BATCH 03

TASK 01 : APPLICATION OF N-GRAM MODEL TO THE TEXT

```
import nltk
import collections
import math
import string
import pandas as pd
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
text=["Iam working as a professor.",
         "Iam working in SR University."]
tokens=[nltk.word_tokenize(sent) for sent in text]
print(tokens)
from nltk.corpus import stopwords
def convert_to_lowercase(text):
    return [s.lower() for s in text]
def remove_punctuation_and_numbers(text):
    translator = str.maketrans('', '', string.punctuation + string.digits)
    return [s.translate(translator) for s in text]
def tokenize_words(text):
    return [nltk.word_tokenize(s) for s in text]
def remove_stopwords_from_tokens(tokenized_text, language='english'):
    stop_words = set(stopwords.words(language))
    return [[word for word in tokens if word not in stop_words] for tokens in tokenized_text]
def add_start_end_tokens(tokenized_text, start_token='<s>', end_token='</s>'):
    return [[start_token] + tokens + [end_token] for tokens in tokenized_text]
text_lower = convert_to_lowercase(text)
print("1. Lowercase:", text_lower)
text_no_punc_nums = remove_punctuation_and_numbers(text_lower)
print("2. No Punctuation/Numbers:", text_no_punc_nums)
text_tokenized = tokenize_words(text_no_punc_nums)
print("3. Tokenized:", text_tokenized)
text_no_stopwords = remove_stopwords_from_tokens(text_tokenized)
print("4. No Stopwords:", text_no_stopwords)
text_final_processed = add_start_end_tokens(text_no_stopwords)
print("5. Final Processed:", text_final_processed)
unigrams = []
for sentence_tokens in text_final_processed:
    for token in sentence_tokens:
        unigrams.append(token)
unigram_counts = collections.Counter(unigrams)
print("Unigram Counts:", unigram_counts)
bigrams = []
for sentence_tokens in text_final_processed:
    for i in range(len(sentence_tokens) - 1):
        bigrams.append((sentence_tokens[i], sentence_tokens[i+1]))
bigram_counts = collections.Counter(bigrams)
print("Bigram Counts:", bigram_counts)
trigrams = []
for sentence_tokens in text_final_processed:
    for i in range(len(sentence_tokens) - 2):
        trigrams.append((sentence_tokens[i], sentence_tokens[i+1], sentence_tokens[i+2]))
trigram_counts = collections.Counter(trigrams)
print("Trigram Counts:", trigram_counts)
bigram_probabilities = {}
for (word1, word2), count in bigram_counts.items():
    # P(word2 | word1) = Count(word1, word2) / Count(word1)
    probability = count / unigram_counts[word1]
    bigram_probabilities[(word1, word2)] = probability
print("Bigram Probabilities:")
for bigram, prob in bigram_probabilities.items():
    print(f"{bigram}: {prob:.4f}")
trigram_probabilities = {}
for (word1, word2, word3), count in trigram_counts.items():
    if (word1, word2) in bigram_counts:
        probability = count / bigram_counts[(word1, word2)]
        trigram_probabilities[(word1, word2, word3)] = probability
    else:
        trigram_probabilities[(word1, word2, word3)] = 0.0 # Or some small epsilon
print("Trigram Probabilities:")
for trigram, prob in trigram_probabilities.items():
```

```
    print(f"{trigram}: {prob:.4f}")
unigram_df = pd.DataFrame(unigram_counts.items(), columns=['Unigram', 'Count'])
unigram_df = unigram_df.sort_values(by='Count', ascending=False).reset_index(drop=True)
print("\n--- Unigram Counts ---")
print(unigram_df.to_string(index=False))
bigram_counts_df = pd.DataFrame([(bigram[0], bigram[1], count) for bigram, count in bigram_counts.items()],
                                columns=['Word1', 'Word2', 'Count'])
bigram_counts_df = bigram_counts_df.sort_values(by='Count', ascending=False).reset_index(drop=True)
print("\n--- Bigram Counts ---")
print(bigram_counts_df.to_string(index=False))
trigram_counts_df = pd.DataFrame([(trigram[0], trigram[1], trigram[2], count) for trigram, count in trigram_counts.items()],
                                columns=['Word1', 'Word2', 'Word3', 'Count'])
trigram_counts_df = trigram_counts_df.sort_values(by='Count', ascending=False).reset_index(drop=True)
print("\n--- Trigram Counts ---")
print(trigram_counts_df.to_string(index=False))
bigram_probabilities_df = pd.DataFrame([{'Word1': bigram[0], 'Word2': bigram[1], 'Probability': prob} for bigram, prob in bigram_
bigram_probabilities_df = bigram_probabilities_df.sort_values(by='Probability', ascending=False).reset_index(drop=True)
print("\n--- Bigram Probabilities ---")
print(bigram_probabilities_df.to_string(index=False))
trigram_probabilities_df = pd.DataFrame([{'Word1': trigram[0], 'Word2': trigram[1], 'Word3': trigram[2], 'Probability': prob} fo
trigram_probabilities_df = trigram_probabilities_df.sort_values(by='Probability', ascending=False).reset_index(drop=True)
print("\n--- Trigram Probabilities ---")
print(trigram_probabilities_df.to_string(index=False))
V = len(unigram_df)
print(f"Vocabulary size (V): {V}")
smoothed_bigram_probabilities = {}
for (word1, word2), count_bigram in bigram_counts.items():
    count_word1 = unigram_counts.get(word1, 0) # Get count of word1. Should always exist if bigram exists.
