words) as well as words with punctuation symbols that are still attached (when white space does not separate a punctuation mark from an adjacent word). For example, consider the following sentence:

"This is a funny-looking sentence" , she said !

This sentence should be divided into exactly nine unigrams:
(1) "This (2) is (3) a (4) funny-looking (5) sentence" (6) , (7) she (8) said (9) !

The *test_file* will have exactly the same format as the *training_file* and it should be divided into unigrams exactly the same way.

Building the N-gram Language Models

To create the N-gram language models, you will need to generate tables of frequency counts from the training corpus for unigrams (1-grams) and bigrams (2-grams). An N-gram should not cross sentence boundaries. All of your N-gram tables should be case-insensitive (i.e., "the", "The", and "THE" should be treated as the same word).

You should create three different types of language models:

(a) A unigram language model with no smoothing.
(b) A bigram language model with no smoothing.
(c) A bigram language model with add-one smoothing.

You can assume that the set of unigrams found in the training corpus is the entire universe of unigrams. We will not give you test sentences that contain unseen unigrams. So the vocabulary V for this assignment is the set of unique unigrams that occur in the training corpus.

However, we will give you test sentences that contain bigrams that did not appear in the training corpus. The n-grams will consist entirely of unigrams that appeared in the training corpus, but there may be new (previously unseen) combinations of the unigrams. The first two language models (a and b) do not use smoothing, so unseen bigrams should be assigned a probability of zero. For the last language model (c), you should use *add-one smoothing* to compute the probabilities for all of the bigrams.

For bigrams, you will need to have a special pseudo-word called "phi" (ϕ) as a beginning-of-sentence marker. Bigrams of the form "$\phi\ w_i$" mean that word w_i occurs at the beginning of the sentence. The pseudo-word ϕ should be included as a word in your vocabulary for the bigram language models. (Do not include ϕ as a word in your vocabulary for the unigram language model or include ϕ in the sentence probability for the unigram model.)

You should NOT use an end-of-sentence marker. The last bigram for a sentence of length k should represent the last 2 words of the sentence: "$w_{k-1}\ w_k$".

Computing Sentence Probabilities

For each of the language models, you should create a function that computes the probability of a sentence $P(w_1...w_n)$ using that language model. Since the probabilities will get very small, you must do the probability computations in log space (as discussed in class, also see the lecture slides). **Please do these calculations using log base 2.** If your programming language uses a different log base, then you can use the formula on the lecture slides to convert between log bases.

Output Specifications

Your program should print the following information for each test sentence. When printing the logprob numbers, please print **exactly 4 digits** after the decimal point. For example, print -8.9753864210 as -8.9754. The programming language will have a mechanism for controlling the number of digits that are printed. If P(S) = 0, then the logarithm is not defined, so print logprob(S) = undefined.

Please print the following information, formatted exactly like this:

S = <sentence>

Unsmoothed Unigrams, logprob(S) = #
Unsmoothed Bigrams, logprob(S) = #
Smoothed Bigrams, logprob(S) = #

For example, your output might look like this (the example below is not real, it is just for illustration!):

S = Elvis has left the building .

Unsmoothed Unigrams, logprob(S) = -9.9712
Unsmoothed Bigrams, logprob(S) = -12.2933
Smoothed Bigrams, logprob(S) = -8.4819

GRADING CRITERIA

We will run your program on the files that we give you as well as new files to evaluate the generality and correctness of your code. **So please test your program thoroughly!** Even if your program works perfectly on the examples that we give you, that does not guarantee that it will work perfectly on different test cases.

Please make sure that your program conforms exactly to the input and output specifications, or a penalty will be deducted! Programs that do not conform to the specifications are a lot more difficult for us to grade.

FOR CS-6340 STUDENTS ONLY! (30 additional points)

CS-6340 students should create an additional function that can automatically *generate* sentences using the **unsmoothed bigram language model**. Your program should accept an alternative command-line option "-gen" followed by a *seeds_file*, which will contain a list of words to begin the language generation process, following this format:

ngrams <training_file> -gen <seeds_file>

For example, we should be able to run your program like this:

ngrams train.txt -gen seeds.txt

When given this "-gen" option, your program should perform the language generation process described below. The *training_file* will have the same format described earlier. The *seeds_file* will have one word per line, and each word should be used to start the language generation process.

Creating an N-gram Language Generator

Your language generator should use the unsmoothed bigram language model to produce new sentences! Given a *seed word*, the language generation algorithm is:

1. Find all bigrams that begin with the seed word - let's call this set B_{seed}. Probabilistically select one of the bigrams in B_{seed} with a likelihood proportional to its probability.

   For example, suppose "crazy" is the seed and exactly two bigrams begin with "crazy": "crazy people" (frequency=10) and "crazy horse" (frequency=15).
   Consequently, $P(people \mid crazy) = \frac{10}{25} = .40$ and $P(horse \mid crazy) = \frac{15}{25} = .60$.
   There should be a 40% chance that your program selects "crazy people" and a 60% chance that it selects "crazy horse".

