ASSIGNMENT 2
N-GRAM LANGUAGE MODELS

CS 478 NATURAL LANGUAGE PROCESSING (FALL 2024)
https://nlp.cs.gmu.edu/course/cs478-fall24/

OUT: Sept 04, 2024

DUE: Sept 18, 2024

TOTAL CREDITS: 100 Points

Your name: Arron Birhanu

Your GID: G01315277

Shareable link to your notebook:
https://drive.google.com/file/d/15mTuuPw9Em4f7s9xCiwNNGD7HdiCQCJV/view?usp=sharing

Academic Honesty: Please note that students should complete the assignment independently. While you may discuss the assignment with other students, **all work you submit must be your own!** In addition, if you use any external resources for the assignment, you must include references to these resources at the end of this PDF. However, **you are NOT allowed to use AI assistants such as ChatGPT and Claude to complete your assignment; however, using them for generic concept understanding is okay.**

Overview and Submission Guideline: In this project, you will implement a bigram language model (Part 1), evaluate its performance (Part 2), and sample texts from it (Part 3). The assignment also comes with a Jupyter notebook (`https://jupyter.org/`), which can be completed on your laptop/computer locally or on the cloud by using Google Colab (`https://colab.research.google.com/`), although using Colab requires extra effort in setting up the data access. **Complete the notebook and save your edits *(make sure to save the cell outputs as well)*. Create a shareable link to your notebook and copy it to the line above.**

How to fill out this PDF? In most answer blanks, you only need to provide the execution output of your code implementation. There are only a few blanks that ask you to provide your code implementation for ease of grading. When you complete the PDF, submit it to Gradescope.

Important Notes: (1) Please do NOT modify existing code/markdown cells in the notebook. (2) While there may have been well-packed Python tools for producing and evaluating language models, you should not use them. Instead, you are expected to implement the language model from scratch, following the instructions in the notebook. The only exceptions are commonly used general-purpose Python libraries such as `numpy` and `collections`, as they do not directly produce or evaluate the language model. Please consult the grader or the instructor if you have questions.

Part 1: Language Model Construction (50 points)

Your first task is to implement the bigram language model.

Step 1 – Construct a vocabulary (20 points): To get started, you will first need to construct a vocabulary based on the `sents_train` corpus. Also, don't forget the special start-of-sentence (<s>) and end-of-sentence (</s>) tokens – your LM should eventually be able to model $p(w|\texttt{<s>})$ (i.e., how to start a sentence) and $p(\texttt{</s>}|w)$ (i.e., when to stop a sentence).

Now, show the size of your vocabulary (i.e., how many distinct words in your vocab, including UNK):

Your Code Output

Vocabulary size (including UNK): 4183

If we use this vocabulary to index sentences in `sents_train`, what are the most frequent words (including UNK)? Show the top 10 word types with their counts, one pair in a row.

Provide the top 10 word types and their counts below:

Your Code Output

```
The most frequent 10 word types with counts:
,, 9129
<s>, 6173
</s>, 6173
., 5513
to, 4115
the, 3805
and, 3748
of, 3386
UNK, 2931
I, 2518
```

Step 2 – Build the bigram LM (30 points): Next, implement the bigram LM $p(w_t|w_{t-1})$ (with add-one smoothing) based on the `sents_train` corpus.

First, accumulate the bigrams in `sents_train` and create a bigram counter.

For how many times in `sents_train` does a sentence start with the word "I"?

Your Code Output

Sentences in sent_train starts with the word 'I' for times: 502

Next, create the bigram language model (with add-one smoothing). **Copy your code below.**

Your Code

```
import time

start_time = time.time()

# TODO: create your bigram LM
model_p = dict() # a dict of {(w_t-1, w_t): count}

for w_tm1 in vocabulary:
    if w_tm1 == "</s>":
        continue
    for w_t in vocabulary:
        if( (w_tm1 + " " + w_t) in bigram):
            model_p[(w_tm1, w_t)] = ( bigram[w_tm1 + " " + w_t] + 1) / ( (
    frequency[w_tm1]) + len(vocabulary))
            continue
        model_p[(w_tm1, w_t)] = 1 / (frequency[w_tm1] + len(vocabulary) )

# Do not modify code after this line
end_time = time.time()
print("Spent %s for the bigram LM construction." % time.strftime("%Hh%Mm%Ss",
    time.gmtime(end_time-start_time)))

# The following lines verify the validity of the probability distribution -- it
    takes a while, please wait.
# If you receive an assertion error, then your implementation could be
    problematic.
for w_tm1 in vocabulary:
   if w_tm1 == "</s>": # not needed
       continue
   pr_mass = 0 # sum over different w's of p(w|w_tm1)
   for w_t in vocabulary:
       pr_mass += model_p[(w_tm1, w_t)]
   assert isclose(pr_mass, 1.0), "Probability mass of %s should sum to 1" % w_tm1
     # sum should equals to 1
```

Now, can you show the most frequent 10 starting words (i.e., words following <s>) with their probabilities to three decimal places (e.g., 0.123)?

