```
import nltk
import numpy as np
import pandas as pd
from collections import Counter, defaultdict
from nltk.tokenize import word_tokenize
from nltk.util import ngrams

# Download necessary NLTK data for tokenization
nltk.download('punkt')

"""
LIBRARY EXPLANATIONS:
1. NLTK: The primary toolkit for Natural Language Processing; used here for
   tokenizing text and generating n-gram sequences.
2. NUMPY: Used for mathematical operations, specifically for calculating
   logarithms during Perplexity computation.
3. PANDAS: Useful for creating frequency tables and structured views of our data.
4. COLLECTIONS (Counter/defaultdict): Efficiently counts occurrences of words
   and sequences to build our probability distributions.
"""
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
'\nLIBRARY EXPLANATIONS:\n1. NLTK: The primary toolkit for Natural Language Processing; used here
for \n   tokenizing text and generating n-gram sequences.\n2. NUMPY: Used for mathematical operat
ions, specifically for calculating \n   logarithms during Perplexity computation.\n3. PANDAS: Use
ful for creating frequency tables and structured views of our data.\n4. COLLECTIONS (Counter/defa
ultdict): Efficiently counts occurrences of words \n   and sequences to build our probability dis
tributions.\n'
```

```
from nltk.corpus import gutenberg
import re

# 1. Load the dataset (using 'Melville's Moby Dick' as an example)
nltk.download('gutenberg')
raw_text = gutenberg.raw('melville-moby_dick.txt')

# 2. Cleaning: Remove unnecessary lines, numbers, and extra whitespace
# We limit to the first 12,000 characters to ensure we hover around 2000 words
sample_raw = raw_text[:12000]

# Remove chapter headings (e.g., CHAPTER 1) and non-alphabetical noise
cleaned_text = re.sub(r'CHAPTER \d+', '', sample_raw)
cleaned_text = re.sub(r'\s+', ' ', cleaned_text).strip()

# 3. Display sample and word count
words = cleaned_text.split()
print(f"Total Words in Dataset: {len(words)}")
print("-" * 30)
print("Sample Text (First 300 characters):")
print(cleaned_text[:300] + "...")
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
Total Words in Dataset: 1969
------------------------------
Sample Text (First 300 characters):
[Moby Dick by Herman Melville 1851] ETYMOLOGY. (Supplied by a Late Consumptive Usher to a Grammar
[nltk_data]   Unzipping corpora/gutenberg.zip.
```

```
from nltk.tokenize import sent_tokenize
import nltk
nltk.download('punkt_tab')

# Split into sentences first
sentences = sent_tokenize(cleaned_text.lower())

# Calculate split index (80% Train, 20% Test)
split_idx = int(len(sentences) * 0.8)
train_sents = sentences[:split_idx]
test_sents = sentences[split_idx:]

print(f"Training Sentences: {len(train_sents)}")
print(f"Testing Sentences: {len(test_sents)}")
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Training Sentences: 122
Testing Sentences: 31
```

```
import string
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')

def preprocess_text(sentences, remove_stops=False):
    cleaned_data = []
    stop_words = set(stopwords.words('english'))

    for sent in sentences:
        # 1. Lowercase
        sent = sent.lower()

        # 2. Remove punctuation and numbers
        # This creates a translation table that maps punctuation/digits to None
        sent = sent.translate(str.maketrans('', '', string.punctuation + string.digits))

        # 3. Tokenization
        tokens = word_tokenize(sent)

        # 4. Optional: Remove Stopwords
        if remove_stops:
            tokens = [w for w in tokens if w not in stop_words]

        # 5. Add Start and End Tokens
        # We add <s> to denote the start and </s> for the end of a sequence
        processed_sent = ['<s>'] + tokens + ['</s>']

        cleaned_data.append(processed_sent)

    return cleaned_data

# Execute preprocessing
processed_train = preprocess_text(train_sents)
processed_test = preprocess_text(test_sents)

# Display a sample result
print("Original Sentence Sample:", train_sents[0])
print("Processed Tokens:", processed_train[0])
```

```
Original Sentence Sample: [moby dick by herman melville 1851] etymology.
