EX.NO.: 10

DATE: 27.01.2025

PROBABILITY N-GRAM

To implement bigram modeling on a given corpus (Berkeley restaurant project corpus file) and perform the following tasks:

1. Extract text from the corpus file.
2. Generate all possible bigrams from the text.
3. Apply **Add-One Smoothing** and calculate the probabilities of all bigrams.
4. Query and print the probability of a given random bigram.
5. Implement and apply **Kneser-Ney Smoothing** for the bigram model.
6. Display the probability matrices for both smoothing methods.

PROCEDURE:

1. Read the Text Corpus:
   a. Use the read_file() function to load the corpus from the specified text file and convert it to lowercase for consistency.
2. Tokenize the Text:
   a. Use NLTK's word_tokenize() function to split the text into individual tokens (words).
3. Generate Unigrams and Bigrams:
   a. Extract unigrams (single tokens) directly from the tokenized text.
   b. Use the ngrams() function from NLTK to generate all bigrams from the tokenized text.
4. Calculate Unigram and Bigram Counts:
   a. Count the occurrences of each unigram using Counter.
   b. Similarly, count the occurrences of each bigram using Counter.
5. Add-One Smoothing:
   a. Implement Add-One smoothing to adjust the probabilities of bigrams, avoiding zero probabilities for unseen bigrams.
   b. Use the formula: P(w2∣w1)=C(w1, w2)+1C(w1)+∣V∣P(w_2|w_1) = \frac{\text{C(w1, w2)} + 1}{\text{C(w1)} + |\text{V}|}P(w2∣w1)=C(w1)+∣V∣C(w1, w2)+1 where ∣V∣|\text{V}|∣V∣ is the vocabulary size.
6. Generate Probability Matrix for Add-One Smoothing:
   a. Use the generate_bigram_matrix() function to construct a matrix where rows and columns represent vocabulary words, and each cell shows the smoothed probability of the bigram.
7. Kneser-Ney Smoothing:
   a. Implement Kneser-Ney smoothing, which redistributes probability mass to account for lower-order n-gram probabilities.
   b. Use the formula: PKN(w2∣w1)=max(C(w1, w2)−d,0)/C(w1)+d⋅∣Continuations(w2)∣∣Vocabulary∣P_{\text{KN}}(w_2|w_1) = \max(\text{C(w1, w2)} - d, 0) / \text{C(w1)} + d \cdot \frac{|\text{Continuations}(w_2)|}{|\text{Vocabulary}|}PKN(w2∣w1)=max(C(w1, w2)−d,0)/C(w1)+d⋅∣Vocabulary∣∣Continuations(w2)∣ where ddd is the discount factor.
8. Generate Probability Matrix for Kneser-Ney Smoothing:
   a. Use the generate_bigram_matrix() function to display the Kneser-Ney smoothed bigram probabilities in a matrix format.
9. Query for a Random Bigram:
   a. Accept or define a random bigram input (e.g., ('restaurant', 'berkeley')).
   b. Retrieve and print the probability of the bigram for both Add-One and Kneser-Ney smoothing methods.
10. Output Results:
    a. Display the bigram probability matrices for both Add-One and Kneser-Ney smoothing.
    b. Print the queried bigram probabilities with clear labels.