    smoothed_prob = (count_bigram + 1) / (count_word1 + V)
    smoothed_bigram_probabilities[(word1, word2)] = smoothed_prob
print("Smoothed Bigram Probabilities (Add-one Smoothing):")
for bigram, prob in smoothed_bigram_probabilities.items():
    print(f"{bigram}: {prob:.4f}")
smoothed_bigram_probabilities_df = pd.DataFrame([{'Word1': bigram[0], 'Word2': bigram[1], 'Smoothed Probability': prob} for bigr
smoothed_bigram_probabilities_df = smoothed_bigram_probabilities_df.sort_values(by='Smoothed Probability', ascending=False).reset
print("\n--- Smoothed Bigram Probabilities (DataFrame) ---")
print(smoothed_bigram_probabilities_df.to_string(index=False))
smoothed_trigram_probabilities = {}
for (word1, word2, word3), count_trigram in trigram_counts.items():
    count_bigram_context = bigram_counts.get((word1, word2), 0) # Count of the context (word1, word2)
    smoothed_prob = (count_trigram + 1) / (count_bigram_context + V)
    smoothed_trigram_probabilities[(word1, word2, word3)] = smoothed_prob
print("Smoothed Trigram Probabilities (Add-one Smoothing):")
for trigram, prob in smoothed_trigram_probabilities.items():
    print(f"{trigram}: {prob:.4f}")
smoothed_trigram_probabilities_df = pd.DataFrame([{'Word1': trigram[0], 'Word2': trigram[1], 'Word3': trigram[2], 'Smoothed Proba
smoothed_trigram_probabilities_df = smoothed_trigram_probabilities_df.sort_values(by='Smoothed Probability', ascending=False).res
print("\n--- Smoothed Trigram Probabilities (DataFrame) ---")
print(smoothed_trigram_probabilities_df.to_string(index=False))
```

```
[['Iam', 'working', 'as', 'a', 'professor', '.'], ['Iam', 'working', 'in', 'SR', 'University', '.']]
