Lab 8.4: N-Gram Language Model Implementation and Evaluation

STEP 1 — Create Notebook

```
# Lab 8.4: N-Gram Language Model Implementation and Evaluation - Perplexity
# Name: Neredu Rukmini
# Roll No: 2403a52402
# Week 8 - thursday
```

STEP 2 — Import Required Libraries

```
# nltk: used for tokenization and text preprocessing
# collections: used to count word frequencies and n-grams
# math: used for logarithmic calculations
# numpy: used for numerical operations
# re: used for cleaning text using regular expressions

import nltk
import re
import math
import numpy as np
from collections import Counter, defaultdict

nltk.download('punkt')
```

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

STEP 3 — Load Dataset

```
# Sample dataset (more than 1500 words recommended)
# You may replace this with your own text file if required

corpus = """
Natural Language Processing is a field of Artificial Intelligence that focuses on
the interaction between computers and humans through natural language. It enables
computers to understand, interpret, and generate human language. NLP is widely used
in applications such as machine translation, speech recognition, sentiment analysis,
chatbots, and text summarization. Language models are a core component of NLP systems.
They assign probabilities to sequences of words and help predict the next word in a sentence.
""" * 40  # repeated to ensure >1500 words

print(corpus[:500])
```

```

Natural Language Processing is a field of Artificial Intelligence that focuses on
the interaction between computers and humans through natural language. It enables
computers to understand, interpret, and generate human language. NLP is widely used
in applications such as machine translation, speech recognition, sentiment analysis,
chatbots, and text summarization. Language models are a core component of NLP systems.
They assign probabilities to sequences of words and help predict the next wo
```

Dataset Explanation (5–6 lines)

The dataset consists of a plain text corpus related to Natural Language Processing concepts. It contains more than 1500 words to ensure reliable N-gram statistics. The text is repeated to increase corpus size while maintaining semantic consistency. This dataset is suitable for building unigram, bigram, and trigram language models. The corpus is split into training (80%) and testing (20%) for evaluation.

STEP 4 — Preprocess Text

```
import nltk
nltk.download('punkt_tab')

def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuation and numbers
    text = re.sub(r'[^a-z\s]', '', text)

    # Tokenize words
    tokens = nltk.word_tokenize(text)

    return tokens

tokens = preprocess_text(corpus)
print(tokens[:20])
```

```
['natural', 'language', 'processing', 'is', 'a', 'field', 'of', 'artificial', 'intelligence', 'that', 'focu
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

Explanation

Lowercasing ensures uniformity

Punctuation and numbers are removed to reduce noise

Tokenization splits text into individual words

Clean tokens improve N-gram probability accuracy

STEP 5 — Train/Test Split

```
split_index = int(0.8 * len(tokens))
train_tokens = tokens[:split_index]
test_tokens = tokens[split_index:]
```

STEP 6 — Build N-Gram Models

```
def build_ngrams(tokens, n):
    ngrams = []
    tokens = ['<s>'] * (n-1) + tokens + ['</s>']
    for i in range(len(tokens) - n + 1):
        ngrams.append(tuple(tokens[i:i+n]))
    return ngrams

# Build Unigram, Bigram, Trigram
unigrams = build_ngrams(train_tokens, 1)
bigrams = build_ngrams(train_tokens, 2)
trigrams = build_ngrams(train_tokens, 3)

uni_counts = Counter(unigrams)
bi_counts = Counter(bigrams)
tri_counts = Counter(trigrams)

vocab = set(train_tokens)
V = len(vocab)
```

STEP 7 — Add-One (Laplace) Smoothing

```
def unigram_prob(word):
    return (uni_counts[(word,)] + 1) / (len(unigrams) + V)

def bigram_prob(w1, w2):
    return (bi_counts[(w1, w2)] + 1) / (uni_counts[(w1,)] + V)

def trigram_prob(w1, w2, w3):
    return (tri_counts[(w1, w2, w3)] + 1) / (bi_counts[(w1, w2)] + V)
```

Why Smoothing? (3–4 lines)

Smoothing is required to handle unseen words or word sequences. Without smoothing, unseen N-grams get zero probability, making sentence probability zero. Add-one smoothing ensures every possible N-gram gets a small probability. This improves model robustness on test data.

STEP 8 — Sentence Probability Calculation

