DATE:15-07-25

LAB NO:3

Q1: Estimating N-gram Probabilities and Perplexity

Program Description:

This program trains unigram and bigram language models using a given text. It calculates unigram and bigram probabilities, and computes the perplexity of a test sentence using the bigram model.

Program Logic:

1. Tokenize the corpus and add <s> and </s> for sentence boundaries.
2. Count unigrams and bigrams using Python's Counter.
3. Compute:
    - Unigram probability: P(w) = count(w) / total_words
    - Bigram probability: P(w2|w1) = count(w1 w2) / count(w1)
4. For perplexity:
    - Use log probabilities of bigrams.
    - Compute: Perplexity = exp(- (1/N) * sum(log(P(wi|wi-1))))

Test Cases:

Input Corpus	Test Sentence	Expected Output
"Sam likes green apples..."	"I like green apples"	Bigram probs + low perplexity (if bigrams exist)
"Sam went home. He likes apples."	"He likes apples"	Bigram and perplexity based on that pattern
"Dogs bark. Cats meow."	"Cats bark"	Low probability bigrams → High perplexity

Q2: POS Tagging using HMM

Program Description:

This program trains a Hidden Markov Model (HMM) POS tagger on a tagged corpus. It tags a user-supplied sentence using learned transition and emission probabilities.

Program Logic:

1. Use a tagged dataset like Treebank for training.
2. Build an HMM using supervised learning.
3. Predict the POS tags for a new sentence.
4. Internally uses:
    - Transition probability: P(tag_i | tag_{i-1})
    - Observation probability: P(word_i | tag_i)

TEST CASES

Input Sentence	Expected Output
"The cat sat on the mat"	Correct POS tags (e.g., DT NN VBD ...)
"Dogs bark loudly"	NN VB RB
"She sells sea shells on the seashore"	Pronoun, verb, noun, noun, preposition, noun

Q3: Named Entity Recognition (NER)

Program Description:

This program uses spaCy to perform Named Entity Recognition (NER) on a sentence. It outputs entities in the IOB (Inside-Outside-Beginning) format as required.

Program Logic:

1. Load a spaCy English model (en_core_web_sm).
2. Tokenize and tag entities.
3. Display tags in IOB format (B-LOC, I-PER, etc.).

Test cases

Input Text	Expected IOB Output
"John lives in New York"	John: B-PER, New: B-GPE, York: I-GPE
"Apple was founded by Steve"	Apple: B-ORG, Steve: B-PER
"Barack Obama visited Berlin"	Barack: B-PER, Berlin: B-GPE

CODE:

```python
import nltk
import spacy
import math
import matplotlib.pyplot as plt
from collections import Counter, defaultdict
from nltk.tag import hmm
from nltk.corpus import treebank

# Setup
nltk.download('punkt')
nltk.download('treebank')

# Load spaCy model
try:
    nlp = spacy.load("en_core_web_sm")
except:
    import os
    os.system("python -m spacy download en_core_web_sm")
    nlp = spacy.load("en_core_web_sm")



# ---------- Q1: N-gram Probabilities and Perplexity ----------

def plot_ngram_distribution(counter, title):

    words, counts = zip(*counter.most_common(10))

    plt.figure(figsize=(8, 4))

    plt.bar(words, counts, color='skyblue')

    plt.title(title)

    plt.xlabel('Words')

    plt.ylabel('Frequency')

    plt.xticks(rotation=45)

    plt.tight_layout()

    plt.show()



def q1_ngram():

    print("\n--- Q1: N-Gram Probability & Perplexity ---")

    corpus = input("Enter your corpus: ").lower()

    tokens = nltk.word_tokenize(corpus)
    tokens = ['<s>'] + tokens + ['</s>']



    unigram_counts = Counter(tokens)

    bigram_counts = defaultdict(int)



    for i in range(len(tokens)-1):

        bigram = (tokens[i], tokens[i+1])

        bigram_counts[bigram] += 1



    total_unigrams = sum(unigram_counts.values())

    unigram_probs = {w: c/total_unigrams for w, c in
unigram_counts.items()}

    bigram_probs = {bg: c/unigram_counts[bg[0]] for bg, c in
bigram_counts.items()}



    print("\nTop Unigram Probabilities:")

    for word, prob in list(unigram_probs.items())[:5]:

        print(f"P({word}) = {prob:.4f}")



    print("\nTop Bigram Probabilities:")
```