   An easy way to do this is to generate a random number x between [0,1]. Then establish ranges based on the bigram probabilities. For example, if $0 \leq x \leq .40$ then your program selects "crazy people", but if $.40 < x \leq 1$ then your program selects "crazy horse".

2. Let's call the selected bigram $B' = w_0\ w_1$ (where w_0 is the seed). Generate w_1 as the next word in your new sentence.
3. Return to Step 1 using w_1 as the new seed word.

Your program should stop generating words when one of the following conditions exists:

- your program generates one of these words: . ? !

- your program generates 10 words (NOT including the original seed word)
- B_{seed} is empty (i.e., there are no bigrams that begin with the seed word).

IMPORTANT: For each seed word, your language generator should **randomly generate 10 sentences** that begin with that word. Since each sentence is generated probabilistically, the sentences will (usually) be different from each other.

Output Specifications

Your program should print each seed word followed by a blank line and then the 10 sentences generated from that seed word. You should format your output like this:

Seed = <seed>

Sentence 1: <sentence>
Sentence 2: <sentence>
Sentence 3: <sentence>
Sentence 4: <sentence>
Sentence 5: <sentence>
Sentence 6: <sentence>
Sentence 7: <sentence>
Sentence 8: <sentence>
Sentence 9: <sentence>
Sentence 10: <sentence>

For example, your output might look like this:

Seed = Elvis

Sentence 1: Elvis has gone fishing for french fries .
Sentence 2: Elvis has left McDonald's !
Sentence 3: Elvis sings to eat peanut butter sandwiches ?
Sentence 4: Elvis has sandwiches and left the building .
Sentence 5: Elvis is alive and this sentence has exactly ten new words
Sentence 6: Elvis sings a lot .
Sentence 7: Elvis is buried at Graceland and likes peanut butter .
Sentence 8: Elvis has left and lived in Mississippi .
Sentence 9: Elvis sang rock and roll and sings ?
Sentence 10: Elvis has left !

GRADING CRITERIA

We will run your program on the files that we give you as well as new files to evaluate the generality and correctness of your code. **So please test your program thoroughly!** Even if your program works perfectly on the examples that we give you, that does not guarantee that it will work perfectly on different test cases.

Please make sure that your program conforms exactly to the input and output specifications, or a penalty will be deducted! Programs that do not conform to the specifications are a lot more difficult for us to grade.

SUBMISSION INSTRUCTIONS
(a.k.a. "What to turn in and how to do it")

Please use CANVAS to submit a gzipped tarball file named "ngrams.tar.gz". (This is an archived file in "tar" format and then compressed with gzip. Instructions appear below if you're not familiar with tar files.) Your tarball should contain the following 3 items:

1. The source code for your program. Be sure to include all files that are needed to compile and run your program!

2. A **README.txt** file that includes the following information:

   (a) what programming language and version you used (e.g., python2 or python3)

   (b) instructions on how to compile and run your code

   (c) which CADE machine you tested your program on
   (this info may be useful to us if we have trouble running your program)

   (d) any known bugs, problems, or limitations of your program

   REMINDER: your program *must* compile and run on the linux-based CADE machines! We will not grade programs that cannot be run on these CADE machines.

3. *CS-5340 and CS-6340 students:* submit a trace file called **ngrams-test.trace** that shows the output of your language model program when applied to the sample training and test files.

   CS-6340 students: submit an additional trace file called **ngrams-gen.trace** that shows the output of your program to generate new sentences when applied to the sample training and seed word files.

   The sample training, test, and seed word files are available in the Program #1 folder on CANVAS.

HELPFUL HINTS

TAR FILES: First, put all of the files that you want to submit in a directory called "ngrams". Then from the parent directory where the "ngrams" folder resides, issue the following command:

tar cvfz ngrams.tar.gz ngrams/

This will put everything inside the "ngrams/" directory into a single file called "ngrams.tar.gz". This file will be "archived" to preserve any structure (e.g., subdirectories) inside the "ngrams/" directory and then compressed with "gzip".

FYI, to unpack the gzipped tarball, move the ngrams.tar.gz to a new location and issue the command:

tar xvfz ngrams.tar.gz

This will create a new directory called "ngrams" and restore the original structure and contents.

For more general information on the "tar" command, this web site may be useful: https://www.howtogeek.com/248780/how-to-compress-and-extract-files-using-the-tar-command-on-linux/

TRACE FILES: You can generate a trace file in (at least!) 3 different ways: (1) print your output to a file called `ngrams-test.trace`, (2) print your output to standard output and then pipe it to a file (e.g., `ngrams training.txt -test test.txt > ngrams-test.trace`), or (3) print your output to standard output and invoke the unix *script* command before running your program. The sequence of commands to use is:

```
script ngrams-test.trace
ngrams training.txt -test test.txt
exit
```

This will save everything that printed to standard output during the session to a file called `ngrams-test.trace`.