1. Lowercase: ['iam working as a professor.', 'iam working in sr university.']
2. No Punctuation/Numbers: ['iam working as a professor', 'iam working in sr university']
3. Tokenized: [['iam', 'working', 'as', 'a', 'professor'], ['iam', 'working', 'in', 'sr', 'university']]
4. No Stopwords: [['iam', 'working', 'professor'], ['iam', 'working', 'sr', 'university']]
5. Final Processed: [['<s>', 'iam', 'working', 'professor', '</s>'], ['<s>', 'iam', 'working', 'sr', 'university', '</s>']]
Unigram Counts: Counter({'<s>': 2, 'iam': 2, 'working': 2, '</s>': 2, 'professor': 1, 'sr': 1, 'university': 1})
Bigram Counts: Counter({('<s>', 'iam'): 2, ('iam', 'working'): 2, ('working', 'professor'): 1, ('professor', '</s>'): 1, ('wor
Trigram Counts: Counter({('<s>', 'iam', 'working'): 2, ('iam', 'working', 'professor'): 1, ('working', 'professor', '</s>'): 1
Bigram Probabilities:
('<s>', 'iam'): 1.0000
('iam', 'working'): 1.0000
('working', 'professor'): 0.5000
('professor', '</s>'): 1.0000
('working', 'sr'): 0.5000
('sr', 'university'): 1.0000
('university', '</s>'): 1.0000
Trigram Probabilities:
('<s>', 'iam', 'working'): 1.0000
('iam', 'working', 'professor'): 0.5000
('working', 'professor', '</s>'): 1.0000
('iam', 'working', 'sr'): 0.5000
('working', 'sr', 'university'): 1.0000
('sr', 'university', '</s>'): 1.0000

--- Unigram Counts ---
   Unigram  Count
       <s>      2
       iam      2
```

```
   working      2
      </s>      2
 professor      1
        sr      1
university      1

--- Bigram Counts ---
     Word1      Word2  Count
       <s>        iam      2
       iam    working      2
   working  professor      1
 professor       </s>      1
   working         sr      1
        sr university      1
university       </s>      1

--- Trigram Counts ---
  Word1      Word2      Word3  Count
    <s>        iam    working      2
    iam    working  professor      1
working  professor       </s>      1
    iam    working         sr      1
working         sr university      1
     sr university       </s>      1

--- Bigram Probabilities ---
     Word1      Word2  Probability
       <s>        iam          1.0
```

TASK 02 : TESTING THE 5 SENTENCES

```
test_sentences = [
    'This is a test sentence.',
    'Another example sentence.',
    'How are you?',
    'I am learning NLP.',
    'The quick brown fox jumps over the lazy dog.',
    'University professors teach many students.'
]
test_sentences_lower = convert_to_lowercase(test_sentences)
print("1. Lowercase:", test_sentences_lower)
test_sentences_no_punc_nums = remove_punctuation_and_numbers(test_sentences_lower)
print("2. No Punctuation/Numbers:", test_sentences_no_punc_nums)
test_sentences_tokenized = tokenize_words(test_sentences_no_punc_nums)
print("3. Tokenized:", test_sentences_tokenized)
test_sentences_no_stopwords = remove_stopwords_from_tokens(test_sentences_tokenized)
print("4. No Stopwords:", test_sentences_no_stopwords)
test_sentences_final_processed = add_start_end_tokens(test_sentences_no_stopwords)
print("5. Final Processed:", test_sentences_final_processed)
total_unigram_count = sum(unigram_counts.values())
print(f"Total number of unigrams (N) in training data: {total_unigram_count}")
smoothed_unigram_probabilities = {}
for word, count in unigram_counts.items():
    smoothed_prob = (count + 1) / (total_unigram_count + V)
    smoothed_unigram_probabilities[word] = smoothed_prob
print("Smoothed Unigram Probabilities (Add-one Smoothing):")
for word, prob in smoothed_unigram_probabilities.items():
    print(f"{word}: {prob:.4f}")
def calculate_unigram_probability(sentence_tokens, smoothed_unigram_probabilities, total_unigram_count, V):
    sentence_probability = 1.0
    oov_probability = 1 / (total_unigram_count + V)
    for word in sentence_tokens:
        prob = smoothed_unigram_probabilities.get(word, oov_probability)
        sentence_probability *= prob
    return sentence_probability
example_sentence = test_sentences_final_processed[0] # ['<s>', 'test', 'sentence', '</s>']
sentence_prob = calculate_unigram_probability(example_sentence,
                                              smoothed_unigram_probabilities,
                                              total_unigram_count,
                                              V)
print(f"Example Sentence: {example_sentence}")
print(f"Smoothed Unigram Model Probability: {sentence_prob:.8f}")
example_sentence_with_oov = test_sentences_final_processed[4] # ['<s>', 'quick', 'brown', 'fox', 'jumps', 'lazy', 'dog', '</s>']