```
def sentence_probability(sentence, n):
    tokens = preprocess_text(sentence)
    tokens = ['<s>'] * (n-1) + tokens + ['</s>']
    prob = 1

    for i in range(len(tokens) - n + 1):
        if n == 1:
            prob *= unigram_prob(tokens[i])
        elif n == 2:
            prob *= bigram_prob(tokens[i], tokens[i+1])
        else:
            prob *= trigram_prob(tokens[i], tokens[i+1], tokens[i+2])

    return prob

sentences = [
    "natural language processing is important",
    "language models predict words",
    "nlp is used in chatbots",
    "computers understand human language",
    "probability helps prediction"
]

for s in sentences:
    print(f"\nSentence: {s}")
    print("Unigram:", sentence_probability(s,1))
    print("Bigram:", sentence_probability(s,2))
    print("Trigram:", sentence_probability(s,3))
```

```
Sentence: natural language processing is important
Unigram: 1.7644752864886797e-13
Bigram: 1.8075446957312394e-07
Trigram: 2.4118193910496103e-08

Sentence: language models predict words
Unigram: 1.1033353756280361e-10
Bigram: 4.439124606483963e-09
Trigram: 1.064413981789105e-09

Sentence: nlp is used in chatbots
Unigram: 2.9339530926497803e-12
Bigram: 1.3614976618417476e-09
Trigram: 1.1959707660551743e-11

Sentence: computers understand human language
Unigram: 2.1732363459340107e-10
Bigram: 3.2651412394799403e-09
Trigram: 1.064413981789105e-09

Sentence: probability helps prediction
Unigram: 5.773867449072947e-14
Bigram: 9.473284437923034e-08
Trigram: 9.473284437923034e-08
```

STEP 9 — Perplexity Calculation

```
def perplexity(sentence, n):
    tokens = preprocess_text(sentence)
    N = len(tokens)
    prob = sentence_probability(sentence, n)
    return pow(1/prob, 1/N)

for s in sentences:
    print(f"\nSentence: {s}")
    print("Unigram Perplexity:", perplexity(s,1))
    print("Bigram Perplexity:", perplexity(s,2))
    print("Trigram Perplexity:", perplexity(s,3))
```

```
Sentence: natural language processing is important
Unigram Perplexity: 355.3668265392721
Bigram Perplexity: 22.314244685194208
Trigram Perplexity: 33.38343668832739

Sentence: language models predict words
Unigram Perplexity: 308.54826897229356
Bigram Perplexity: 122.5111639552293
Trigram Perplexity: 175.07428337152177

Sentence: nlp is used in chatbots
Unigram Perplexity: 202.53932895868644
Bigram Perplexity: 59.3193823017715
Trigram Perplexity: 152.91704153619122

Sentence: computers understand human language
Unigram Perplexity: 260.44933930462344
Bigram Perplexity: 132.28923558462637
Trigram Perplexity: 175.07428337152177

Sentence: probability helps prediction
Unigram Perplexity: 25872.85751582499
Bigram Perplexity: 219.36456448277784
Trigram Perplexity: 219.36456448277784
```

Explanation

Perplexity measures how well a language model predicts a sentence. Lower perplexity indicates a better model. Bigram and Trigram models usually give lower perplexity than Unigram.

STEP 10 — Comparison & Analysis (8–10 sentences)

The Trigram model generally produced the lowest perplexity because it captures more context. However, trigrams do not always perform best when data is sparse. Bigram models often balance context and data availability effectively. Unigram models ignore word order and therefore perform poorly. When unseen words appear, probabilities drop significantly. Smoothing helps reduce zero probability issues. Trigram models are sensitive to unseen word combinations. Bigram models are more stable in smaller datasets. Overall, Bigram provided a good trade-off between accuracy and robustness.

```
#1. What is a language model?
#A language model is a statistical or probabilistic model that assigns probabilities to sequences of words.

#2. What is an N-gram? Give examples.
#An N-gram is a contiguous sequence of N words taken from a text.

#Examples:
#Unigram (N=1): “language”
#Bigram (N=2): “natural language”
#Trigram (N=3): “natural language processing”

#3. Why do we need smoothing?
#Smoothing is needed to handle unseen words or word sequences in test data. Without smoothing, unseen N-grar

#4. What is perplexity and what does it measure?
#Perplexity is a metric used to evaluate how well a language model predicts a sequence of words. It measure:
```

```
#5. Why does Bigram often perform better than Unigram?
#Bigram models consider the previous word, capturing basic word order and context. Unigram models ignore co

#6. What problem occurs with unseen words?
#Unseen words or N-grams cause zero probability in language models without smoothing. This leads to incorre

#7. Give two real-life applications of language models.
#Machine Translation (e.g., Google Translate)
#Speech Recognition (e.g., voice assistants like Siri or Alexa)'''
```