

```
# Import libraries
import re                      # for text cleaning (regex)
import nltk                     # for tokenization and stopwords
from collections import Counter, defaultdict # for counting n-grams
import math                     # for probability and perplexity calculations

# Explanation:
# re → remove punctuation/numbers
# nltk → tokenize words, handle stopwords
# collections.Counter → count word frequencies
# collections.defaultdict → store conditional probabilities
# math → log probabilities, perplexity calculations
```

```
# Example dataset: a text corpus (at
with open("dataset.txt", "r") as f:
    text = f.read()
# The file 'dataset.txt' was not fo
# To fix this, we will temporarily u
# Please replace this placeholder wi
# or upload 'dataset.txt' to your Cc

text = """
This is a placeholder text for your
You should replace this with a much
to properly test the language model.
an article, or any other long piece
If you have a 'dataset.txt' file, pl
and then uncomment the original 'wit
```

A language model is a probability distribution over words given a context. Given such a sequence, say of length n , the language model estimates the probability of the next word. Language models are useful for a variety of NLP tasks, such as information retrieval, and many natural language processing systems use them. The simplest type of language model is the unigram model, which assigns the same probability to every word. An n -gram is a contiguous sequence of n words. For example, in the phrase "the quick brown fox", "the quick" and "the quick brown" is a trigram (triple of words).

N-gram models estimate the probability of a word given the previous $n-1$ words. This is done by counting how often each sequence of words appears in the training data. For example, to calculate $P(\text{word}_k | \text{context}_{k-n+1:k-1})$, we count the occurrences of the sequence $\text{context}_{k-n+1:k-1}, \text{word}_k$ and divide it by the count of the sequence $\text{context}_{k-n+1:k-1}$. This approach relies on the Markov assumption, which states that the probability of a word depends only on the previous $n-1$ words.

Smoothing techniques are often used to handle cases where a word has never appeared before. Common smoothing methods include Additive smoothing, Good-Turing smoothing, and Kneser-Ney smoothing. These methods redistribute some probability mass from words that have zero probability to words that have non-zero probability, preventing zero probabilities that would otherwise occur.

```
"""
# If you have 'dataset.txt' uploaded
# with open("dataset.txt", "r") as f:
#     text = f.read()

# Clean unnecessary lines
text = text.strip()
```

```
# Display sample
print(text[:500])
```

This is a placeholder text for your dataset.
 You should replace this with a much larger text corpus, ideally at least 1500 words, to properly test the language model. For example, you can paste text from a book, an article, or any other long piece of writing here.
 If you have a 'dataset.txt' file, please upload it to your Colab environment, and then uncomment the original 'with open("dataset.txt", "r") as f: text = f.read()' line.

A language model is a probability distribution over a sequence o

◆ Gemini

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab') # Add thi

def preprocess(text):
    # Lowercase
    text = text.lower()
    # Remove punctuation and numbers
    text = re.sub(r'^[a-z\s]', ' ', t
    # Tokenize
    tokens = nltk.word_tokenize(text)
    # Optionally remove stopwords
    stopwords = set(nltk.corpus.stopw
    tokens = [t for t in tokens if t
    # Add start/end tokens
    processed = []
    for sentence in nltk.sent_tokenize(
        words = nltk.word_tokenize(s
        processed.append(["<s>"] + w
    return processed

processed_text = preprocess(text)

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
def build_ngram(tokens, n):
    ngrams = []
    for sentence in tokens:
        for i in range(len(sentence)-n+1):
            ngrams.append(tuple(sentence[i:i+n]))
    return Counter(ngrams)

# Unigram, Bigram, Trigram
unigrams = build_ngram(processed_text, 1)
bigrams = build_ngram(processed_text, 2)
trigrams = build_ngram(processed_text, 3)

print("Sample Unigrams:", list(unigrams.items())[:10])
print("Sample Bigrams:", list(bigrams.items())[:10])
```

Sample Unigrams: [((<s>,), 1), (('this',), 4), (('is',), 7), (('a',), 16), (('placeholder',), 1)
 Sample Bigrams: [((<s>, 'this'), 1), (('this', 'is'), 2), (('is', 'a'), 5), (('a', 'placeholder')