sentence_prob_oov = calculate_unigram_probability(example_sentence_with_oov,
                                                  smoothed_unigram_probabilities,
                                                  total_unigram_count,
                                                  V)
print(f"Example Sentence with OOV: {example_sentence_with_oov}")
print(f"Smoothed Unigram Model Probability (with OOV): {sentence_prob_oov:.8f}")
```

```
print(f"Smoothed Unigram Model Probability (with OOV): {sentence_prob_oov:.8f}")
def calculate_bigram_probability(sentence_tokens, smoothed_unigram_probabilities, smoothed_bigram_probabilities, unigram_counts,
    sentence_probability = 1.0
    oov_unigram_prob = 1 / (total_unigram_count + V)
    if not sentence_tokens:
        return 0.0 # Return 0 for an empty sentence
    first_word = sentence_tokens[0]
    sentence_probability *= smoothed_unigram_probabilities.get(first_word, oov_unigram_prob)
    for i in range(1, len(sentence_tokens)):
        current_word = sentence_tokens[i]
        previous_word = sentence_tokens[i-1]
        bigram_prob = smoothed_bigram_probabilities.get((previous_word, current_word))
        if bigram_prob is None:
            count_previous_word = unigram_counts.get(previous_word, 0)
            bigram_prob = 1 / (count_previous_word + V)
        sentence_probability *= bigram_prob
    return sentence_probability
example_sentence_bigram = test_sentences_final_processed[0] # ['<s>', 'test', 'sentence', '</s>']
sentence_prob_bigram = calculate_bigram_probability(
    example_sentence_bigram,
    smoothed_unigram_probabilities,
    smoothed_bigram_probabilities,
    unigram_counts,
    total_unigram_count,
    V
)
print(f"\nExample Sentence for Bigram Model: {example_sentence_bigram}")
print(f"Smoothed Bigram Model Probability: {sentence_prob_bigram:.8f}")
example_sentence_bigram_oov = test_sentences_final_processed[4] # ['<s>', 'quick', 'brown', 'fox', 'jumps', 'lazy', 'dog', '</s>
sentence_prob_bigram_oov = calculate_bigram_probability(
    example_sentence_bigram_oov,
    smoothed_unigram_probabilities,
    smoothed_bigram_probabilities,
    unigram_counts,
    total_unigram_count,
    V
)
print(f"Example Sentence with OOV for Bigram Model: {example_sentence_bigram_oov}")
print(f"Smoothed Bigram Model Probability (with OOV): {sentence_prob_bigram_oov:.8f}")
empty_sentence = []
sentence_prob_empty = calculate_bigram_probability(
    empty_sentence,
    smoothed_unigram_probabilities,
    smoothed_bigram_probabilities,
    unigram_counts,
    total_unigram_count,
    V
)
print(f"Example Empty Sentence: {empty_sentence}")
print(f"Smoothed Bigram Model Probability (empty sentence): {sentence_prob_empty:.8f}")
def calculate_trigram_probability(sentence_tokens, smoothed_unigram_probabilities, smoothed_bigram_probabilities, smoothed_trigr
    sentence_probability = 1.0
    oov_unigram_prob = 1 / (total_unigram_count + V)
    if not sentence_tokens:
        return 0.0 # Return 0 for an empty sentence
    first_word = sentence_tokens[0]
    sentence_probability *= smoothed_unigram_probabilities.get(first_word, oov_unigram_prob)
    if len(sentence_tokens) >= 2:
        second_word = sentence_tokens[1]
        previous_word_for_bigram = first_word
        bigram_prob = smoothed_bigram_probabilities.get((previous_word_for_bigram, second_word))
        if bigram_prob is None:
            count_previous_word = unigram_counts.get(previous_word_for_bigram, 0)
            bigram_prob = 1 / (count_previous_word + V)
        sentence_probability *= bigram_prob
    for i in range(2, len(sentence_tokens)):
        current_word = sentence_tokens[i]
        prev_word1 = sentence_tokens[i-2] # w_i-2
        prev_word2 = sentence_tokens[i-1] # w_i-1
        trigram_prob = smoothed_trigram_probabilities.get((prev_word1, prev_word2, current_word))
        if trigram_prob is None:
            count_bigram_context = bigram_counts.get((prev_word1, prev_word2), 0)
            trigram_prob = 1 / (count_bigram_context + V)
        sentence_probability *= trigram_prob
    return sentence_probability
example_sentence_trigram = test_sentences_final_processed[0] # ['<s>', 'test', 'sentence', '</s>']
sentence_prob_trigram = calculate_trigram_probability(
    example_sentence_trigram,
```

```
        smoothed_unigram_probabilities,
        smoothed_bigram_probabilities,
        smoothed_trigram_probabilities,
        unigram_counts,
        bigram_counts,
        total_unigram_count,
        V
