CS 5134/6034 Natural Language Processing
Assignment #2
Due: Monday, Sept 30, 2024 by 9am (before class)

*This assignment contains two parts: a written part and a programming part. Please provide your answer under each question and submit this pdf (or a scanned pdf if you prefer handwriting) to Gradescope for the written part, **together with** one python script file for the programming part.*

I. Written (30 pts)

1. (6 pts) Use the Dictionary and Morphology Rules shown below to answer this question.

Dictionary	
appropriate	ADJ
infect	VERB
human	NOUN
humane	ADJ
smoke	NOUN, VERB

Rule ID	Prefix	Suffix	Replace Chars	Root POS	Derived POS
R1		ant		VERB	NOUN
R2		ation	e	VERB	NOUN
R3		er		VERB	NOUN
R4		er	e	VERB	NOUN
R5		ier	y	ADJ	ADJ
R6		ize		NOUN	VERB
R7		ly		ADJ	ADV
R8		ness		ADJ	NOUN
R9		s		NOUN	NOUN
R10		s		VERB	VERB
R11		y	e	NOUN	ADJ
R12		y		NOUN	ADJ
R13	de			VERB	VERB
R14	dis			VERB	VERB
R15	in			ADJ	ADJ

For each word given below, list **ALL** of the derivations that are possible using the Dictionary and Morphology Rules shown above. Provide each derivation in the following format:

WORD=[given below]   POS=____   ROOT=____   PATH=____

POS should be the part-of-speech that would ultimately be assigned to the given word. ROOT should be the word ultimately found in the dictionary during the application of morphology rules. The PATH should begin with the rule that was applied to the word found in the dictionary and then list the other rules in the order that they were subsequently applied. Be sure to list ALL legal derivations. If no derivations are possible for a word, then answer NO DERIVATIONS.

(a) smokier

(b) inappropriateness

(c) humanly

(d) dehumanization

(e) disinfects

(f) disinfectants

2. (10 pts) Consider the 9 training documents below, where each document consists of one sentence from (imaginary) restaurant reviews.

Class	Document
NEG	terrible food and slow service
NEG	so expensive and incredibly bad service
NEG	ok food but too expensive
NEG	service was so slow
POS	incredibly good food
POS	so tasty
POS	great food and not too expensive
POS	service a bit slow but incredibly great food
POS	incredibly tasty and ok service

(a) Based on the training corpus above, compute the following probabilities. Do not perform smoothing. **Leave your answers in fractional form!**

   i. P(NEG)

   ii. P(POS)

   iii. P("incredibly" | NEG)

   iv. P("incredibly" | POS)

   v. P("service" | NEG)

   vi. P("service" | POS)

(b) Using the Naive Bayes classification algorithm, determine which Class (NEG or POS) would be assigned to the test document: "incredibly expensive but ok". **Show ALL of your work! You will get zero credit if you only give a class name without showing the work.**

3. (14 pts) Tom and Jerry each labeled 10 newspaper articles (D1-D10) with respect to 3 categories: **Arts (A)**, **Finance (F)**, and **Politics (P)**. Their labels are shown below.

Document	Tom	Jerry
D1	P	A
D2	A	A
D3	A	A
D4	F	F
D5	P	P
D6	P	A
D7	F	F
D8	A	P
D9	P	P
D10	P	F

Show all your work, including the numerator and denominator of fractions! You will not get credit if you <u>only</u> show the final decimal number as the answer to a question.

(a) Compute the inter-annotator agreement between Tom and Jerry's labels using Cohen's Kappa (κ) statistic.

(b) Compute the Accuracy of Tom's labels when treating Jerry's labels as the gold standard.

(c) Compute the Recall and Precision of Tom's labels for the **Arts** category when treating Jerry's labels as the gold standard.

(d) Compute the Recall and Precision of Tom's labels for the **Finance** category when treating Jerry's labels as the gold standard.

(e) Compute the Recall and Precision of Tom's labels for the **Politics** category when treating Jerry's labels as the gold standard.

(f) Compute the macro-averaged Recall and Precision, as well as micro-averaged Recall and Precision of Tom's labels when treating Jerry's labels as the gold standard.

(g) Imagine a trivial system that assigns every document to the **Arts** category. Compute the system's Recall and Precision for the **Arts** category when treating Jerry's labels as the gold standard.

II. Programming (70 pts)

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

```
python3 ngrams.py [training file] [test file]
```

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

```
python3 ngrams.py train.txt 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 (we show white space explicitly using ␣):

"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 Language Models

To create the 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 all unique unigrams that occur in the training corpus, including punctuation marks.

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 "$<$s$>$" as a beginning-of-sentence symbol. Bigrams of the form "$<$s$>$ w_i" mean that word w_i occurs at the beginning of the sentence. Do NOT include $<$s$>$ as a word in your vocabulary for the unigram language model or include $<$s$>$ in the sentence probability for the unigram model.

For simplicity, just use the unigram frequency count of w_{k-1} to compute the conditional probability $P(w_k \mid w_{k-1})$. (This means you won't have to worry about cases where w_{k-1} occurs at the end of the sentence and isn't followed by anything.) For example, just compute $P(w_k \mid w_{k-1}) = \text{count}(w_{k-1}w_k) / \text{count}(w_{k-1})$.

You should NOT use an end-of-sentence symbol. The last bigram for a sentence of length n should represent the last 2 words of the sentence: "$w_{n-1}w_n$".

For each of the language models, you should create a function that computes the probability of a sentence $P(w_1 \ldots w_n)$ 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.

Output Specifications

Your program should print the following information to standard output 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". Be sure to use a function that rounds, not truncates. For example, 1.33337 should be rounded to 1.3334 . You should only do the rounding as the **last step** in your calculations, just before printing. Do not round numbers during intermediate calculations or your final results will not be fully accurate. Print an empty line between different sentences.

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 just for illustration):

```
S = Elvis has left the building .
Unsmoothed Unigrams, logprob(S)= -7.5025
Unsmoothed Bigrams, logprob(S) = undefined
Smoothed Bigrams, logprob(S) = -12.1393

S = Who ?
Unsmoothed Unigrams, logprob(S)= -2.9527
Unsmoothed Bigrams, logprob(S) = 0.0000
Smoothed Bigrams, logprob(S) = -1.2333
```

Grading Criteria

1. **Please use Gradescope to submit one source code named "ngrams.py" and one pdf named "a2.pdf" (written part)**. We will run your program on the 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, this does not guarantee that it will work perfectly on different test cases.

2. Please exactly follow the formatting instructions specified in this assignment and use the correct answer files "trace[#].txt" to make sure that your program conforms exactly to the input and output specifications. **You will get a 0 grade if you fail to follow the specifications.**

When you submit your code on Gradescope, you can get an instant score from Autograder based on the examples given to you so you can make sure your output has the correct format and "get a sense of" what grades your program may receive. After the due date, we will use new test cases to re-evaluate your program.

3. You can **NOT** use any external software packages or dictionaries to complete this assignment except for **Python3.9 Standard Library**.[1] For example, libraries for general I/O handling, math

---

[1]`https://docs.python.org/3.9/library/index.html`

functions, and regular expression matching are ok to use. The Gradescope Autograder will raise an error and give you zero when you import external packages.

4. You can **NOT** use any generative AI tools such as ChatGPT for this homework. All submitted code **must be your own**.