Submission instructions TBD
Late policy: No late homeworks accepted.

You may discuss problems at a high level with classmates, but you must solve and write them up yourselves.

This homework will give you practice coding up simple analyses of text data as well as implementing a language model and analyzing its results. Two data files are associated with this assignment: `orwell-train.txt` and `orwell-test.txt`. Both are drawn from Orwell’s novel 1984 (the first 5737 sentences and the final 1000 sentences, respectively). The file format is simple: One sentence per line, and all tokens separated by a space. The words have also been lowercased. Please do no further preprocessing, filtering, or tokenization of the data. Simply treat every space-separated string as a word token (including punctuation and digits). My implementation consists of 170 lines of python code (and a few lines of matlab to produce the plots).

**Question 1**

In this question we will analyze the distribution over word frequencies found in our text and compare it to the Zipf power-law. Recall from class that under the Zipf power-law, the frequency of a word is inversely proportional to its rank (where the rank of the most frequent word is 1, that of the second most frequent word is 2, that of the third most frequent word is 3, etc):

\[ freq(w) \propto \frac{1}{\text{rank}(w)} \]

To make this an exact equation (rather than a proportion), we simply normalize as follows:

\[ freq(w) = \frac{1}{\text{rank}(w) \sum_{n=1}^{\left|V\right|} 1/n} \]

Where \( |V| \) is the total number of word types (not tokens). This normalization ensures that the frequencies sum to one. Write a program that computes the empirical rank and frequency of all the words in `orwell-train.txt`. Note: there are 8459 word types in `orwell-train.txt`, resulting in a normalization value of \( \sum_{n=1}^{8459} 1/n \approx 9.62 \).

- Create a plot showing the relationship between these two values (rank on the x-axis, frequency on the y-axis). Create a similar plot for the actual Zipf power law (i.e. based on the above equation, not using counts from the data).

- Do the two look similar? If you’ve done this correctly, it should be very hard to see the graphs and compare them. It will appear that the frequency drops down to nearly zero very quickly in both cases.
• However, recall from lecture that what distinguishes a power law from an exponential function is that it has a “fat tail.” Let’s try zooming in on the tail to see this. Again, plot both the empirical frequencies and the Zipf power law, but this time begin at \( rank(w) = 500 \). Now do the graphs look similar?

• An even better way to do this is to plot the log-log graphs (i.e., plot \( \log(freq(w)) \) on the y-axis and \( \log(rank(w)) \) on the x-axis). Do so for both the empirical frequencies and the Zipf power law. The latter should be exactly linear. How close is the former to the latter?

**Question 2**

In this question we will be implementing several variants of n-gram language models. We will train the models using `orwell-train.txt` and then evaluate the cross-entropy and perplexity on both the training data as well as the test data (`orwell-test.txt`). In all cases, we will use add\( \delta \) smoothing. I.e., we will compute:

\[
P_{n-gram}(w_n|w_1, \ldots, w_{n-1}) = \frac{\text{count}(w_1, \ldots, w_n) + \delta}{\text{count}(w_1, \ldots, w_{n-1}) + |V|\delta}
\]

where \( |V| \) is the size of the vocabulary, \( \text{count}(\cdot) \) counts the number of occurrences of an n-gram in the training corpus, and \( \delta \) is the smoothing parameter. For our vocabulary, we will use all the word types that occur in either the training or test sentences, resulting in \( |V| = 9197 \). We will set our smoothing parameter to \( \delta = 0.00001 \).

• Train unigram, bigram, and trigrams models on the training data. Report cross-entropy and perplexity on both the training-set and the test-set. What trends do you notice as we move from unigrams to bigrams to trigrams? How do you explain these trends? (Note: to avoid underflow, do your computation in the log-domain. For example, to get perplexity, compute the negative log probability of the data, divide by the number of word tokens, then exponentiate. You will probably want to use log base 2 so that no conversion will be needed when computing the cross-entropy.)