Processed Tokens: ['<s>', 'moby', 'dick', 'by', 'herman', 'melville', 'etymology', '</s>']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
all_train_tokens = [token for sent in processed_train for token in sent]

# 1. Unigram Counts
unigram_counts = Counter(all_train_tokens)

# 2. Bigram Counts
bigram_list = [bg for sent in processed_train for bg in ngrams(sent, 2)]
bigram_counts = Counter(bigram_list)

# 3. Trigram Counts
trigram_list = [tg for sent in processed_train for tg in ngrams(sent, 3)]
trigram_counts = Counter(trigram_list)

# Convert to DataFrame for a "Table" view (Top 10 Unigrams)
unigram_df = pd.DataFrame(unigram_counts.items(), columns=['Word', 'Count']).sort_values(by='Cour
print("Top 10 Unigrams:\n", unigram_df.head(10))
```

```
Top 10 Unigrams:
     Word  Count
7   </s>    122
0    <s>    122
16   the     81
45    of     55
23   and     48
9      a     37
19    in     37
13    to     31
84  that     26
33   his     25
```

```
def build_prob_table(subset_words, bigram_counts, unigram_counts):
    vocab = list(unigram_counts.keys())
    V = len(vocab)

    # Initialize a matrix with zeros
    matrix = np.zeros((len(subset_words), len(subset_words)))

    for i, w_prev in enumerate(subset_words):
        for j, w_curr in enumerate(subset_words):
            count_bg = bigram_counts.get((w_prev, w_curr), 0)
            count_uni = unigram_counts.get(w_prev, 0)

            # Apply Laplace Smoothing
            prob = (count_bg + 1) / (count_uni + V)
            matrix[i, j] = prob

    return pd.DataFrame(matrix, index=subset_words, columns=subset_words)

# Display a slice of the probability table for common words
sample_words = ['<s>', 'the', 'and', 'i', 'of', 'whale']
prob_matrix = build_prob_table(sample_words, bigram_counts, unigram_counts)

print("\nBigram Conditional Probability Table (Sample):")
print(prob_matrix)
```

```
Bigram Conditional Probability Table (Sample):
            <s>       the       and         i        of     whale
<s>    0.001192  0.010727  0.004768  0.002384  0.002384  0.003576
the    0.001253  0.001253  0.001253  0.001253  0.001253  0.005013
and    0.001307  0.005229  0.001307  0.001307  0.001307  0.001307
i      0.001381  0.001381  0.001381  0.001381  0.001381  0.001381
of     0.001295  0.012953  0.001295  0.001295  0.001295  0.001295
whale  0.001362  0.001362  0.001362  0.001362  0.001362  0.001362
```

```
# 1. Generate Trigram Counts
# We iterate through our processed sentences to find triplets
trigram_list = []
for sent in processed_train:
    # ngrams(sent, 3) creates tuples of (word1, word2, word3)
    trigram_list.extend(list(ngrams(sent, 3)))

trigram_counts = Counter(trigram_list)

# 2. Trigram Probability Function with Laplace Smoothing
def get_trigram_prob(word, w_minus_1, w_minus_2, tri_counts, bi_counts, vocab_size):
    # The numerator is the count of the full trigram (context + current word)
    trigram_key = (w_minus_2, w_minus_1, word)
    n_gram_count = tri_counts.get(trigram_key, 0)

    # The denominator is the count of the bigram context (the prefix)
    context_key = (w_minus_2, w_minus_1)
    context_count = bi_counts.get(context_key, 0)

    # Laplace Smoothing: add 1 to numerator, add V to denominator
    probability = (n_gram_count + 1) / (context_count + vocab_size)
    return probability

# 3. Quick Test: What is the probability of 'ishmael' following 'call me'?
v_size = len(unigram_counts)
test_prob = get_trigram_prob('ishmael', 'me', 'call', trigram_counts, bigram_counts, v_size)

print(f"Trigram Count for ('call', 'me', 'ishmael'): {trigram_counts[('call', 'me', 'ishmael')]}")
print(f"P(ishmael | call, me): {test_prob:.6f}")
```

```
Trigram Count for ('call', 'me', 'ishmael'): 0
P(ishmael | call, me): 0.001395
```

```
# Create a summary table for the top 10 trigrams
top_trigrams = trigram_counts.most_common(10)
tri_records = []

for (w1, w2, w3), count in top_trigrams:
    # Calculate smoothed probability
    prob = get_trigram_prob(w3, w2, w1, trigram_counts, bigram_counts, v_size)
    tri_records.append({
        'Context (w-2, w-1)': f"({w1}, {w2})",
        'Current Word (w)': w3,
        'Count': count,
        'Smoothed Prob': round(prob, 6)
    })

trigram_df = pd.DataFrame(tri_records)
print("\nTop 10 Trigram Frequency & Probability Table:")
print(trigram_df)
```

```
Top 10 Trigram Frequency & Probability Table:
  Context (w-2, w-1) Current Word (w)  Count  Smoothed Prob
0     (supplied, by)                a      2       0.004172
1         (it, will)               be      2       0.004167
2      (poor, devil)               of      2       0.004172
3        (devil, of)                a      2       0.004172
4            (of, a)           subsub      2       0.004149
5        (much, the)             more      2       0.004172
6         (of, this)         monsters      2       0.004167
7          (let, us)              fly      2       0.004172
8     (quantity, of)              oil      2       0.004172
9        (<s>, ibid)             </s>      2       0.004172
```

```
def calculate_smoothed_prob(count_ngram, count_context, vocab_size):
    """
    Implements Laplace (Add-One) Smoothing.
    Formula: (Count(sequence) + 1) / (Count(context) + V)
    """
    numerator = count_ngram + 1
    denominator = count_context + vocab_size
    return numerator / denominator

# Example usage for a Bigram: P(whale | the)
word_of_interest = "whale"
context = "the"

# Get counts from our previously built dictionaries
ngram_count = bigram_counts.get((context, word_of_interest), 0)
context_count = unigram_counts.get(context, 0)
v_size = len(unigram_counts)

smoothed_p = calculate_smoothed_prob(ngram_count, context_count, v_size)

print(f"Count of ('the', 'whale'): {ngram_count}")
print(f"Smoothed Probability P(whale | the): {smoothed_p:.6f}")
```

```
Count of ('the', 'whale'): 3
Smoothed Probability P(whale | the): 0.005013
```

```
import math

def get_model_probabilities(sentence, unigram_counts, bigram_counts, trigram_counts):
    v_size = len(unigram_counts)
    n = len(sentence)

    # 1. Unigram Probability: P(w1) * P(w2) * ...
    uni_prob = 1.0
    for word in sentence:
        uni_prob *= (unigram_counts.get(word, 0) + 1) / (sum(unigram_counts.values()) + v_size)

    # 2. Bigram Probability: P(w1|<s>) * P(w2|w1) ...
    bi_prob = 1.0
    for bg in ngrams(sentence, 2):
        cnt_bg = bigram_counts.get(bg, 0)
        cnt_uni = unigram_counts.get(bg[0], 0)
        bi_prob *= (cnt_bg + 1) / (cnt_uni + v_size)

    # 3. Trigram Probability: P(w3|w1,w2) ...
    tri_prob = 1.0
    for tg in ngrams(sentence, 3):
```

```
            cnt_tg = trigram_counts.get(tg, 0)
            cnt_bg = bigram_counts.get((tg[0], tg[1]), 0)
            tri_prob *= (cnt_tg + 1) / (cnt_bg + v_size)

        return uni_prob, bi_prob, tri_prob

    # Select 5 sentences from the test set
    test_samples = processed_test[:5]
    results = []

    for i, sent in enumerate(test_samples):
        u_p, b_p, t_p = get_model_probabilities(sent, unigram_counts, bigram_counts, trigram_counts)
        results.append({
            'Sentence': " ".join(sent[1:-1]), # Hide the <s> tokens for display
            'Unigram Prob': f"{u_p:.2e}",
            'Bigram Prob': f"{b_p:.2e}",
            'Trigram Prob': f"{t_p:.2e}"
        })

    # Display as a Table
    prob_results_df = pd.DataFrame(results)
    print(prob_results_df)
```

```
                                          Sentence Unigram Prob Bigram Prob  \
0                                               ad     2.88e-06    3.32e-06
1                 whales in the sea gods voice obey     1.56e-21    5.08e-22
2                                       n e primer     3.05e-13    3.23e-12
3  we saw also abundance of large whales there be...     1.72e-86    1.49e-90
4          captain cowleys voyage round the globe ad     1.90e-24    2.19e-23

  Trigram Prob
0     1.39e-03
1     2.02e-20
2     2.71e-09
3     2.98e-89
4     1.03e-20
```

```
    import numpy as np

    def calculate_perplexity(sentence, model_type='unigram'):
        # N is the number of tokens (excluding the start token context)
        N = len(sentence)
        log_prob_sum = 0
        v_size = len(unigram_counts)

        if model_type == 'unigram':
            for word in sentence:
                # P(word)
                p = (unigram_counts.get(word, 0) + 1) / (sum(unigram_counts.values()) + v_size)
                log_prob_sum += np.log(p)

        elif model_type == 'bigram':
            for bg in ngrams(sentence, 2):
                # P(w2 | w1)
                p = (bigram_counts.get(bg, 0) + 1) / (unigram_counts.get(bg[0], 0) + v_size)
                log_prob_sum += np.log(p)

        elif model_type == 'trigram':
            for tg in ngrams(sentence, 3):
                # P(w3 | w1, w2)
                p = (trigram_counts.get(tg, 0) + 1) / (bigram_counts.get((tg[0], tg[1]), 0) + v_size)
                log_prob_sum += np.log(p)
```

```
        # Handle case where log_prob_sum is 0 or trigram couldn't be formed
        if log_prob_sum == 0: return float('inf')

        # Perplexity formula: exp( - (1/N) * sum(log(P)) )
        perplexity = np.exp(-log_prob_sum / N)
        return perplexity

    # Evaluate the 5 sample sentences
    perp_results = []
    for sent in test_samples:
        u_pp = calculate_perplexity(sent, 'unigram')
        b_pp = calculate_perplexity(sent, 'bigram')
        t_pp = calculate_perplexity(sent, 'trigram')

        perp_results.append({
            'Sentence': " ".join(sent[1:-1]),
            'Unigram PP': round(u_pp, 2),
            'Bigram PP': round(b_pp, 2),
            'Trigram PP': round(t_pp, 2)
        })

    perp_df = pd.DataFrame(perp_results)
    print("Perplexity Comparison for 5 Test Sentences:")
    print(perp_df)
```

```
Perplexity Comparison for 5 Test Sentences:
                                          Sentence  Unigram PP  Bigram PP  \
0                                                ad       70.31      67.06
1                 whales in the sea gods voice obey      205.04     232.30
2                                       n e primer      318.54     198.69
3  we saw also abundance of large whales there be...      397.11     527.22
4          captain cowleys voyage round the globe ad      432.31     329.41

   Trigram PP
0        8.95
1      154.31
2       51.68
3      481.49
4      166.33
```