Homework 2 - Language Models

Due Wednesday by 9pm **Points** 100 **Submitting** an external tool
Available after Sep 24 at 8am

In this assignment, we'll explore the statistical approach to language modeling using n-gram language models.

Turn in:

- lm.py
  - We will be running your code with the command:
    - python3 lm.py berp-training.txt hw2-test.txt hw2-my-test.txt
    - Note! Some of the packages that we use aren't supported alongside python 3.8 on the backend for autograding, so this will be python version 3.6.9
  - We will be running unit tests on lm.py. You should not have any code that is outside of classes, functions or an **if __name__ == "__main__":** conditional guard.
  - Submit your work via Gradescope as a python script. (note! Do not submit an ipython notebook/jupyter notebook file)
- hw2-my-test.txt (containing your out-of-domain test set)

Learning objectives:

- Fully implement a foundational statistical model
- Deal with live data, including addressing issues of unknown n-grams and words
- Testing testing testing!

Allowed python modules

- you may **not** use nltk for this assignment
- you may use numpy, matplotlib, all built-in libraries like math, collections, sys, etc
- if you have a question about a particular library, please ask or post on piazza!
  - (if you think that using a certain library would make the problem very easy, then you probably aren't allowed to use it)

Provided files

- Starter code: lm_starter.py (https://northeastern.instructure.com/courses/85010/files/10531224/download?download_frd=1)
- Mini unit tests: test_minitrainingprovided.py (https://northeastern.instructure.com/courses/85010/files/11370198/download?download_frd=1)
- Training data:
  - Mini data: iamsam.txt (https://northeastern.instructure.com/courses/85010/files/10531227/download?download_frd=1) , iamsam2.txt (https://northeastern.instructure.com/courses/85010/files/10531228/download?download_frd=1) , unknowns.txt (https://northeastern.instructure.com/courses/85010/files/10531229/download?download_frd=1) , unknowns_mixed.txt (https://northeastern.instructure.com/courses/85010/files/10531246/download?download_frd=1)
  - Full data: berp-training.txt (https://northeastern.instructure.com/courses/85010/files/10531225/download?download_frd=1) , berp-training-tri.txt (https://northeastern.instructure.com/courses/85010/files/10531223/download?download_frd=1) , berp-training-four.txt (https://northeastern.instructure.com/courses/85010/files/10531226/download?download_frd=1)
- Testing data:
  - berp: hw2-test.txt (https://northeastern.instructure.com/courses/85010/files/11370504/download?download_frd=1) , hw2-test-tri.txt (https://northeastern.instructure.com/courses/85010/files/11370502/download?download_frd=1) , hw2-test-four.txt (https://northeastern.instructure.com/courses/85010/files/11370501/download?download_frd=1)

Clarifications/hints

Based on questions during office hours, we've added the following clarifications/hints.

- make sure that you are counting <s>, </s>, and <UNK> just like any other tokens
- defaultdict (as opposed to a regular dict or a Counter) doesn't handle floats well—recommend not to use these for probabilities
  - We **do** suggest that you use the Counter class from the collections module, but regular dicts are fine as well.
- you'll need to take two passes through the training data—once to see which words to replace with <UNK> and the second time to actually collected your unigram, bigram, etc counts
- do not add <UNK> to your vocabulary if no words in your training occurred only once

- For the autograder tests—the number that it expects is on the left hand side and the number that it got is on the right hand. For example, if it gives you the following output, this means that your score is producing 0.066666666666667 when it should get 0.1.
  - Test Failed: 0.1 != 0.06666666666666667 within 3 places : tests probability of <s> flamingo, trained on iamsam2.txt
- You are dealing with a lot of data—we'll run any implementations that cause the autograder to timeout locally, but if you're getting timeouts, think about ways that you can reduce complexity (e.g. look at any double & triple-nested for loops you might have and try to eliminate them).
- When generating sentences, you may find the **numpy random.choice (Links to an external site.) (https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html)** function useful.
- Do not use Laplace smoothing for sentence generation.

Set-up 1: Training data

We'll be using the Berkeley Restaurant Corpus as our training data. There are around 7000 sentences in the full corpus (I'll reserve some for the test set). The data is organized as one sentence per line with each sentence beginning with an <s> tag and ending with a </s> tag. As in the following:

<s> i'd like to go to a fancy restaurant </s>

<s> tell me about le bateau ivre </s>

<s> dinner please </s>

The corpus has been lower-cased. For the purposes of this assignment, leave the various possessive markers etc alone. That is, just treat tokens like "i'd" as a single word.

You'll perform tokenization by using the python string **.split()** function without any parameters. (Do NOT use .split(" ")!)

Set-up 2: What your program will do

Your program will instantiate 2 language models, one unigram model and one bigram model, both using Laplace smoothing. With each model, you will do the following tasks (do all tasks with both models). See the later instructions for detailed notes on both parts.