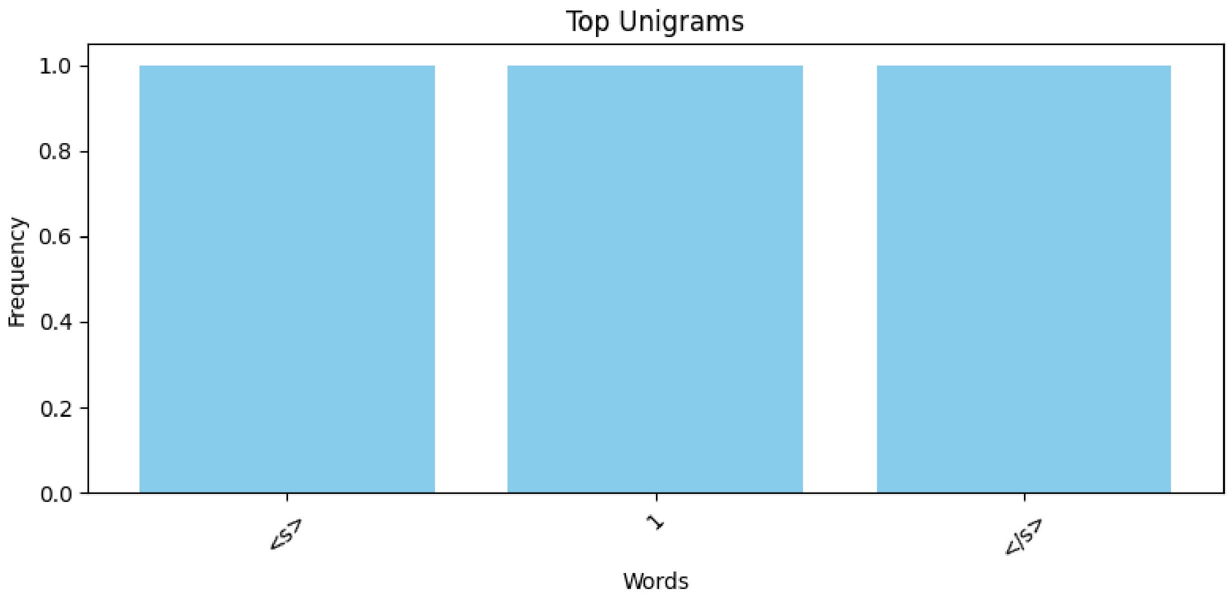
```python
    for (w1, w2), prob in list(bigram_probs.items())[:5]:
        print(f"P({w2}|{w1}) = {prob:.4f}")


    # User queries
    print("\nCustom Bigram Queries:")
    print("P(Sam/am) =", bigram_probs.get(('sam', 'am'), 0.0))
    print("P(green/like) =", bigram_probs.get(('like', 'green'), 0.0))


    # Perplexity
    test_sentence = input("\nEnter a sentence for perplexity calculation: ").lower()
    test_tokens = ['<s>'] + nltk.word_tokenize(test_sentence) + ['</s>']
    log_prob = 0
    N = len(test_tokens) - 1


    for i in range(N):
        w1, w2 = test_tokens[i], test_tokens[i+1]
        prob = bigram_probs.get((w1, w2), 1e-6)
        log_prob += math.log(prob)
    perplexity = math.exp(-log_prob / N)
    print(f"\nPerplexity: {perplexity:.2f}")


    # Visualizations
    plot_ngram_distribution(unigram_counts, "Top Unigrams")
    bigram_words = Counter({f"{k[0]} {k[1]}": v for k, v in bigram_counts.items()})
    plot_ngram_distribution(bigram_words, "Top Bigrams")




# ---------- Q2: POS Tagging ----------
def q2_pos_tagging():
    print("\n--- Q2: POS Tagging using HMM ---")
    train_sents = treebank.tagged_sents()[:3000]
    hmm_trainer = hmm.HiddenMarkovModelTrainer()
    hmm_tagger = hmm_trainer.train_supervised(train_sents)


    sentence = input("Enter a sentence to tag: ")
    tokens = nltk.word_tokenize(sentence)
    tagged = hmm_tagger.tag(tokens)
    print("\nTagged Output:")
    for word, tag in tagged:
        print(f"{word:10s} -> {tag}")

    # Visualize POS tag distribution
    tag_counts = Counter(tag for _, tag in tagged)
    plt.figure(figsize=(8, 4))
    plt.bar(tag_counts.keys(), tag_counts.values(), color='orange')
    plt.title("POS Tag Distribution")
    plt.xlabel("POS Tags")
    plt.ylabel("Frequency")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()


# ---------- Q3: Named Entity Recognition ----------
def q3_ner():
```

