Assignment 1 – Next Word Prediction

CSCI-LING 5832 – Spring 2025

Due 2/9/2025

Disclaimer: All code turned in is expected to be python. You are given the choice of making a script file (*.py) or a jupyter notebook (*.ipynb). Whichever route you choose, please ensure your final submission is clean and well-organized – enough so as to be quickly evaluated by your TAs. You will also need to upload a pdf file including a written report, details below.

N-Grams

In this assignment, you are tasked with constructing two different n-gram models. First, let's understand the mathematical components of these models.

Probability

We know that the probability of some event is a measure of its likelihood under some constraints. We won't be particularly interested in unigrams in this assignment. However, it's still important to know how to calculate the probability of a particular word occurring in our corpus in isolation. This is a simple measure of its frequency in the dataset (N is the total number of tokens in our dataset).

$$P(w_1) = \frac{count(w_1)}{N} \tag{1}$$

In the case of other n-gram models, we are concerned with conditional probabilities. Conditional probability represents the likelihood of some event A given another event B has already occurred. For instance, in the utterance *The cat is on the mat*, we calculate the probability of *cat* appearing after the word *the* as the number of times *cat* appears after *the* divided by the total number of times *the* appears. In this case, the probability is $\frac{1}{2}$, or 0.5. Conditional probability is calculated as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

When applied to co-occurrence of words, we can modify this formula to reflect the counting method indicated above:

$$P(w_2|w_1) = \frac{count(w_1 w_2)}{count(w_1)} \tag{2}$$

To predict the probability of an entire sentence we would want to calculate the probability $P(w_1 w_2 w_3 ... w_n)$. This requires a slight modification to our previous equation as this probability would be akin to $P(X_1 \cap X_2 \cap X_3 \cap ... X_n)$. This is called a joint probability, or the likelihood of some number of event co-occurring. In the simple case of two events, we can derive from our conditional probability formula:

$$P(A \cap B) = P(A|B)P(B)$$

However, in the case of a longer sequence the appearance of each word is conditioned on the appearance of **all prior words in the sentence**. Thus, to find the probability of the entire sequence, we can use the chain rule of probability:

$$P(X_1...X_n) = P(X_1)P(X_2|X_1)P(X_3|X_{1:2})...P(X_n|X_{1:n-1})$$
$$= \prod_{k=1}^{n} P(X_k|X_{1:k-1})$$

Putting this equation in the context of our sentences:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w_n|w_{1:n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1})$$

Implementing N-Gram Models

You will be implementing bigram and trigram models in this assignment. These models are used to approximate the probabilities of an entire sequence by simplifying the joint probability of any length sequence to the probability of a smaller number of words. The appearance of any word only being conditioned on the previous word is called the **Markov** assumption. In the case of a bigram, we follow this assumption strictly – only the current and previous word are used in our calculations (bi= 2). In the case of trigrams, we use the current word and the two previous words, and this pattern continues for any n-gram.

Calculations

It so happens that we've seen an example of the bigram likelihood calculation above in our *The cat is on the mat* example. We calculate the probability of each bigram in a corpus as in Eq. (2). For trigrams we can use the formula:

$$P(w_3|w_{1:2}) = \frac{count(w_1w_2w_3)}{count(w_1w_2)}$$

Now that we have these formulas, we can download a corpus and start creating some n-gram models. We recommend following these steps:

1. Write a function that calculates the unigram probability of a word appearing in some corpus.

```
Example:
In: prob("this", corpus)
Out: 0.25
```

2. Create the bigram model.

   i. Write a function that finds all bigrams in a sentence and returns them as a list of tuples.

```
Example:
In: "The quick brown fox jumps over the lazy dog"
Out: [("The", "quick"), ("quick", "brown"), ("brown", "fox"), ("fox", "jumps"),
("jumps", "over"), ("over", "the"), ("the", "lazy"), ("lazy", "dog")]
```

   ii. Write a function that calculates the conditional probability of a particular word given some previous word and a list of bigram tuples. This should be done using Laplace Smoothing, see `https://web.stanford.edu/~jurafsky/slp3/3.pdf`, pg 15, Eq. (3.26).

```
Example:
In: conditional_prob_bg("one", "this", bigrams)
Out: 0.0
```

   iii. Write a function that predicts the next word in a sequence (i.e., find the bigram which yields the highest conditional probability given a set initial word).

```
Example:
In: predict_next_word("this", bigrams)
Out: "is"
```

iv. Write a function that predicts an entire sentence by continuously finding the most probable next word given a list of bigrams and some initial sequence. Only do this up to some specified limit in length (assume the initial sequence len ≥ 1, and use a limit of 5-10 for testing).

```
Example:
In: predict_sentence("<s> This is", bigrams, limit=6)
Out: "<s> This is sentence one </s>"
```