```

def laplace_smoothing(count, total, vocab_size):
    return (count + 1) / (total + vocab_size)

def sentence_probability(sentence, model, n, vocab_size):
    tokens = ["<s>"] + nltk.word_tokenize(sentence.lower()) + ["</s>"]
    prob = 1.0
    for i in range(len(tokens)-n+1):
        ngram = tuple(tokens[i:i+n])
        count = model[ngram]
        total = sum(model.values())
        prob *= laplace_smoothing(count, total, vocab_size)
    return prob

sentences = [
    "The cat sat on the mat.",
    "Students are learning language models.",
    "Artificial intelligence is powerful.",
    "Dogs bark loudly at night.",
    "The quick brown fox jumps over the lazy dog."
]

for s in sentences:
    print("Sentence:", s)
    print("Unigram Probability:", sentence_probability(s, unigrams, 1, len(unigrams)))
    print("Bigram Probability:", sentence_probability(s, bigrams, 2, len(bigrams)))
    print("Trigram Probability:", sentence_probability(s, trigrams, 3, len(trigrams)))
    print()

```

Sentence: The cat sat on the mat.
Unigram Probability: 5.944485537594832e-21
Bigram Probability: 2.6875722256150096e-22
Trigram Probability: 3.109854318894012e-20

Sentence: Students are learning language models.
Unigram Probability: 7.925571316962695e-20
Bigram Probability: 7.941775926692354e-20
Trigram Probability: 1.903230843163135e-17

Sentence: Artificial intelligence is powerful.
Unigram Probability: 5.612478651123212e-18
Bigram Probability: 2.346794786337591e-17
Trigram Probability: 1.1647772760158386e-14

Sentence: Dogs bark loudly at night.
Unigram Probability: 2.935396784060257e-21
Bigram Probability: 3.970887963346177e-20
Trigram Probability: 1.903230843163135e-17

Sentence: The quick brown fox jumps over the lazy dog.
Unigram Probability: 6.531483454005439e-28
Bigram Probability: 7.811761079006753e-30
Trigram Probability: 8.140236616337258e-28

```

def perplexity(sentence, model, n, vocab_size):
    tokens = ["<s>"] + nltk.word_tokenize(sentence.lower()) + ["</s>"]
    log_prob = 0
    for i in range(len(tokens)-n+1):
        ngram = tuple(tokens[i:i+n])
        count = model[ngram]
        total = sum(model.values())

```

```
prob = laplace_smoothing(count, total, vocab_size)
log_prob += math.log(prob)
return math.exp(-log_prob/len(tokens))

for s in sentences:
    print("Sentence:", s)
    print("Unigram Perplexity:", perplexity(s, unigrams, 1, len(unigrams)))
    print("Bigram Perplexity:", perplexity(s, bigrams, 2, len(bigrams)))
    print("Trigram Perplexity:", perplexity(s, trigrams, 3, len(trigrams)))
    print()

Sentence: The cat sat on the mat.
Unigram Perplexity: 176.73421536811202
Bigram Perplexity: 249.3088880735985
Trigram Perplexity: 147.05281019460824

Sentence: Students are learning language models.
Unigram Perplexity: 244.13001494881965
Bigram Perplexity: 244.06769312592664
Trigram Perplexity: 123.04488553884742

Sentence: Artificial intelligence is powerful.
Unigram Perplexity: 291.3442633548697
Bigram Perplexity: 237.49059315444097
Trigram Perplexity: 97.8445702538009

Sentence: Dogs bark loudly at night.
Unigram Perplexity: 368.58838727249776
Bigram Perplexity: 266.15770664759424
Trigram Perplexity: 123.04488553884742

Sentence: The quick brown fox jumps over the lazy dog.
Unigram Perplexity: 184.2534721981669
Bigram Perplexity: 266.4429626201017
Trigram Perplexity: 180.90347703814962
```