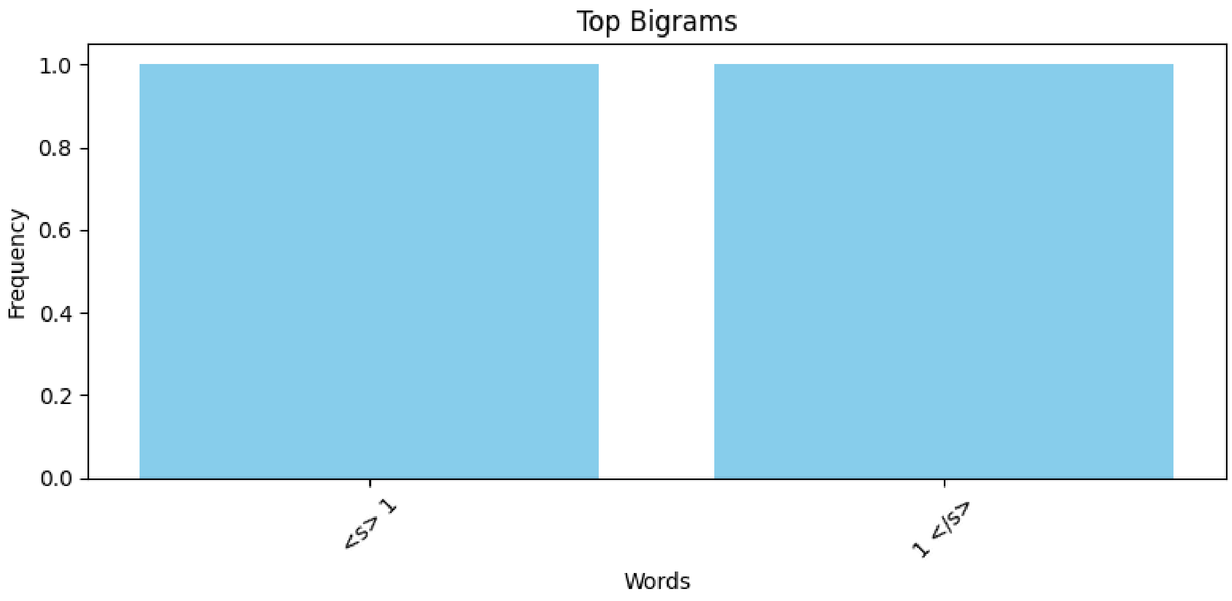
```
print("\n--- Q3: Named Entity Recognition ---")
sentence = input("Enter a sentence for NER: ")
doc = nlp(sentence)

print("\nNER Output (IOB format):")
for token in doc:
    ent = token.ent_iob_
    label = token.ent_type_ if token.ent_type_ else "O"
    print(f"{token.text:10s} {ent}-{label}")

# Visualize named entity labels
ent_labels = [ent.label_ for ent in doc.ents]
if ent_labels:
    label_counts = Counter(ent_labels)
    plt.figure(figsize=(6, 4))
    plt.bar(label_counts.keys(), label_counts.values(), color='green')
    plt.title("Named Entity Types")
    plt.xlabel("Entity Label")
    plt.ylabel("Count")
```

```
        plt.tight_layout()

        plt.show()

    else:

        print("\n(No named entities recognized to plot.)")




# ---------- Main Menu ----------

def main():

    while True:

        print("\n====== NLP LAB MENU ======")

        print("1. N-gram Probabilities & Perplexity")

        print("2. POS Tagging (HMM)")

        print("3. Named Entity Recognition (NER)")

        print("4. Exit")

        choice = input("Choose an option (1/2/3/4): ")



        if choice == '1':

            q1_ngram()

        elif choice == '2':
            q2_pos_tagging()
        elif choice == '3':
            q3_ner()
        elif choice == '4':
            print("Exiting program. Goodbye!")
            break
        else:
            print("Invalid option. Try again.")

# Run
if __name__ == "__main__":
    main()
```