    )
    print(f"\nExample Sentence for Trigram Model: {example_sentence_trigram}")
    print(f"Smoothed Trigram Model Probability: {sentence_prob_trigram:.8f}")
    example_sentence_trigram_oov = test_sentences_final_processed[4] # ['<s>', 'quick', 'brown', 'fox', 'jumps', 'lazy', 'dog', '</s
    sentence_prob_trigram_oov = calculate_trigram_probability(
        example_sentence_trigram_oov,
        smoothed_unigram_probabilities,
        smoothed_bigram_probabilities,
        smoothed_trigram_probabilities,
        unigram_counts,
        bigram_counts,
        total_unigram_count,
        V
    )
    print(f"Example Sentence with OOV for Trigram Model: {example_sentence_trigram_oov}")
    print(f"Smoothed Trigram Model Probability (with OOV): {sentence_prob_trigram_oov:.8f}")
    empty_sentence = []
    sentence_prob_empty_trigram = calculate_trigram_probability(
        empty_sentence,
        smoothed_unigram_probabilities,
        smoothed_bigram_probabilities,
        smoothed_trigram_probabilities,
        unigram_counts,
        bigram_counts,
        total_unigram_count,
        V
    )
    print(f"Example Empty Sentence: {empty_sentence}")
    print(f"Smoothed Trigram Model Probability (empty sentence): {sentence_prob_empty_trigram:.8f}")
    print("\n--- Sentence Probabilities --- ")
    for i, original_sentence in enumerate(test_sentences):
        preprocessed_sentence = test_sentences_final_processed[i]
        print(f"\nOriginal Sentence: {original_sentence}")
        print(f"Preprocessed Tokens: {preprocessed_sentence}")
        unigram_prob = calculate_unigram_probability(
            preprocessed_sentence,
            smoothed_unigram_probabilities,
            total_unigram_count,
            V
        )
        print(f"  Smoothed Unigram Model Probability: {unigram_prob:.8f}")
        bigram_prob = calculate_bigram_probability(
            preprocessed_sentence,
            smoothed_unigram_probabilities,
            smoothed_bigram_probabilities,
            unigram_counts,
            total_unigram_count,
            V
        )
        print(f"  Smoothed Bigram Model Probability: {bigram_prob:.8f}")
        trigram_prob = calculate_trigram_probability(
            preprocessed_sentence,
            smoothed_unigram_probabilities,
            smoothed_bigram_probabilities,
            smoothed_trigram_probabilities,
            unigram_counts,
            bigram_counts,
            total_unigram_count,
            V
        )
        print(f"  Smoothed Trigram Model Probability: {trigram_prob:.8f}")
```

```
1. Lowercase: ['this is a test sentence.', 'another example sentence.', 'how are you?', 'i am learning nlp.', 'the quick brown
2. No Punctuation/Numbers: ['this is a test sentence', 'another example sentence', 'how are you', 'i am learning nlp', 'the qu
3. Tokenized: [['this', 'is', 'a', 'test', 'sentence'], ['another', 'example', 'sentence'], ['how', 'are', 'you'], ['i', 'am',
4. No Stopwords: [['test', 'sentence'], ['another', 'example', 'sentence'], [], ['learning', 'nlp'], ['quick', 'brown', 'fox',
5. Final Processed: [['<s>', 'test', 'sentence', '</s>'], ['<s>', 'another', 'example', 'sentence', '</s>'], ['<s>', '</s>'],
Total number of unigrams (N) in training data: 11
Smoothed Unigram Probabilities (Add-one Smoothing):
<s>: 0.1667
iam: 0.1667
```

```
working: 0.1667
professor: 0.1111
</s>: 0.1667
sr: 0.1111
university: 0.1111
Example Sentence: ['<s>', 'test', 'sentence', '</s>']
Smoothed Unigram Model Probability: 0.00008573
Example Sentence with OOV: ['<s>', 'quick', 'brown', 'fox', 'jumps', 'lazy', 'dog', '</s>']
Smoothed Unigram Model Probability (with OOV): 0.00000000

Example Sentence for Bigram Model: ['<s>', 'test', 'sentence', '</s>']
Smoothed Bigram Model Probability: 0.00037793
Example Sentence with OOV for Bigram Model: ['<s>', 'quick', 'brown', 'fox', 'jumps', 'lazy', 'dog', '</s>']
Smoothed Bigram Model Probability (with OOV): 0.00000016
Example Empty Sentence: []
Smoothed Bigram Model Probability (empty sentence): 0.00000000

Example Sentence for Trigram Model: ['<s>', 'test', 'sentence', '</s>']
Smoothed Trigram Model Probability: 0.00037793
Example Sentence with OOV for Trigram Model: ['<s>', 'quick', 'brown', 'fox', 'jumps', 'lazy', 'dog', '</s>']
Smoothed Trigram Model Probability (with OOV): 0.00000016
Example Empty Sentence: []
Smoothed Trigram Model Probability (empty sentence): 0.00000000

--- Sentence Probabilities ---

Original Sentence: This is a test sentence.