1. Display 50 generated sentences from this model using Shannon's method. (see clarification about generation + Laplace smoothing)
2. Score the probabilities of the provided test sentences and display the average and standard deviance of these sentences.
   1. once for the provided test set
   2. once for the test set that you curated

Example:

Model: unigram, laplace smoothed

Sentences:

…

test corpus: name-of-first-test-file.txt

# of test sentences: FILL ME IN

Average probability: FILL ME IN

Standard deviation: FILL ME IN

test corpus: name-of-second-test-file.txt

# of test sentences: FILL ME IN

Average probability: FILL ME IN

Standard deviation: FILL ME IN

Model: bigram, laplace smoothed

Sentences:

…

test corpus: name-of-first-test-file.txt

# of test sentences: FILL ME IN

Average probability: FILL ME IN

Standard deviation: FILL ME IN

test corpus: name-of-second-test-file.txt

# of test sentences: FILL ME IN

Average probability: FILL ME IN

Standard deviation: FILL ME IN

Your program will have 60 auto-graded points that judge the correct implementation of part 1. The remaining 40 points will be manually graded, split as follows:

- 10 points: overall program functionality
- 15 points: part 2 - sentence generation

- 10 points: part 3 - find your own test set
- 5 points: overall comments and style

1: Build an n-gram language model

You need to develop an n-gram model that could model any order n-gram, but that we'll be using specifically to look at unigrams and bigrams. Specifically, you'll write code that builds this language model from the training data and provides functions that can take a sentence in (formatted the same as in the training data) and return the probability assigned to that sentence by your model.

For **unknown words,** you should use the technique described in class and in the book that does not depend on having a vocabulary beforehand. From your training data, replace any token with a frequency of 1 with the token <UNK>; collect statistics on that token as with any other word and use those counts whenever an unknown word is encountered during the evaluation phase.

For **smoothing counts,** your model has the option to use Laplace/add-1 smoothing. Note that add-1 smoothing happens after the <UNK> pre-processing. Therefore, you should have counts for things like (eat, <UNK>) and (<UNK>, on) and the like. These will come into play when you encounter unknown words during evaluation.

You'll be implementing a LanguageModel **class in python (https://docs.python.org/3/tutorial/classes.html)**. We've left skeletons of the methods that you must implement in the starter code. You are always welcome to add more methods to the class or outside helper functions as appropriate. We recommend taking a look at previous work that you've done in lecture notebooks and quizzes in particular.

This task consists of implementing the **__init__**, **train**, and **score** methods.

2: Implement Sentence Generation

For this task, you'll implement sentence generation for your Language Model using Shannon's method.

Start by generating the <s> token, then sampling from the n-grams beginning with <s>. Stop generating words when you hit an </s> token.

- Note 1: when generating sentences for unigrams, do not count the pseudoword "<s>" as part of the unigram probability mass after you've chosen it as the beginning token in a sentence.
- Note 2: All unigram sentences that you generate should start with one <s> and end with one </s>
- Note 3: For n-grams larger than 1, the sentences that you generate should start with n - 1 <s> tokens. They should end with n - 1 </s> tokens. Note that for n > 2, there should be no chance of generating a word other than </s> after your first </s> token. You may "tack these on" the end if you would like.

- Note 4: you should not generate novel (unseen) n-grams using this method of sentence generation.
- Note 5: you should expect to see sentences that are just <s> </s> in the unigram case.
- Note 6: you should expect to see sentences with <UNK>.

3: Create an out-of-domain test set & measure performance (10 points)

Find or curate an out-of-domain test set of 100 sentences. You'll need to make sure that they are formatted consistently with the provided test set. Each sentence should begin with <s> and end with </s>, and should not end with punctuation.

You may need to do some curation of this by hand.

For each test set, score all sentences in the test set, then compute two numbers—the average and the standard deviation of the set, then report these numbers to the user. Feel free to use your favorite python library to calculate these numbers.

4: Required for CS 6120/Extra Credit 1 for CS 4120 - Flexible model (up to +3)

We will only test your models for unigrams and bigrams, however, you should always strive to write flexible and extensible code.

Make your Language Model class work for any integer value of n for n >= 1. We have provided training data from tri-grams and 4-grams but you may want to produce your own training data for higher-order n.

If you are in CS 6120, not implementing a flexible model will result in a 15 point deduction. We will be testing your models against higher-order n-grams. If your model times out the autograder, this will result in an automatic 5 point deduction, even if we are able to run the full tests locally.

5: Required for CS 6120/Extra Credit 2 for CS 4120 - Perplexity (up to +5)

Add a method with the following signature to your LanguageModel class:

```
def perplexity(self, test_sequence):
  """Measures the perplexity for the given test sequence with this trained model.
        As described in the text, you may assume that this sequence
        may consist of many sentences "glued together".
  Parameters:
    test_sequence (string): a sequence of space-separated tokens to measure the perplexity of
  Returns:
    float: the perplexity of the given sequence
```

"""

Modify your program so that at the end it prints out the perplexity of **the first 10 sentences (glued together) of** the two passed-in testing files. hw3-test.txt and hw3-my-test.txt.

E.g.

Perplexity for 1-grams:

hw2-test.txt: [your calculated number here]

hw2-my-test.txt: [your calculated number here]

Perplexity for 2-grams:

hw2-test.txt: [your calculated number here]

hw2-my-test.txt: [your calculated number here]

If you are in CS 6120, not implementing perplexity correctly will result in a 7.5 point deduction.

CS 4120/6120 | Fall 2021

NAME	STATUS	RELEASED
CS 6120 - Research Presentation	No Submission	SEP 28
Homework 2 - Language Models	No Submission	SEP 23 LATE DUE DATE:
Homework 1.2 - Test a Chatbot	20.0 / 20.0	SEP 17 LATE DUE DATE:
Homework 1.1 - How many words do you know?	39.0 / 40.0	SEP 14 LATE DUE DATE: