NAME : NITHIN.V

ROLL.NO : 2403A52355

BATCH.NO : 13(AIML)

1.Import Required Libraries

```
# nltk → for tokenization and stopwords
import nltk

# numpy → numerical operations
import numpy as np

# pandas → to display tables (counts & probabilities)
import pandas as pd

# re → regular expressions for text cleaning
import re

# collections → Counter for counting n-grams
from collections import Counter

# math → for logarithmic probability & perplexity
import math

# Download required nltk resources
nltk.download('punkt')
nltk.download('stopwords')

from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
```

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

2.Load Dataset

```
# Sample text corpus (≈ 1500+ words)
# You can replace this with any text file or dataset

corpus = """
Natural language processing is a subfield of artificial intelligence.
It focuses on the interaction between computers and human language.
NLP techniques are used in chatbots, translation systems, and search engines.
Language models are a core component of NLP.
They help predict the probability of word sequences.
Machine learning and deep learning play an important role in NLP.
Statistical language models such as n-grams are simple but effective.
Unigrams, bigrams, and trigrams capture contextual information.
Smoothing techniques help handle unseen words.
Perplexity is used to evaluate language models.
""" * 150  # repeated to ensure sufficient words
# Display a sample of the dataset
print(corpus[:500])
```

```

Natural language processing is a subfield of artificial intelligence.
It focuses on the interaction between computers and human language.
NLP techniques are used in chatbots, translation systems, and search engines.
Language models are a core component of NLP.
They help predict the probability of word sequences.
Machine learning and deep learning play an important role in NLP.
Statistical language models such as n-grams are simple but effective.
Unigrams, bigrams, and trigrams capture contextua
```

3.Preprocess Text

```
nltk.download('punkt_tab')
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Convert text to lowercase
```

```
        text = text.lower()

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

        # Sentence tokenization
        sentences = sent_tokenize(text)

        processed_sentences = []

        for sentence in sentences:
            # Word tokenization
            words = word_tokenize(sentence)

            # Remove stopwords (optional)
            words = [word for word in words if word not in stop_words]

            # Add start and end tokens
            words = ['<s>'] + words + ['</s>']

            processed_sentences.append(words)

        return processed_sentences

    processed_data = preprocess_text(corpus)
    print(processed_data[:2])
```

```
 'used', 'evaluate', 'language', 'models', 'natural', 'language', 'processing', 'subfield', 'artificial', 'intelligence', 'f
```

4.Build N-Gram Models

```
    def build_ngrams(sentences, n):
        ngrams = []
        for sentence in sentences:
            for i in range(len(sentence) - n + 1):
                ngrams.append(tuple(sentence[i:i+n]))
        return ngrams
    # Build models
    unigrams = build_ngrams(processed_data, 1)
    bigrams = build_ngrams(processed_data, 2)
    trigrams = build_ngrams(processed_data, 3)

    # Count n-grams
    uni_counts = Counter(unigrams)
    bi_counts = Counter(bigrams)
    tri_counts = Counter(trigrams)
    # Convert to tables
    uni_df = pd.DataFrame(uni_counts.items(), columns=["Unigram", "Count"])
    bi_df = pd.DataFrame(bi_counts.items(), columns=["Bigram", "Count"])
    tri_df = pd.DataFrame(tri_counts.items(), columns=["Trigram", "Count"])

    uni_df.head()
```

	Unigram	Count
0	(<s>,)	1
1	(natural,)	150
2	(language,)	750
3	(processing,)	150
4	(subfield,)	150

Next steps: Generate code with uni_df New interactive sheet

5.Apply Add-One (Laplace) Smoothing

```
    vocab_size = len(uni_counts)

    def laplace_probability(ngram, ngram_counts, prev_counts=None):
        if prev_counts:
            return (ngram_counts[ngram] + 1) / (prev_counts[ngram[:-1]] + vocab_size)
        else:
```

```
        return (ngram_counts[ngram] + 1) / (sum(ngram_counts.values()) + vocab_size)
```

6.Sentence Probability Calculation

```
def sentence_probability(sentence, n, ngram_counts, prev_counts=None):
    words = ['<s>'] + word_tokenize(sentence.lower()) + ['</s>']
    prob = 1

    for i in range(len(words) - n + 1):
        ngram = tuple(words[i:i+n])
        prob *= laplace_probability(ngram, ngram_counts, prev_counts)

    return prob
sentences = [
    "language models are important",
    "nlp uses machine learning",
    "trigrams capture context",
    "smoothing helps unseen words",
    "probability estimation is crucial"
]

for s in sentences:
    print(f"Sentence: {s}")
    print("Unigram Probability:", sentence_probability(s, 1, uni_counts))
    print("Bigram Probability:", sentence_probability(s, 2, bi_counts, uni_counts))
    print("Trigram Probability:", sentence_probability(s, 3, tri_counts, bi_counts))
    print()
```

```
Sentence: language models are important
Unigram Probability: 3.7186572044891964e-16
Bigram Probability: 2.2107843137254904e-09
Trigram Probability: 1.6e-08

Sentence: nlp uses machine learning
Unigram Probability: 1.4904338462732991e-16
Bigram Probability: 1.69187675070028e-09
Trigram Probability: 4.000000000000001e-08

Sentence: trigrams capture context
Unigram Probability: 1.5006905906006969e-15
Bigram Probability: 1.4803921568627452e-06
Trigram Probability: 2.0000000000000003e-06

Sentence: smoothing helps unseen words
Unigram Probability: 2.5033614580281178e-17
Bigram Probability: 7.401960784313726e-09
Trigram Probability: 4.000000000000001e-08

Sentence: probability estimation is crucial
Unigram Probability: 1.0979173974948985e-21
Bigram Probability: 7.843137254901961e-10
Trigram Probability: 1.6000000000000003e-07
```

7.Perplexity Calculation

```
def perplexity(sentence, n, ngram_counts, prev_counts=None):
    words = ['<s>'] + word_tokenize(sentence.lower()) + ['</s>']
    log_prob = 0
    N = len(words)

    for i in range(len(words) - n + 1):
        ngram = tuple(words[i:i+n])
        prob = laplace_probability(ngram, ngram_counts, prev_counts)
        log_prob += math.log(prob)

    return math.exp(-log_prob / N)
for s in sentences:
  print(f"Sentence: {s}")
  print("Unigram Perplexity:", perplexity(s, 1, uni_counts))
  print("Bigram Perplexity:", perplexity(s, 2, bi_counts, uni_counts))
  print("Trigram Perplexity:", perplexity(s, 3, tri_counts, bi_counts))
  print()
```

```
Sentence: language models are important
Unigram Perplexity: 372.90852261566
Bigram Perplexity: 27.706113197129273
Trigram Perplexity: 19.921100948292235
```

```
Sentence: nlp uses machine learning
Unigram Perplexity: 434.29138654761437
Bigram Perplexity: 28.969336353685485
Trigram Perplexity: 17.099759466766965

Sentence: trigrams capture context
Unigram Perplexity: 922.023028387209
Bigram Perplexity: 14.652935735465366
Trigram Perplexity: 13.797296614612149

Sentence: smoothing helps unseen words
Unigram Perplexity: 584.6725974941992
Bigram Perplexity: 22.65212085430083
Trigram Perplexity: 17.099759466766965

Sentence: probability estimation is crucial
Unigram Perplexity: 3113.4248600665123
Bigram Perplexity: 32.929491988567
Trigram Perplexity: 13.572088082974531
```

8.Comparison and Analysis (8–10 Sentences)

The trigram model generally produced the lowest perplexity. This indicates better contextual understanding. However, trigrams did not always perform best due to data sparsity. Unseen words caused probability drops without smoothing. Laplace smoothing reduced zero-probability issues. Bigram models balanced context and data availability. Unigrams performed worst due to lack of context. Smoothing improved all models significantly. Higher-order n-grams require larger datasets.

9.Lab Report Structure (2–3 Pages)

1. Objective
    - To build and evaluate N-gram language models.
2. Dataset Description
    - NLP-based corpus with over 1500 words.
3. Preprocessing
    - Cleaning, tokenization, stopword removal.
4. Model Construction
    - Unigram, Bigram, Trigram models.
5. Sentence Probability Results
    - Comparison across models.
6. Perplexity Analysis
    - Lower perplexity for trigrams.
7. Observations & Conclusion
    - Context improves performance but increases sparsity.