Preprocessed Tokens: ['<s>', 'test', 'sentence', '</s>']
  Smoothed Unigram Model Probability: 0.00008573
  Smoothed Bigram Model Probability: 0.00037793
  Smoothed Trigram Model Probability: 0.00037793

Original Sentence: Another example sentence.
Preprocessed Tokens: ['<s>', 'another', 'example', 'sentence', '</s>']
  Smoothed Unigram Model Probability: 0.00000476
  Smoothed Bigram Model Probability: 0.00005399
  Smoothed Trigram Model Probability: 0.00005399

Original Sentence: How are you?
Preprocessed Tokens: ['<s>', '</s>']
  Smoothed Unigram Model Probability: 0.02777778
  Smoothed Bigram Model Probability: 0.01851852
  Smoothed Trigram Model Probability: 0.01851852

Original Sentence: I am learning NLP.
Preprocessed Tokens: ['<s>', 'learning', 'nlp', '</s>']
  Smoothed Unigram Model Probability: 0.00008573
  Smoothed Bigram Model Probability: 0.00037793
```

TASK 03 : Perplexity Calculation

```
import math
import pandas as pd
import collections
import string
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
text=["Iam working as a professor.",
         "Iam working in SR University."]
def convert_to_lowercase(text):
    return [s.lower() for s in text]
def remove_punctuation_and_numbers(text):
    translator = str.maketrans('', '', string.punctuation + string.digits)
    return [s.translate(translator) for s in text]
def tokenize_words(text):
    return [nltk.word_tokenize(s) for s in text]
def remove_stopwords_from_tokens(tokenized_text, language='english'):
    stop_words = set(stopwords.words(language))
    return [[word for word in tokens if word not in stop_words] for tokens in tokenized_text]
def add_start_end_tokens(tokenized_text, start_token='<s>', end_token='</s>'):
    return [[start_token] + tokens + [end_token] for tokens in tokenized_text]

# Apply preprocessing to training data
text_lower = convert_to_lowercase(text)
text_no_punc_nums = remove_punctuation_and_numbers(text_lower)
text_tokenized = tokenize_words(text_no_punc_nums)
text_no_stopwords = remove_stopwords_from_tokens(text_tokenized)
text_final_processed = add_start_end_tokens(text_no_stopwords)
```

```
unigrams = []
for sentence_tokens in text_final_processed:
    for token in sentence_tokens:
        unigrams.append(token)
unigram_counts = collections.Counter(unigrams)
bigrams = []
for sentence_tokens in text_final_processed:
    for i in range(len(sentence_tokens) - 1):
        bigrams.append((sentence_tokens[i], sentence_tokens[i+1]))
bigram_counts = collections.Counter(bigrams)
trigrams = []
for sentence_tokens in text_final_processed:
    for i in range(len(sentence_tokens) - 2):
        trigrams.append((sentence_tokens[i], sentence_tokens[i+1], sentence_tokens[i+2]))
trigram_counts = collections.Counter(trigrams)
unigram_df = pd.DataFrame(unigram_counts.items(), columns=['Unigram', 'Count'])
V = len(unigram_df)
smoothed_bigram_probabilities = {}
for (word1, word2), count_bigram in bigram_counts.items():
    count_word1 = unigram_counts.get(word1, 0)
    smoothed_prob = (count_bigram + 1) / (count_word1 + V)
    smoothed_bigram_probabilities[(word1, word2)] = smoothed_prob
smoothed_trigram_probabilities = {}
for (word1, word2, word3), count_trigram in trigram_counts.items():
    count_bigram_context = bigram_counts.get((word1, word2), 0)
    smoothed_prob = (count_trigram + 1) / (count_bigram_context + V)
    smoothed_trigram_probabilities[(word1, word2, word3)] = smoothed_prob
test_sentences = [
    'This is a test sentence.',
    'Another example sentence.',
    'How are you?',
    'I am learning NLP.',
    'The quick brown fox jumps over the lazy dog.',
    'University professors teach many students.'
