```
import nltk # Import the Natural Language Toolkit library
import re # Import the regular expression module
import math # Import the math module for mathematical operations
from collections import Counter # Import Counter for efficient counting of hashable objects
```

```
with open("ngram_corpus.txt", "r", encoding="utf-8") as file:
    text = file.read() # Read the entire content of the corpus file into the 'text' variable

print(text[:400]) # Print the first 400 characters of the text to inspect it
```

```
Natural language processing is a field of artificial intelligence that focuses on the interaction between computers and human
Language models play an important role in understanding how words are structured and used in sentences.
In everyday communication, humans naturally predict the next word based on context.
Similarly, machines use probabilistic models to estimate the likelihood of w
```

```
def preprocess_text(text):
    text = text.lower() # Convert all text to lowercase for consistency
    # First, tokenize into sentences. Punctuation is needed for this.
    raw_sentences = nltk.sent_tokenize(text) # Split the text into sentences using NLTK's sentence tokenizer

    processed = [] # Initialize an empty list to store processed sentences
    for sent in raw_sentences:
        # Now remove punctuation from each sentence and tokenize words
        clean_sent = re.sub(r'[^a-z\s]', '', sent) # Remove non-alphabetic characters (keep spaces) from each sentence
        words = nltk.word_tokenize(clean_sent) # Tokenize the cleaned sentence into words
        if words: # Ensure the sentence is not empty after cleaning
            # Add start-of-sentence (<s>) and end-of-sentence (</s>) tokens
            processed.append(['<s>'] + words + ['</s>'])
    return processed # Return the list of processed sentences
```

```
nltk.download('punkt_tab') # Download the 'punkt_tab' tokenizer data, required by nltk.sent_tokenize
sentences = preprocess_text(text) # Preprocess the raw text to get a list of tokenized sentences
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
def build_ngram(sentences, n):
    ngrams = [] # Initialize an empty list to store n-grams
    for sent in sentences:
        for i in range(len(sent) - n + 1):
            ngrams.append(tuple(sent[i:i+n])) # Extract n-grams and add them as tuples
    return Counter(ngrams) # Return a Counter object with the frequency of each n-gram
```

```
split_index = int(0.8 * len(sentences)) # Calculate the index to split data (80% for training)
train_data = sentences[:split_index] # Create the training dataset
test_data = sentences[split_index:] # Create the testing dataset

unigram = build_ngram(train_data, 1) # Build unigram counts from the training data
bigram = build_ngram(train_data, 2) # Build bigram counts from the training data
trigram = build_ngram(train_data, 3) # Build trigram counts from the training data

vocab_size = len(unigram) # Determine the vocabulary size from the unigrams
```

```
def laplace_probability(ngram, ngram_counts, prefix_counts, vocab_size):
    # Calculate Laplace (add-1) smoothed probability for an n-gram
    # (count(ngram) + 1) / (count(prefix) + vocab_size)
    # ngram[:-1] gets the prefix (e.g., for a bigram (w1, w2), prefix is (w1,))
    return (ngram_counts[ngram] + 1) / (prefix_counts[ngram[:-1]] + vocab_size)
```

```
```

```
def sentence_probability(sentence, n, ngram_counts, prefix_counts):
    probability = 1 # Initialize probability to 1
    # Iterate through the sentence to extract all n-grams
    for i in range(len(sentence) - n + 1):
        ngram = tuple(sentence[i:i+n]) # Get the current n-gram
        # Multiply by the Laplace smoothed probability of the current n-gram
        probability *= laplace_probability(
            ngram, ngram_counts, prefix_counts, vocab_size
        )
    return probability # Return the overall probability of the sentence
```

```
sample_sentences = test_data[:5] # Take the first 5 sentences from the test data for sampling

for sent in sample_sentences:
    print("Sentence:", " ".join(sent)) # Print the current sentence
    # Calculate and print unigram probability (prefix_counts for unigram is an empty Counter)
    print("Unigram Probability:", sentence_probability(sent, 1, unigram, Counter()))
    # Calculate and print bigram probability (unigram counts as prefix_counts)
    print("Bigram Probability:", sentence_probability(sent, 2, bigram, unigram))
    # Calculate and print trigram probability (bigram counts as prefix_counts)
    print("Trigram Probability:", sentence_probability(sent, 3, trigram, bigram))
    print() # Print an empty line for readability
```

```
Sentence: <s> natural language processing is a field of artificial intelligence that focuses on the interaction between comput
Unigram Probability: 3.6282281532084516e-17
Bigram Probability: 5.728666499035252e-29
Trigram Probability: 7.3630858905907785e-28

Sentence: <s> language models play an important role in understanding how words are structured and used in sentences </s>
Unigram Probability: 2.8559273450575154e-15
Bigram Probability: 7.437365592981607e-25
Trigram Probability: 1.4594677858771676e-23

Sentence: <s> in everyday communication humans naturally predict the next word based on context </s>
