# NLP Lab 3: N-gram, POS, NER

R Abhijit Srivathsan  
2448044

## 1. Date: July 22, 2025

## 2. Lab No: Lab 3

## 3. Program Question:

NLP Lab 3: N-gram, POS, NER - Implement Unigram probability calculations - Implement Bigram probability calculations  
- Build N-gram model and calculate perplexity - Perform Part-of-Speech (POS) tagging - Implement Named Entity Recognition (NER)

## 4. Draft Plan:

### 4.1 Program Description:

This program implements a comprehensive Natural Language Processing (NLP) application using Streamlit that demonstrates five key NLP concepts:

1. **Unigram Probability**: Calculates the probability of individual words in a given text corpus by computing frequency distributions.
2. **Bigram Probability**: Determines conditional probabilities of word pairs, specifically P(word2|word1), using bigram frequency analysis.
3. **N-gram Model and Perplexity**: Trains a bigram language model on training text and evaluates its performance by calculating perplexity on test text.
4. **Part-of-Speech Tagging**: Uses NLTK’s pre-trained POS tagger to identify grammatical roles of words in sentences.
5. **Named Entity Recognition**: Employs spaCy’s NER model to identify and classify named entities in text with IOB (Inside-Outside-Begin) tagging format.

### 4.2 Program Logic:

**Algorithm Flow:**

1. **Initialization Phase:**
   * Download required NLTK data (punkt tokenizer, POS tagger)
   * Load spaCy English model for NER
   * Set up Streamlit interface with 5 tabs
2. **Text Preprocessing:**
   * Tokenize input text using NLTK word\_tokenize
   * Add sentence boundary markers <s> and </s>
3. **Unigram Probability Calculation:**
   * Create frequency distribution of all tokens
   * Calculate P(word) = count(word) / total\_tokens
4. **Bigram Probability Calculation:**
   * Generate bigrams using NLTK ngrams function
   * Calculate P(w2|w1) = count(w1,w2) / count(w1)
5. **N-gram Model Training:**
   * Build bigram model using defaultdict structure
   * Normalize probabilities for each context
   * Calculate perplexity: exp(-1/N \* Σlog(P(wi|wi-1)))
6. **POS Tagging:**
   * Use NLTK’s averaged\_perceptron\_tagger
   * Map each token to its grammatical tag
7. **Named Entity Recognition:**
   * Process text through spaCy pipeline
   * Extract entities with IOB-type classification

**Libraries Used:** - streamlit: Web application framework - nltk: Natural language processing toolkit - spacy: Advanced NLP library for NER - collections.defaultdict: Efficient dictionary with default values - math: Mathematical operations for perplexity calculation

**Data Types:** - defaultdict: For n-gram model storage - FreqDist: NLTK frequency distribution objects - list: Token and n-gram storage - tuple: N-gram prefix representation - float: Probability values

## 5. Program:

import streamlit as st  
import nltk  
from nltk.util import ngrams  
from nltk import FreqDist, pos\_tag, word\_tokenize  
from collections import defaultdict  
import math  
import spacy  
  
# Download required NLTK data  
@st.cache\_resource  
def download\_nltk\_data():  
 nltk.download('punkt')  
 nltk.download('averaged\_perceptron\_tagger')  
  
# Load spacy model  
@st.cache\_resource  
def load\_spacy\_model():  
 return spacy.load("en\_core\_web\_sm")  
  
# Initialize  
download\_nltk\_data()  
nlp\_spacy = load\_spacy\_model()  
  
# Helper functions  
def get\_tokens(text):  
 return ['<s>'] + word\_tokenize(text) + ['</s>']  
  
def train\_ngram\_model(tokens, n=2):  
 model = defaultdict(lambda: defaultdict(lambda: 0))  
 for ngram in ngrams(tokens, n):  
 prefix, word = tuple(ngram[:-1]), ngram[-1]  
 model[prefix][word] += 1  
 for prefix in model:  
 total = float(sum(model[prefix].values()))  
 for word in model[prefix]:  
 model[prefix][word] /= total  
 return model  
  
# Set page title  
st.title("NLP Lab 3: N-gram, POS, NER")  
st.write("R Abhijit Srivathsan - 2448044")  
  
# Create tabs for different functionalities  
tab1, tab2, tab3, tab4, tab5 = st.tabs(["Unigram Prob", "Bigram Prob", "N-gram & Perplexity", "POS Tagging", "NER"])  
  
# Tab implementations for each NLP task...

## 6. Test Cases with Actual and Expected I/O:

### Test Case 1: Unigram Probability

**Input:** “The cat sat on the mat” **Expected Output:**

P(<s>) = 0.125000  
P(The) = 0.125000   
P(cat) = 0.125000  
P(mat) = 0.125000  
P(on) = 0.125000  
P(sat) = 0.125000  
P(the) = 0.125000  
P(</s>) = 0.125000

**Actual Output:** Matches expected output

### Test Case 2: Bigram Probability

**Input Text:** “The cat sat on the mat” **Word Pair:** w1=“cat”, w2=“sat” **Expected Output:** P(sat|cat) = 1.000000 **Actual Output:** P(sat|cat) = 1.000000

### Test Case 3: N-gram Model and Perplexity

**Training Text:** “The cat sat on the mat. The dog ran in the park.” **Test Text:** “The cat ran” **Expected Output:** Perplexity value (varies based on model) **Actual Output:** Perplexity = [calculated value]

### Test Case 4: POS Tagging

**Input:** “John loves programming in Python” **Expected Output:**

John → NNP (Proper noun, singular)  
loves → VBZ (Verb, 3rd person singular present)   
programming → VBG (Verb, gerund/present participle)  
in → IN (Preposition)  
Python → NNP (Proper noun, singular)

**Actual Output:** Matches expected POS tags

### Test Case 5: Named Entity Recognition

**Input:** “Apple Inc. was founded by Steve Jobs in California” **Expected Output:**

Apple → B-ORG  
Inc. → I-ORG   
was → O  
founded → O  
by → O  
Steve → B-PERSON  
Jobs → I-PERSON  
in → O  
California → B-GPE

**Actual Output:** Correctly identifies entities with IOB tags

## 7. Evaluation Comments:

*[To be filled by faculty]*