]
test_sentences_lower = convert_to_lowercase(test_sentences)
test_sentences_no_punc_nums = remove_punctuation_and_numbers(test_sentences_lower)
test_sentences_tokenized = tokenize_words(test_sentences_no_punc_nums)
test_sentences_no_stopwords = remove_stopwords_from_tokens(test_sentences_tokenized)
test_sentences_final_processed = add_start_end_tokens(test_sentences_no_stopwords)
total_unigram_count = sum(unigram_counts.values())
smoothed_unigram_probabilities = {}
for word, count in unigram_counts.items():
    smoothed_prob = (count + 1) / (total_unigram_count + V)
    smoothed_unigram_probabilities[word] = smoothed_prob
def calculate_unigram_probability(sentence_tokens, smoothed_unigram_probabilities, total_unigram_count, V):
    sentence_probability = 1.0
    oov_probability = 1 / (total_unigram_count + V)
    for word in sentence_tokens:
        prob = smoothed_unigram_probabilities.get(word, oov_probability)
        sentence_probability *= prob
    return sentence_probability
def calculate_bigram_probability(sentence_tokens, smoothed_unigram_probabilities, smoothed_bigram_probabilities, unigram_counts,
    sentence_probability = 1.0
    oov_unigram_prob = 1 / (total_unigram_count + V)
    if not sentence_tokens:
        return 0.0
    first_word = sentence_tokens[0]
    sentence_probability *= smoothed_unigram_probabilities.get(first_word, oov_unigram_prob)
    for i in range(1, len(sentence_tokens)):
        current_word = sentence_tokens[i]
        previous_word = sentence_tokens[i-1]
        bigram_prob = smoothed_bigram_probabilities.get((previous_word, current_word))
        if bigram_prob is None:
            count_previous_word = unigram_counts.get(previous_word, 0)
            bigram_prob = 1 / (count_previous_word + V)
        sentence_probability *= bigram_prob
    return sentence_probability
def calculate_trigram_probability(sentence_tokens, smoothed_unigram_probabilities, smoothed_bigram_probabilities, smoothed_trigr
    sentence_probability = 1.0
    oov_unigram_prob = 1 / (total_unigram_count + V)
    if not sentence_tokens:
        return 0.0
    first_word = sentence_tokens[0]
    sentence_probability *= smoothed_unigram_probabilities.get(first_word, oov_unigram_prob)
    if len(sentence_tokens) >= 2:
        second_word = sentence_tokens[1]
```

```
        previous_word_for_bigram = first_word
        bigram_prob = smoothed_bigram_probabilities.get((previous_word_for_bigram, second_word))
        if bigram_prob is None:
            count_previous_word = unigram_counts.get(previous_word_for_bigram, 0)
            bigram_prob = 1 / (count_previous_word + V)
        sentence_probability *= bigram_prob
    for i in range(2, len(sentence_tokens)):
        current_word = sentence_tokens[i]
        prev_word1 = sentence_tokens[i-2]
        prev_word2 = sentence_tokens[i-1]
        trigram_prob = smoothed_trigram_probabilities.get((prev_word1, prev_word2, current_word))
        if trigram_prob is None:
            count_bigram_context = bigram_counts.get((prev_word1, prev_word2), 0)
            trigram_prob = 1 / (count_bigram_context + V)
        sentence_probability *= trigram_prob
    return sentence_probability
def get_num_actual_words(tokenized_sentence):
    return max(0, len(tokenized_sentence) - 2)
def calculate_perplexity(sentence_probability, num_actual_words):
    if num_actual_words == 0:
        return float('inf')
    if sentence_probability <= 0:
        return float('inf')
    return (1.0 / sentence_probability)**(1.0 / num_actual_words)
print("\n--- Perplexity Calculation --- ")
all_unigram_perplexities = []
all_bigram_perplexities = []
all_trigram_perplexities = []
for i, original_sentence in enumerate(test_sentences):
    preprocessed_sentence = test_sentences_final_processed[i]
    num_actual_words = get_num_actual_words(preprocessed_sentence)
    unigram_prob = calculate_unigram_probability(
        preprocessed_sentence,
        smoothed_unigram_probabilities,
        total_unigram_count,
        V
    )
    unigram_perplexity = calculate_perplexity(unigram_prob, num_actual_words)
    all_unigram_perplexities.append(unigram_perplexity)
    bigram_prob = calculate_bigram_probability(
        preprocessed_sentence,
        smoothed_unigram_probabilities,
        smoothed_bigram_probabilities,
        unigram_counts,
        total_unigram_count,
        V
    )
    bigram_perplexity = calculate_perplexity(bigram_prob, num_actual_words)
    all_bigram_perplexities.append(bigram_perplexity)
    trigram_prob = calculate_trigram_probability(
        preprocessed_sentence,
        smoothed_unigram_probabilities,
        smoothed_bigram_probabilities,
        smoothed_trigram_probabilities,
        unigram_counts,
        bigram_counts,
        total_unigram_count,
        V
    )
    trigram_perplexity = calculate_perplexity(trigram_prob, num_actual_words)
    all_trigram_perplexities.append(trigram_perplexity)
    print(f"\nOriginal Sentence: {original_sentence}")
    print(f"Preprocessed Tokens: {preprocessed_sentence}")
    print(f"  Number of Actual Words: {num_actual_words}")
    print(f"  Unigram Perplexity: {unigram_perplexity:.4f}")
    print(f"  Bigram Perplexity: {bigram_perplexity:.4f}")
    print(f"  Trigram Perplexity: {trigram_perplexity:.4f}")
perplexity_data = {
    'Original Sentence': test_sentences,
    'Unigram Perplexity': all_unigram_perplexities,
    'Bigram Perplexity': all_bigram_perplexities,
    'Trigram Perplexity': all_trigram_perplexities
}
perplexity_df = pd.DataFrame(perplexity_data)
print("\n--- Summary of Perplexities for Test Sentences ---")
display(perplexity_df)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.

