```
import re # Regular expressions for text cleaning
import math # Math functions for log and exp
import random # Random for shuffling data
from collections import defaultdict, Counter # Containers for counting n-gra

import nltk # Natural Language Toolkit
import numpy as np # Numerical computing
import pandas as pd # Data manipulation
```

```
# Open and read the corpus file
with open("corpus.txt", "r", encoding="utf-8") as f:
    text = f.read()

# Preview the first 500 characters
print(text[:500])
```

```
Once upon a time, in a small village surrounded by green hills, there
lived a curious boy named Arun. Arun loved to explore the forests,
rivers, and fields around his home. Every morning, he would wake up
early, watch the sunrise, and plan a new adventure for the day.

The village was peaceful and full of friendly people. Farmers worked in
the fields, children played near the river, and elders sat under the
large banyan tree discussing stories from the past. Life was simple, but
Arun always drea
```

```
def preprocess(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove all characters except lowercase letters, spaces, and periods
    text = re.sub(r'[^a-z\s\.]', '', text)
    # Split the text into sentences based on periods
    sentences = text.split('.')

    processed_sentences = []
```

```
    for sent in sentences:
        # Tokenize each sentence into words
        tokens = sent.strip().split()
        if len(tokens) > 0:
            # Add start (<s>) and end (</s>) tokens to each sentence
            tokens = ['<s>'] + tokens + ['</s>']
            processed_sentences.append(tokens)
    return processed_sentences

# Apply preprocessing to the entire text
sentences = preprocess(text)
```

```
# Randomly shuffle the sentences to ensure a diverse split
random.shuffle(sentences)
# Calculate the split index for 80% training data
split = int(0.8 * len(sentences))

# Split the data into training and testing sets
train_data = sentences[:split]
test_data = sentences[split:]
```

```
#Unigram
# Initialize a Counter for unigrams (single words)
unigram_counts = Counter()

# Count occurrences of each word in the training data
for sent in train_data:
    unigram_counts.update(sent)

# Calculate total number of unigrams and unique vocabulary size
total_unigrams = sum(unigram_counts.values())
vocab_size = len(unigram_counts)
```

```
#Bigram
# Initialize a nested dictionary for bigram counts
bigram_counts = defaultdict(Counter)
```

```
# Count occurrences of word pairs in the training data
for sent in train_data:
    for i in range(len(sent)-1):
        bigram_counts[sent[i]][sent[i+1]] += 1
```

```
#Trigram
# Initialize a nested dictionary for trigram counts using tuples of (w1, w2)
trigram_counts = defaultdict(lambda: defaultdict(int))

# Count occurrences of word triplets in the training data
for sent in train_data:
    for i in range(len(sent)-2):
        trigram_counts[(sent[i], sent[i+1])][sent[i+2]] += 1
```

```
# Calculate unigram probability with Laplace (add-one) smoothing
def unigram_prob(word):
    return (unigram_counts[word] + 1) / (total_unigrams + vocab_size)

# Calculate bigram probability with Laplace smoothing
def bigram_prob(w1, w2):
    return (bigram_counts[w1][w2] + 1) / (sum(bigram_counts[w1].values()) + vocab_size)

# Calculate trigram probability with Laplace smoothing
def trigram_prob(w1, w2, w3):
    return (trigram_counts[(w1,w2)][w3] + 1) / (sum(trigram_counts[(w1,w2)].values()) + vocab_size)
```

```
# Calculate the total probability of a sentence using the unigram model
def sentence_prob_unigram(sentence):
    prob = 1
    for w in sentence:
        prob *= unigram_prob(w)
    return prob

# Calculate the total probability of a sentence using the bigram model
```

```
def sentence_prob_bigram(sentence):
    prob = 1
    for i in range(len(sentence)-1):
        prob *= bigram_prob(sentence[i], sentence[i+1])
    return prob

# Calculate the total probability of a sentence using the trigram model
def sentence_prob_trigram(sentence):
    prob = 1
    for i in range(len(sentence)-2):
        prob *= trigram_prob(sentence[i], sentence[i+1], sentence[i+2])
    return prob
```

```
# Evaluate probabilities for the first 5 sentences in the test set
for s in test_data[:5]:
    print("Sentence:", s)
    print("Unigram:", sentence_prob_unigram(s))
    print("Bigram :", sentence_prob_bigram(s))
    print("Trigram:", sentence_prob_trigram(s))
    print()
```

```
Sentence: ['<s>', 'years', 'passed', 'and', 'arun', 'grew', 'up', 'to', 'be', 'a', 'wise', 'and', 'respected', 'man', '<
Unigram: 2.2640603927765624e-34
Bigram : 3.623904383329128e-34
Trigram: 7.819606159930844e-32

Sentence: ['<s>', 'his', 'grandfather', 'smiled', 'and', 'said', 'that', 'long', 'ago', 'people', 'believed', 'there', '
Unigram: 3.519246786228886e-46
Bigram : 1.335464978023653e-44
Trigram: 8.403030329341439e-44

Sentence: ['<s>', 'inside', 'the', 'house', 'everything', 'was', 'dusty', 'and', 'old', '</s>']
Unigram: 2.5925645439732143e-21
Bigram : 7.057801045262797e-22
Trigram: 1.432006253286335e-19

Sentence: ['<s>', 'arun', 'believed', 'that', 'every', 'path', 'leads', 'to', 'learning', 'every', 'challenge', 'builds'
Unigram: 1.6960446865545432e-47
```

```
Bigram : 4.590346773077812e-43
Trigram: 2.051233172162813e-41

Sentence: ['<s>', 'as', 'he', 'walked', 'further', 'the', 'forest', 'became', 'quieter', 'and', 'the', 'air', 'felt', 'c
Unigram: 5.048763013527654e-33
Bigram : 6.105316837785146e-33
Trigram: 7.602651031254519e-32
```

```
# Calculate the perplexity of a language model on a given dataset
def perplexity(model_func, data, n):
    N = 0 # Total number of tokens
    log_prob = 0 # Cumulative log probability

    for sent in data:
        N += len(sent)
        if n == 1:
            # Sum log probabilities for unigrams
            for w in sent:
                log_prob += math.log(model_func(w))
        elif n == 2:
            # Sum log probabilities for bigrams
            for i in range(len(sent)-1):
                log_prob += math.log(model_func(sent[i], sent[i+1]))
        elif n == 3:
            # Sum log probabilities for trigrams
            for i in range(len(sent)-2):
                log_prob += math.log(model_func(sent[i], sent[i+1], sent[i+2]))

    # Perplexity is the exponent of the negative average log probability
    return math.exp(-log_prob / N)
```

```
# Print the perplexity scores for all three models
print("Unigram Perplexity:", perplexity(unigram_prob, test_data, 1))
print("Bigram Perplexity :", perplexity(bigram_prob, test_data, 2))
print("Trigram Perplexity:", perplexity(trigram_prob, test_data, 3))
```

```
Unigram Perplexity: 168.5670547920451
Bigram Perplexity : 135.9076480941145
Trigram Perplexity: 106.0934698012698
```

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