Homework 1

Daniel Gargiullo (ID: 24277715)

CSCI366 – Natural Language Processing | Prof. Alla Rozovskaya | Spring 2024

Part I

Q3.4: We are given the following corpus:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I am Sam </s>

<s> I do not like green eggs and Sam </s>

Using a bigram language model with add-one smoothing, what is P(Sam| am)?

Include <s> and </s> in your counts just like any other token.

$$P(\text{Sam} \mid \text{am})$$

$$= \text{count}(\text{am Sam}) + 1 \;/\; \text{count}(\text{am}) + \text{number of unique words}$$

$$= (2) + 1 \;/\; (3) + (10)$$

$$= 3 \;/\; 13$$

Part II

Q1: There are 41,738 unique words (word types) in the training corpus (excluding <s>). This number is determined at the time of training the UnigramModel. When the UnigramModel is trained, the number of unique words are stored in the dict V (See code snippet below).

Q2: There are a total of 2,468,210 words (tokens) in the training corpus. This number is determined at the time of training the UnigramModel. When the UnigramModel is trained, the total number of words are stored in list W (See code snippet below):

Snippet for Q1, Q2:

```
class UnigramModel:
    def __init__(self):
        self.V = {}  # unigrams
        self.W = []  # full set of tokens

    def train(self, training_corpus: iter):
        with open(file=training_corpus, mode='r', encoding='utf-8') as T:
            for s in T:
                W_ = []  # temporary set for storing the current line's tokens
                for w in s.strip().split():
                    if w != '<s>':
                        W_.append(w)
                self.W.extend(W_)
                for w in W_:
                    self.V[w] = self.V.get(w, 0) + 1


        # ...
```

Q3: Percentage of unseen word tokens in test data: 3.6114%.

Percentage of unseen word types in test data: 1.6612%.

These two figures are determined by `main`'s `missing_unigrams_ratio()` and `missing_uni_token_ratio()` functions, which calculate the percentage of unique word *types* and total word *tokens* in the *test* corpus that did not appear in the *training* corpus:

```
return '%.4f' % (missing_unigrams / total_unigrams * 100)
```

```
return '%.4f' % (missing_tokens / total_tokens * 100)
```

Q4: Percentage of unseen bigram tokens in test data: 30.1192%.
Percentage of unseen bigram types in test data: 27.5009%

These two figures are determined by `main`'s `missing_bigrams_ratio()` and `missing_bi_token_ratio()` functions, which calculate the percentage of unique bigram *types* and total bigram *tokens* in the *test* corpus that did not appear in the *training* corpus:

```
return '%.4f' % (missing_bigrams / total_bigrams * 100)
return '%.4f' % (missing_bigrams / total_bigrams * 100)
```

Q5:

I look forward to hearing your reply .

To compute the logarithmic probabilities, the above sentence is passed into the `compute_probability_log()` function in each of the instances of the three model types created in `main.py`. The raw probability is then calculated using the `compute_probability_raw()` function, the base-2 logarithm of which is the resulting logarithmic probability. ***See the probability values by running the program and viewing the console output, or see the output file.***

Q6: To compute the perplexity of the sentence, I created a `compute_perplexity_of_sentence()` function in the UnigramModel and BigramModel classes that takes the raw probabilities of the individual words (in the case of the UnigramModel) and the raw probabilities of the combinations of bigrams (in the case of the BigramModel), as well as an additional `compute_perplexity_smoothed_of_sentence()` function in the BigramModel for calculating the smoothed perplexity of a sentence, the base-2 logarithms of which are then use in their perplexity formula:

Unigram:

```
...
        p = self.compute_probability_raw(w)
        log_2_p += math.log2(p)
PPL = 2**(-log_2_p / len(s))

return PPL
```

Result: `756.9920623085402`

Bigram:

```
...
p = self.compute_probability_raw(w_0, w_1)
try:
        log_2_p += math.log2(p)
        except ValueError:
        return "UNDEFINED! ..."
PPL = 2**(-log_2_p / len(s) - 1)

return PPL
```

Result: `UNDEFINED! There exists a bigram tuple with a 0 value (cannot divide by zero)!`

Bigram (smooth):

```
...
        p = self.compute_probability_smooth(w_0, w_1)
        try:
                log_2_p += math.log2(p)
        except ValueError:
                return "UNDEFINED! ..."
        PPL = 2**(-log_2_p / len(s) - 1)

        return PPL
```

Result: `658.8564401431914`

Q7: To compute the perplexity of the file, I created a `compute_perplexity_of_corpus()` function in each of the models' respective classes, as well as an additional `compute_perplexity_smoothed_of_corpus()` function that each take either the raw probabilities of the

individual words and bigrams (in the cases of the UnigramModel and BigramModel, respectively) and the smoothed probabilities of the combinations of bigrams (in the case of the BigramModel with smoothing), the base-2 logarithms of which are then used in the respective perplexity formulas, which compute the perplexity for an entire file as opposed to just a sentence. See below.
Output perplexity values:

```
Unigram: 802.0839563494246
```

```
Bigram: UNDEFINED! There exists a bigram tuple with a 0 value (cannot divide by zero)!
```

```
Bigram: 1357.0366087733726
```

A lower perplexity corresponds to a better fit for the language model. The UnigramModel has the lower perplexity of the three, which makes sense since the text being passed to it is almost 3,000 words long. Once again (as in Q6 above), the standard Bigram encounters a tuple with a zero value, and an exception is thrown, with no PPL (perplexity) being calculated, as it is undefined. The Bigram using Laplace (Add-One) Smoothing resolves this issue by giving a value of 1 to all tuples with a 0 value, however, it does not prove to be a better fit for the model in the case of the entire corpus. It *is*, however, a better fit for the model in the case of the sentence “I look forward to hearing your reply .”, as we’ve seen in Q6 above.