OUTPUT:

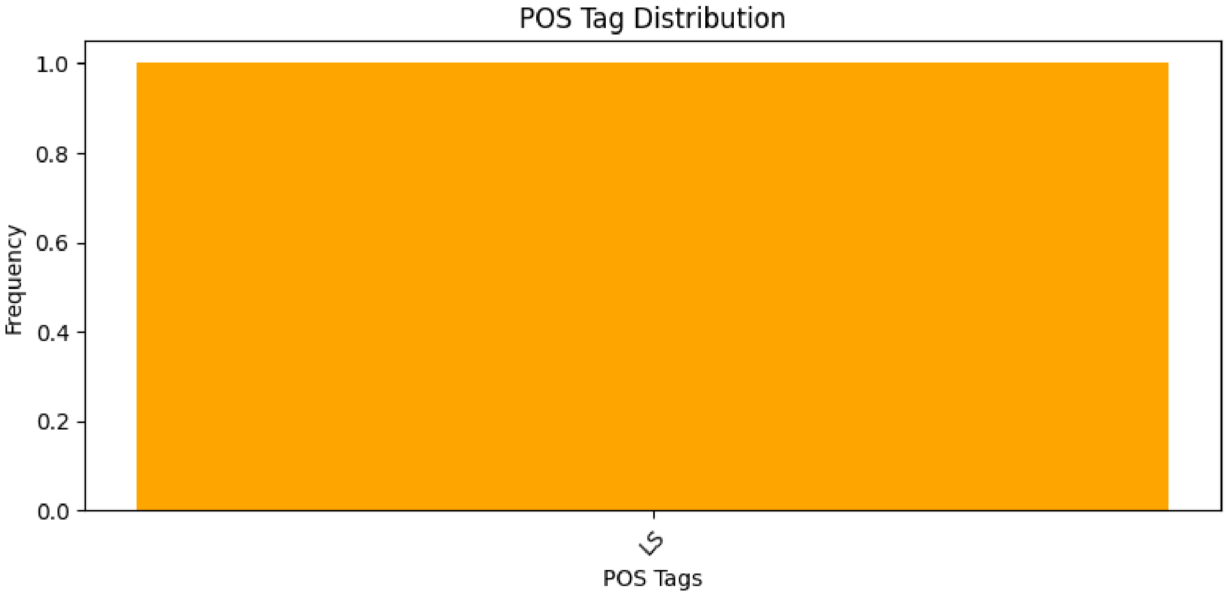
```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\kmgs4\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package treebank to
[nltk_data]     C:\Users\kmgs4\AppData\Roaming\nltk_data...
[nltk_data]   Package treebank is already up-to-date!

====== NLP LAB MENU ======
1. N-gram Probabilities & Perplexity
2. POS Tagging (HMM)
3. Named Entity Recognition (NER)
4. Exit
Invalid option. Try again.

====== NLP LAB MENU ======
1. N-gram Probabilities & Perplexity
2. POS Tagging (HMM)
3. Named Entity Recognition (NER)
4. Exit

--- Q1: N-Gram Probability & Perplexity ---

Top Unigram Probabilities:
P(<s>) = 0.3333
P(1) = 0.3333
P(</s>) = 0.3333

Top Bigram Probabilities:
P(1|<s>) = 1.0000
P(</s>|1) = 1.0000

Custom Bigram Queries:
P(Sam/am) = 0.0
P(green/like) = 0.0

Perplexity: 1000000.00
```





