CS 585 – Fall 2023 – Homework 3 (100 total points)

Due Wednesday October 25, 11:59pm

GOALS

- Apply language model concepts to an NLP dataset
- Gain experience working with sequential text data in python
- Revisit familiar NLP tasks (sentiment analysis & text categorization) with language models

DATA

For this homework you will use the same **Stanford Sentiment Treebank** dataset that you used in Homework 1. You will only need to read in the file "train.tsv".

Here is this link posted in Homework 1 for GLUE datasets: Tasks Download page:

You may not alter the SST train.tsv file in any way prior to reading it into your notebook. Note that the "Getting Started" notebook from Homework 1 shows one way to read this file.

TOOLS

In this homework you may use functions from open-source python library including nltk, numpy, and scikit-learn. You may use other open-source libraries not listed, so long as you follow homework directions.

You should use Python to complete this homework assignment. Other programming languages will not be accepted

WHAT TO SUBMIT

Please upload or submit the following in Blackboard:

- For Problems 1 through 7, please upload to Blackboard:
    - One Jupyter notebook (.ipynb file) with cell output, showing your work for both datasets. Name this file "HW3_[CWID]_[LastName].ipynb".
    - A PDF copy of the **exact same notebook** (same code and same output).
    - You will **lose points** if you do not submit both of these files. Please do not zip or compress files before you upload.
- For Problem 8: Enter your written answer in Blackboard with your HW submission.

PROBLEM 1 – Reading the data (5 pts)

- Read in file "train.tsv" from the Stanford Sentiment Treebank (SST) as shared in the GLUE task. (See section "DATA" above.)
- Next, split your dataset into **train, test,** and **validation** datasets with these sizes (Note that 100 is a small size for test and validation datasets; it was selected to simplify this homework):
    - Validation: 100 rows
    - Test: 100 rows
    - Training: All remaining rows.
- Review the column "label" which indicates positive=1 or negative=0 sentiment. What is the prior probability of each class on your **training** set? Show results in your notebook.

PROBLEM 2 – Tokenizing data (10 pts)

- Write a **function** that takes a sentence as input, represented as a string, and converts it to a tokenized sequence padded by **start and end symbols.** For example, "hello `class`" would be converted to:
    - `['<s>', 'hello', 'class', '</s>']`
- Apply your function to all sentences in your training set. Show the tokenization of the first sentence of your training set in your notebook output.
- What is the **vocabulary size** of your training set? Include your start and end symbol in your vocabulary. Show your result in your notebook.

PROBLEM 3 – Bigram counts (10 pts)

- Write a **function** that takes an array of tokenized sequences as input (i.e., a list of lists) and counts **bigram** frequencies in that dataset. Your function should return a two-level dictionary (dictionary of dictionaries) or similar data structure, where the value at index $[w_i][w_j]$ gives the frequency count of bigram (w_i, w_j). For example, this expression would give the counts of the bigram "academy award":

```
bigram_counts["academy"]["award"]
```

- Apply your function to the output of problem 2. You should build one counter that represents all sentences in the training dataset.
- Use this result to show how many times a sentence starts with "the". That is, how often do you see the bigram ("<s>","the") in your training set? Show results in your notebook.

PROGRAMMING HINTS:

- You can use the function **nltk.bigrams** to convert a sequence to bigrams, but you are not required to do so.
- In python, you can use function "dict.get(key, default)" to return the value "default" if "key" is not present in a dictionary.

PROBLEM 4 – Smoothing (20 pts)

- Write a **function** that implements formula [6.13] in that E-NLP textbook (page 129, 6.2 Smoothing and discounting). That is, write a function that applies smoothing and returns a (negative) log-probability of a word given the previous word in the sequence. It is suggested that you use these parameters:
    - The current word, w_m
    - The previous word, w_{m-1}
    - bigram counts (output of Problem 3)
    - alpha, a smoothing parameter
    - vocabulary size
- Using this function to show the log probability that the word "academy" will be followed by the word "award". Try this with alpha=0.001 and alpha=0.5 (you should see very different results!). Show your results in your notebook.

PROGRAMMING ALTERNATIVE: If you are familiar with python classes, you may build a LanguageModel class that is initialized with the above parameters and implements formula [6.13] as a member function.

PROBLEM 5 – Sentence log-probability (10 pts)

- Write a **function** that returns the log-probability of a sentence which is expected to be a negative number. To do this, assume that the probability of a word in a sequence **only depends on the previous word**. It is suggested that you use these parameters:
    - A sentence represented as a single python string
    - bigram counts (output of Problem 3)
    - alpha, a smoothing parameter
    - vocabulary size
- Use your function to compute the log probability of these two sentences (Note that the 2nd is not natural English, so it should have a lower (more negative) result that the first):
    - "*this was a really great movie but it was a little too long.*"
    - "*long too little a was it but movie great really a was this.*"

PROBLEM 6 – Tuning Alpha (10pts)

Next, use your validation set to select a good value for "alpha".

- Apply the function you wrote in Problem 5 to your validation dataset using 3 different values of "alpha", such as (0.001, 0.01, 0.1). For each value, show the log-likelihood estimate of the validation set. That is, in your notebook show the sum of the log probabilities of all sentences.
- Which alpha gives you the best result? To indicate your selection to the grader, save your selected value to a variable named "**selected_alpha**".

PROBLEM 7 – Applying Language Models (20 pts)

In this problem, you will classify your test set of 100 sentences by sentiment, by applying your work from previous problems and modeling the language of both positive and negative sentiment.

To do this, you can follow these steps:

- Separate your training dataset into positive and negative sentences, and compute vocabulary size and bigram counts for both datasets.
- For each of the 100 sentences in your **test** set:
    - Compute both a "positive sentiment score" and a "negative sentiment score" using (1) the function you wrote in Problem 5, (2) Bayes rule, and (3) class priors as computed in Problem 1.
    - Compare these scores to assign a **predicted sentiment label** to the sentence.
- What is the class distribution of your **predicted label**? That is, how often did your method predict positive sentiment, correctly or incorrectly? How often did it predict negative sentiment? Show results in your notebook.
- Compare your **predicted label** to the **true sentiment label**. What is the accuracy of this experiment? That is, how often did the true and predicted label match on the test set? Show results in your notebook.

For this problem, you do not need to re-tune alpha for your positive and negative datasets (although it may be a good idea to do so), you can re-use the value selected in Problem 6.

PROBLEM 8 – Markov Assumption (10 pts – Answer in Blackboard)

- Where in this homework did you apply the Markov assumption?
- Imagine you applied the 2nd-order Markov assumption, using trigrams. Do you think your accuracy results would increase or decrease? Why? Or, if you are not sure, give a benefit or drawback of using trigrams for this experiment. (Note: You do not need to rerun this experiment with trigrams to answer this question.)

CODE ORGANIZATION AND READABILITY (5 pts)

The receive full credit, please ensure that:

- You have included **both** a notebook and PDF files, as described above in "WHAT TO SUBMIT"
- Your notebook includes cell output, and does NOT contain error messages
- It is easy to match a problem with its code solution in your notebook via markdown or comments (e.g., “Problem 2”)
- You re-ran your notebook before submitting, and cells are numbered sequentially starting at [1]