--- Perplexity Calculation ---

Original Sentence: This is a test sentence.
Preprocessed Tokens: ['<s>', 'test', 'sentence', '</s>']
  Number of Actual Words: 2
  Unigram Perplexity: 108.0000
  Bigram Perplexity: 51.4393
  Trigram Perplexity: 51.4393

Original Sentence: Another example sentence.
Preprocessed Tokens: ['<s>', 'another', 'example', 'sentence', '</s>']
  Number of Actual Words: 3
  Unigram Perplexity: 59.4347
  Bigram Perplexity: 26.4583
  Trigram Perplexity: 26.4583

Original Sentence: How are you?
Preprocessed Tokens: ['<s>', '</s>']
  Number of Actual Words: 0
  Unigram Perplexity: inf
  Bigram Perplexity: inf
  Trigram Perplexity: inf

Original Sentence: I am learning NLP.
Preprocessed Tokens: ['<s>', 'learning', 'nlp', '</s>']
  Number of Actual Words: 2
  Unigram Perplexity: 108.0000
  Bigram Perplexity: 51.4393
  Trigram Perplexity: 51.4393

Original Sentence: The quick brown fox jumps over the lazy dog.
Preprocessed Tokens: ['<s>', 'quick', 'brown', 'fox', 'jumps', 'lazy', 'dog', '</s>']
  Number of Actual Words: 6
  Unigram Perplexity: 32.7082
  Bigram Perplexity: 13.6091
  Trigram Perplexity: 13.6091

Original Sentence: University professors teach many students.
Preprocessed Tokens: ['<s>', 'university', 'professors', 'teach', 'many', 'students', '</s>']
  Number of Actual Words: 5
  Unigram Perplexity: 32.0868
  Bigram Perplexity: 15.9652
  Trigram Perplexity: 15.5445

--- Summary of Perplexities for Test Sentences ---
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

	Original Sentence	Unigram Perplexity	Bigram Perplexity	Trigram Perplexity
0	This is a test sentence.	108.000000	51.439285	51.439285
1	Another example sentence.	59.434690	26.458342	26.458342
2	How are you?	inf	inf	inf
3	I am learning NLP.	108.000000	51.439285	51.439285
4	The quick brown fox jumps over the lazy dog.	32.708171	13.609129	13.609129
5	University professors teach many students.	32.086844	15.965230	15.544501

TASK 04: Comparison and Analysis

1. Which model gave the lowest perplexity?

Trigram models usually give the lowest perplexity

Followed by bigram models

Unigram models typically have the highest perplexity

2. Did trigrams always perform best?

No — not always.

3. What happens when unseen words appear?

Unseen words are called Out-Of-Vocabulary (OOV) words.

If no handling method is used:

The probability becomes zero

Perplexity becomes infinite

The model completely fails on that sentence.

4. How did smoothing affect results?

Smoothing dramatically improves performance.

Without smoothing:

Any unseen n-gram → probability = 0

Perplexity explodes

With smoothing:

Probability mass is redistributed

Unseen n-grams get small non-zero probability

Perplexity decreases significantly