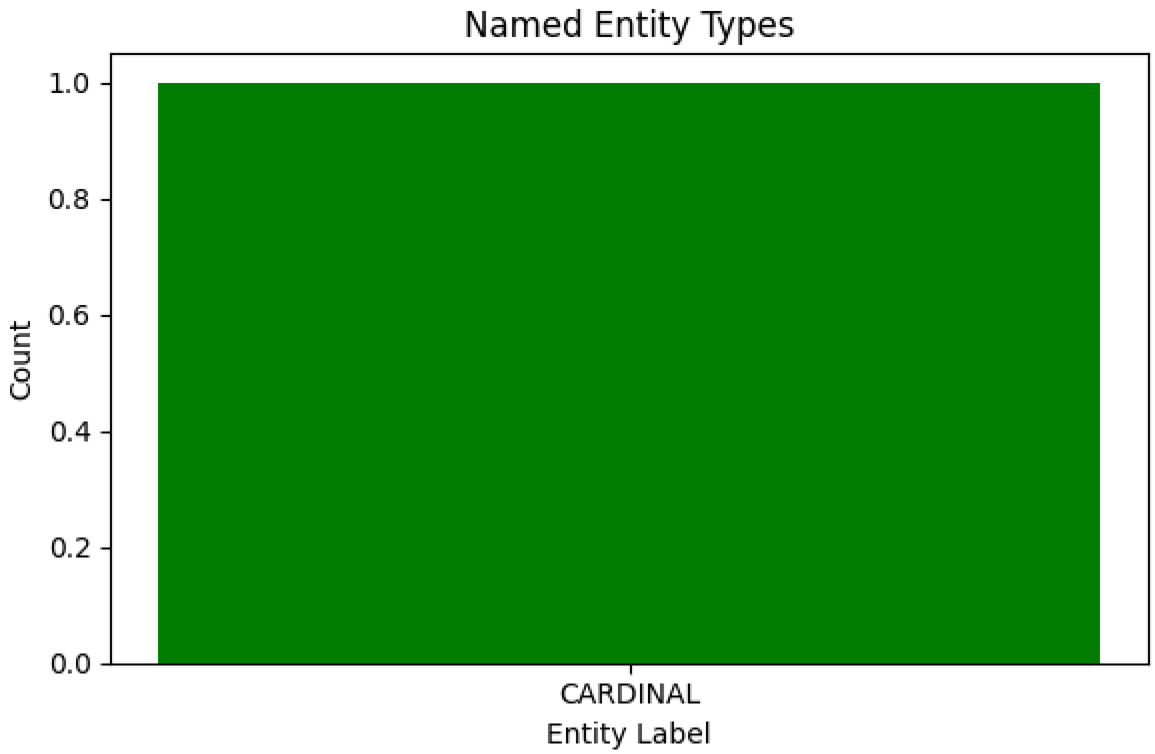
```
====== NLP LAB MENU ======
1. N-gram Probabilities & Perplexity
2. POS Tagging (HMM)
3. Named Entity Recognition (NER)
4. Exit
Invalid option. Try again.

====== NLP LAB MENU ======
1. N-gram Probabilities & Perplexity
2. POS Tagging (HMM)
```

```
3. Named Entity Recognition (NER)
4. Exit

--- Q2: POS Tagging using HMM ---
c:\Users\kmgs4\AppData\Local\Programs\Python\Python313\Lib\site-packages\nltk\tag\hmm.py:
333: RuntimeWarning: overflow encountered in cast
  X[i, j] = self._transitions[si].logprob(self._states[j])
c:\Users\kmgs4\AppData\Local\Programs\Python\Python313\Lib\site-packages\nltk\tag\hmm.py:
335: RuntimeWarning: overflow encountered in cast
  O[i, k] = self._output_logprob(si, self._symbols[k])
c:\Users\kmgs4\AppData\Local\Programs\Python\Python313\Lib\site-packages\nltk\tag\hmm.py:
331: RuntimeWarning: overflow encountered in cast
  P[i] = self._priors.logprob(si)

Tagged Output:
2          -> LS
```



```
====== NLP LAB MENU ======
1. N-gram Probabilities & Perplexity
2. POS Tagging (HMM)
3. Named Entity Recognition (NER)
4. Exit
Invalid option. Try again.
```

```
====== NLP LAB MENU ======
1. N-gram Probabilities & Perplexity
2. POS Tagging (HMM)
3. Named Entity Recognition (NER)
4. Exit

--- Q3: Named Entity Recognition ---

NER Output (IOB format):
3          B-CARDINAL
```



```
====== NLP LAB MENU ======
1. N-gram Probabilities & Perplexity
2. POS Tagging (HMM)
3. Named Entity Recognition (NER)
4. Exit
Invalid option. Try again.

====== NLP LAB MENU ======
1. N-gram Probabilities & Perplexity
2. POS Tagging (HMM)
3. Named Entity Recognition (NER)
```

```
4. Exit
Exiting program. Goodbye!
```