Unigram Probability: 1.0376662168584377e-12
Bigram Probability: 1.6539775813155558e-18
Trigram Probability: 4.917600997563318e-17

Sentence: <s> similarly machines use probabilistic models to estimate the likelihood of word sequences </s>
Unigram Probability: 5.207164734489701e-12
Bigram Probability: 5.1636267711381046e-20
Trigram Probability: 8.198367109656208e-18

Sentence: <s> in a small village there lived a teacher who loved reading books and explaining stories to children </s>
Unigram Probability: 2.784619895532839e-16
Bigram Probability: 1.0041002530737421e-26
Trigram Probability: 5.707829327541519e-25
```

```
def perplexity(sentence, n, ngram_counts, prefix_counts):
    log_prob = 0 # Initialize log probability to 0
    N = len(sentence) # Get the length of the sentence (number of words including <s> and </s>)

    # Iterate through the sentence to calculate log probabilities of n-grams
    for i in range(len(sentence) - n + 1):
        ngram = tuple(sentence[i:i+n]) # Get the current n-gram
        prob = laplace_probability( # Calculate Laplace smoothed probability
            ngram, ngram_counts, prefix_counts, vocab_size
        )
        log_prob += math.log(prob) # Add the natural logarithm of the probability

    # Perplexity is exp(-average_log_probability)
    return math.exp(-log_prob / N)
```

```
for sent in sample_sentences:
    print("Sentence:", " ".join(sent)) # Print the current sentence
    # Calculate and print unigram perplexity (prefix_counts for unigram is an empty Counter)
    print("Unigram Perplexity:", perplexity(sent, 1, unigram, Counter()))
    # Calculate and print bigram perplexity (unigram counts as prefix_counts)
    print("Bigram Perplexity:", perplexity(sent, 2, bigram, unigram))
    # Calculate and print trigram perplexity (bigram counts as prefix_counts)
    print("Trigram Perplexity:", perplexity(sent, 3, trigram, bigram))
    print() # Print an empty line for readability
```

```
Sentence: <s> natural language processing is a field of artificial intelligence that focuses on the interaction between compu
Unigram Perplexity: 6.065570937894682
Bigram Perplexity: 22.12353864680698
Trigram Perplexity: 19.59046606961734

Sentence: <s> language models play an important role in understanding how words are structured and used in sentences </s>
Unigram Perplexity: 6.427085049467162
Bigram Perplexity: 21.90164397652709
Trigram Perplexity: 18.563328938999707

Sentence: <s> in everyday communication humans naturally predict the next word based on context </s>
Unigram Perplexity: 7.177874821110077
Bigram Perplexity: 18.625375073876555
Trigram Perplexity: 14.617555960579692

Sentence: <s> similarly machines use probabilistic models to estimate the likelihood of word sequences </s>
Unigram Perplexity: 6.396715695571121
Bigram Perplexity: 23.858661885689326
Trigram Perplexity: 16.612999255690358

Sentence: <s> in a small village there lived a teacher who loved reading books and explaining stories to children </s>
Unigram Perplexity: 6.587134488179844
Bigram Perplexity: 23.35218498595695
Trigram Perplexity: 18.87883590763391
```

```
def average_perplexity(test_data, n, ngram_counts, prefix_counts):
    total = 0 # Initialize total perplexity to 0
    count = 0 # Initialize a counter for valid sentences

    for sentence in test_data:
        # Only calculate perplexity if the sentence is long enough for the n-gram
        if len(sentence) >= n:
            total += perplexity(sentence, n, ngram_counts, prefix_counts) # Add perplexity of current sentence to total
            count += 1 # Increment sentence counter

    return total / count # Return the average perplexity
```

```
uni_pp = average_perplexity(test_data, 1, unigram, Counter()) # Calculate average unigram perplexity
bi_pp = average_perplexity(test_data, 2, bigram, unigram) # Calculate average bigram perplexity
tri_pp = average_perplexity(test_data, 3, trigram, bigram) # Calculate average trigram perplexity

print("Average Perplexity Comparison") # Print header for comparison
print("Unigram Model :", uni_pp) # Print unigram average perplexity
print("Bigram Model  :", bi_pp) # Print bigram average perplexity
print("Trigram Model :", tri_pp) # Print trigram average perplexity
```

```
Average Perplexity Comparison
Unigram Model : 6.894216645017806
Bigram Model  : 22.11844097117447
Trigram Model : 14.832565699291335
```

Load and Inspect Text

This cell handles loading the `ngram_corpus.txt` file and reading its content into the `text` variable. It then displays the first 400 characters of the loaded text to allow for a quick inspection of the data.

Imports

This cell imports the necessary Python libraries for this project:

- `nltk`: The Natural Language Toolkit, a leading platform for building Python programs to work with human language data.
- `re`: The regular expression module, used for pattern matching and text manipulation.
- `math`: Provides access to mathematical functions, particularly used for logarithmic calculations in perplexity.
- `collections.Counter`: A specialized dictionary subclass for counting hashable objects, essential for building n-gram frequency tables.

```
Start coding or generate with AI.
```