Provide the most frequent 10 starting words and their probabilities below:

Your Code Output

```
Your Answer Here; one word and probability in each row.
", 0.123214
I, 0.048571
She, 0.035921
He, 0.028486
It, 0.022692
The, 0.022306
Emma, 0.017285
But, 0.014774
Mr, 0.014002
You, 0.013422
```

Part 2: Evaluation of a Language Model (30 points)

Your second task is to evaluate the bigram language model that you constructed in Part 1. Please follow the instructions and implement the instructed functions.

Step 1 – Calculate the log2 probability of a test sentence (20 points): Provide your `log2_P` output for the first sentence in `sents_test`.

Your Code Output

log2_P returns: -145.7388770622259

Step 2 – Calculate the per-word cross entropy (10 point): Provide your `H` function's output below:

Your Code Output

H returns: 9.715925137481726

Step 3 – Implement the perplexity metric: Provide your `ppl` output below:

Your Code Output

Perplexity: 840.9785174373525

Part 3: Sampling from a Language Model (20 points)

Your third task is to sample texts from the bigram language model you built in Parts 1 and 2. You will need to implement two sampling methods, greedy decoding and sampling by LM next-word distribution, each worth 1 point.

Step 1 – Greedy decoding (10 point): Provide your code below:

Your Code

```
# greedy decoding
sent = [] # used to store the generated words in your sentence

# TODO: Please input your code below:
iterator = 0
currWord = '<s>'
tempWord = ''

while(iterator!=max_len-1):
    maxPr = 0
    for keys in vocabulary:
        if(None == bigram.get(currWord + ' ' + keys)):
            continue
        if(maxPr < bigram[currWord + ' ' + keys]):
            tempWord = keys # Select next maximum
            maxPr = bigram[(currWord + ' ' + keys)]

    sent.append(currWord)
    if(tempWord == '</s>'):
        break # We have our own sentence ender made outside this while loop
    currWord = tempWord
    iterator += 1

if(iterator!=max_len): # If there's space left to expand the sentence, add the
    ending token
    sent.append('</s>')
# print the decoded sentence
print(sent)
```

Provide the greedily decoded sentence by your implementation below:

Your Code Output

Your Answer Here: <s> " I am sure , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , and the UNK , </s>

Step 2 – Sampling by distribution (10 point): Provide your code below:

Your Code

```
np.random.seed(1234) # for grading purpose, do not change the random seed

sent = [] # used to store your sentence

# TODO: Please input your code below:
iterator = 0
currWord = '<s>'
tempWord = ''

while(iterator!=max_len-1):

    wordsF = {}#the probabilty distribution of next word given currWord
    for w_t in vocabulary:
        wordsF[w_t] = model_p[(currWord, w_t)]



    sent.append(currWord) #add it to our sentence
    currWord = np.random.choice(list(wordsF.keys()), p = np.array(list( wordsF.
    values() )) )#take a sampling
    currWord = currWord.item() # Turn it back into a string


    if(currWord == '</s>'):
        break # We have our own sentence ender made outside this while loop
    iterator += 1

sent.append('</s>')
# print the decoded sentence
print(sent)
```

Provide your sampled sentence below:

Your Code Output

Your Answer Here: <s> When unaccountable ceremonious preparations offended themselves wishes Little related sweep praise wholly chair hurried sex lent attempts by unfeeling concealing established respected laughing sinking agreeably thankful saved secresy Campbell indisputable pages arrive perceiving air grounds relations knowledge trifles ,"– tear however bear respect eternal distinguished encouraging ;– suspense </s>

Post-Assignment Questions (Required)

Q1: Did you use any resources (e.g., online tutorials, blog posts, etc.) while completing this assignment? If you did, please list them below:

Your Answer

Your Answer Here: I spent most of my time outside this assignment on youtube trying to relearn and understand niche concepts of LM (i.e. Add one smoothing). They helped me gain a grounding for moving forward with Part 2.

Q2: Did you use AI assistants (e.g., ChatGPT) while completing this assignment? If you did, please describe how it was used. Note that you are NOT allowed to use the AI assistant's code output as yours, but it is okay to use it to assist you in generic concept understanding. I looked up python documentation to better understand the args and syntax for completing the tasks.

Your Answer

Your Answer Here: I learned the core concepts of selecting the next word by a probability distribution. I also learned the specifics of token usage in next word processing by generating sentences with frequent tokens and using them to debug. This was more efficient than working with the vast corpus presented to us in the project zip file and the small A,B,C 2D array. I also used chatGPT to turn my python output (in list form) into a sentence by asking to put the elements in the form of a sentence.