• Interpolate the trained unigram, bigram, and trigram models, using the equation we gave in class: \( P_{int}(w_3|w_1, w_2) = \lambda_1 P_{uni}(w_1) + \lambda_2 P_{bi}(w_3|w_2) + \lambda_3 P_{tri}(w_3|w_1, w_2) \). Use \( \lambda_1 = \lambda_2 = \lambda_3 = 1/3 \). Report the cross-entropy and perplexity of the training and test-sets. How does this result compare to those of the un-interpolated models? Find a setting of \( \lambda_1, \lambda_2, \) and \( \lambda_3 \) (which sum to 1) giving an even better result.

• Now write a program to generate sentences from the trained language models (as before, defining our vocabulary to include all word types occurring in either the training or test sentences). Report a few generated sentences from all four language models. Can you tell which language model generate which sentences? (Bonus question: the trigram model will occasionally produce very long sentences of unrelated words. Why does this happen?).

**Question 3**

In this question we will explore models of sentence length. As we discussed in class, standard n-gram language models (with an end symbol) implicitly model sentence length as a geometric distribution (the
discrete counterpart to the exponential distribution). A geometric distribution models the probability of flipping a coin (with bias parameter \( p \)) \( \ell \) times successfully (heads) before obtaining the first failure (tails). I.e.

\[
P_{geom}(\ell; p) = p^{\ell}(1 - p)
\]

We will now use the unigram language model implemented in the previous question to model sentence length. In particular, we set \( 1 - p = P_{uni}(\langle /s \rangle) \) (the end symbol probability) and \( p = \sum_{w \in V \text{ s.t. } w \neq \langle /s \rangle} P_{uni}(w) \) (the probability of getting any word other than the end symbol).

- Compute and plot the probability of sentence lengths 1 through 100, according to this model.
- Now plot the frequency of the actual sentence lengths appearing in `orwell-train.txt`. In what ways are these two plots similar? In what way do they differ?
- To quantify the quality of the model, compute and report the cross-entropy and perplexity of the test-set sentence lengths.

Now we will consider two more sophisticated models of sentence length. For the first model, we will assume a multinomial distribution over sentence lengths and estimate it using maximum likelihood. I.e.,

\[
P_{mult}(\ell) = \frac{\text{count}(\ell)}{N}
\]

where \( \text{count}(\ell) \) is the number of sentences of length \( \ell \) in the training data, and \( N \) is the total number of sentences in the training data.

- Estimate this model on the training data, and plot the probabilities of sentence lengths 1 through 100. How does this plot compare to the plot of the actual training-set sentence lengths?
- To quantify the quality of the model, compute and report the cross-entropy and perplexity of the test-set sentence lengths. How do these results stack up to the previous model?

In the final model, we will assume that the sentence lengths follow a negative binomial distribution. This distribution models the probability of obtaining \( \ell \) successes (heads) before \( r \) failures (tails) have been observed, when flipping a coin with bias \( p \):

\[
P_{nbin}(\ell; p, r) = \frac{\Gamma(r + \ell)}{\Gamma(r)\Gamma(\ell + 1)} p^{\ell}(1 - p)^r
\]

The first factor is a normalization factor to ensure that the distribution sums to one, and consists of gamma functions (a generalization of factorial to non-integer arguments). Python has a built-in gamma function that you can use (from math import gamma).

There is no closed form for the maximum likelihood estimator of the negative binomial parameters (\( p \) and \( r \)). Instead, we use iterative gradient search to estimate these values. Using MATLAB’s `nbinfit` command I obtained maximum likelihood estimates of \( p = .8682 \), and \( r = 2.6878 \) (on the training-set sentence lengths).
• Using $p = .8682$, and $r = 2.6878$, plot the probabilities of sentence lengths 1 through 100. How does this plot compare to the plot of the actual training-set sentence lengths?

• To quantify the quality of the model, compute and report the cross-entropy and perplexity of the test-set sentence lengths. How do these results stack up to the previous two models?