3. Create the trigram model by following the same general steps as above, but applied to trigrams.
4. Download the Brown and Webtext corpora from NLTK and generate a set of bigrams and trigrams from each corpus to be used for sentence prediction. You should have 4 models to be used with your sentence predictor utility function: `bigrams_brown, trigrams_brown, bigrams_webtext, trigrams_webtext`. Also download the Reuters corpus, but do not create bigrams or trigrams for it (this will be used for testing later). You should use the following code to download and pre-process the corpora:

```
import nltk
nltk.download("brown")
nltk.download("webtext")
nltk.download("reuters")
nltk.download("punkt_tab")
from nltk.corpus import brown, webtext, reuters
brown_corpus = brown.sents()
brown_corpus = [" ".join(sentence) for sentence in brown_corpus]
brown_corpus = ["<s> " + sentence + " </s>" for sentence in brown_corpus][:5000]
webtext_corpus = webtext.sents()
webtext_corpus = [" ".join(sentence) for sentence in webtext_corpus]
webtext_corpus = ["<s> " + sentence + " </s>" for sentence in webtext_corpus][:5000]
reuters_corpus = reuters.sents()
reuters_corpus = [" ".join(sentence) for sentence in reuters_corpus]
reuters_corpus = ["<s> " + sentence + " </s>" for sentence in reuters_corpus][:5000]
```

At any point in the implementation where you need to count occurrences of some word or sequence, we highly recommend using the `Counter` class found in the base python `collections` package.

Perplexity

Perplexity is a measure of... how *perplexed* our model is by a certain sequence. Essentially, this can be thought of as a metric explaining how surprised we should be if our model outputs a given sequence. Formally, perplexity is the inverse probability of the sentence normalized by its length. It is also the geometric mean of the inverse conditional probability of each word in the sentence given the previous word.

$$PPL = \sqrt[n]{\frac{1}{P(w_1 w_2 w_3 ... w_n)}} = \sqrt[n]{\frac{1}{\prod_{i=1}^{n} P(w_i | w_{1:i-1})}}$$

In our case, using bigrams and trigrams we get the two respective formulas:

$$PPL_{Bi} = \sqrt[n]{\frac{1}{\prod_{i=1}^{n} P(w_i | w_{i-1})}} \qquad PPL_{Tri} = \sqrt[n]{\frac{1}{\prod_{i=1}^{n} P(w_i | w_{i-2} w_{i-1})}}$$

You will be required to implement an additional function to calculate the perplexity of a sentence given a set of bigrams and/or trigrams as well as a set of unigram counts. You could also use only the test sentence and the training corpus and call your unigram, bigram, and trigram functions from within your perplexity function. Essentially, your goal is to implement functions based on the above formulas.

```
Example:
In: perplexity_bg(sample_sentence, bigrams_brown, unigrams_brown)
Out: 2602.55407389149
```

NOTE: We have a problem when finding the conditional probability of w_1 (and w_2 in the trigram case) here as there are no preceding tokens. To make life simple, we can use the standard probability of that word occurring over the *entire corpus*. We can use Eq. 1 for to find the probability of w_1 and in the w_2 case with trigrams we can use Eq. 2.

Written Report

The last item you will be required to turn in is a written report (minimum 1 page) answering the following questions about this assignment and the concepts it is meant to cover.

1. What do you expect the difference between the Brown bigram and trigram models to look like? Which model will provide you with more coherent text? How will the perplexity of each compare? You should test your predictor and perplexity function using the `brown_bigrams` and `brown_trigrams` to confirm your expectations. For perplexity, an average over 2-5 sentences from the Brown corpus should be fine, but make sure you use the same sentences both times. If something you did not expect occurs, explain what happened and why you believe it happened.

2. When testing our bigram models on the Reuters data, do you think a model trained on Brown or Webtext will perform best? Pick any 25 sentences from the Reuters corpus and calculate the average perplexity using each of your bigram datasets. Compare the results of each and provide explanation as to why you believe that one performed better than the other.

3. When predicting the next word in a sentence, what do you believe would happen if we increased the number of sentences in our training data?

Rubric

Base Points	**Item**
30	Bigram Model
30	Trigram Model
20	Perplexity
20	Written Report
Point Multipliers	**Criteria**
1	All expected functionality is present. Report answers questions in a near-perfect manner.
0.75	Minor errors not impacting general functionality. Output may slightly deviate from what is expected. Report answers questions well with minor errors.
0.5	Errors demonstrating a lack of understanding. Some functionality missing. Report demonstrates lack of understanding.
0.25	Significant errors affecting the functionality of the item. Report has significant errors and does not answer questions in an acceptable manner.
0	No work present or code does not run. No report.