CODE AND OUTPUT

```
import nltk
from nltk.util import ngrams
```

```python
from collections import Counter
import pandas as pd

# Function to read text from a file
def read_file(file_path):
    with open(file_path, 'r') as file:
        return file.read().lower()

# Function to calculate bigram probabilities with add-one smoothing
def add_one_smoothing(bigrams, unigram_counts, vocabulary_size):
    bigram_counts = Counter(bigrams)
    smoothed_probabilities = {}

    for bigram in bigram_counts:
        smoothed_probabilities[bigram] = (bigram_counts[bigram] + 1) /
(unigram_counts[bigram[0]] + vocabulary_size)

    return smoothed_probabilities

# Generate a smoothed count matrix
def generate_bigram_count_matrix(tokens, bigrams, vocabulary_size):
    vocab = sorted(set(tokens))
    unigram_counts = Counter(tokens)
    bigram_counts = Counter(bigrams)

    bigram_count_matrix = pd.DataFrame(0, index=vocab, columns=vocab, dtype=int)

    for w1 in vocab:
        for w2 in vocab:
            bigram_count_matrix.loc[w1, w2] = bigram_counts[(w1, w2)] + 1  # Add-One
Smoothing

    return bigram_count_matrix

# Generate a bigram probability matrix
def generate_bigram_probability_matrix(tokens, smoothed_probabilities):
    vocab = sorted(set(tokens))
    bigram_matrix = pd.DataFrame(0, index=vocab, columns=vocab, dtype=float)

    for (w1, w2), prob in smoothed_probabilities.items():
        bigram_matrix.loc[w1, w2] = prob

    return bigram_matrix

# Main function
def main():
    nltk.download('punkt')

    # Read text from file
```

```
    file_path = 'corpus.txt'  # Replace with your file path
    text = read_file(file_path)

    # Tokenize the text
    tokens = nltk.word_tokenize(text)

    # Generate unigrams and bigrams
    unigrams = tokens
    bigrams = list(ngrams(tokens, 2))

    # Calculate unigram counts
    unigram_counts = Counter(unigrams)
    vocabulary_size = len(unigram_counts)

    # Apply Add-One Smoothing
    add_one_probs = add_one_smoothing(bigrams, unigram_counts, vocabulary_size)

    # Generate bigram count matrix
    bigram_count_matrix = generate_bigram_count_matrix(tokens, bigrams,
vocabulary_size)

    # Generate bigram probability matrix
    bigram_probability_matrix = generate_bigram_probability_matrix(tokens,
add_one_probs)

    # Print Bigram Probability Matrix
    print("\nBigram Probability Matrix with Add-One Smoothing:")
    print(bigram_probability_matrix.round(4))

    # Print Bigram Count Matrix
    print("Bigram Count Matrix with Add-One Smoothing:")
    print(bigram_count_matrix)

    # Prompt the user for a bigram input
    def get_bigram_probability(smoothed_probabilities, bigram):
        return smoothed_probabilities.get(bigram, "Bigram not found")

    # Example usage for querying a bigram
    random_bigram = ('spend', 'money')  # example input bigram
    probability = get_bigram_probability(add_one_probs, random_bigram)
    print(f"Probability of the bigram {random_bigram}: {probability}")

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

```
Bigram Probability Matrix with Add-One Smoothing:
           chinese  delicious     eat  favorite    food     for       i  \
chinese     0.0000     0.0000  0.0000    0.0000  0.2963  0.0000  0.0000
delicious   0.1053     0.0000  0.0000    0.0000  0.1053  0.0000  0.0000
eat         0.2308     0.0000  0.0000    0.0000  0.0769  0.0000  0.0000
favorite    0.0000     0.0000  0.0000    0.0000  0.0000  0.0000  0.0000
food        0.0000     0.0000  0.1071    0.0000  0.0000  0.1071  0.0714
for         0.0000     0.0000  0.0000    0.0000  0.0000  0.0000  0.0000
i           0.0000     0.0000  0.0000    0.0000  0.0000  0.0000  0.0000
is          0.0000     0.1429  0.0000    0.0000  0.0000  0.0000  0.0000
lunch       0.1111     0.0000  0.0741    0.0000  0.0000  0.0000  0.1111
meal        0.0000     0.0000  0.0000    0.0000  0.0000  0.0000  0.0000
money       0.0000     0.0000  0.0000    0.0000  0.0000  0.0000  0.0909
my          0.0000     0.0000  0.0000    0.1111  0.0000  0.0000  0.0000
on          0.1000     0.0000  0.0000    0.0000  0.1000  0.0000  0.0000
spend       0.0000     0.0000  0.0000    0.0000  0.0000  0.0000  0.0000
tasty       0.0000     0.0000  0.0000    0.0000  0.1111  0.0000  0.0000
to          0.0000     0.0000  0.2800    0.0000  0.0000  0.0000  0.0000
want        0.0909     0.0000  0.0000    0.0000  0.0000  0.0000  0.0000

               is   lunch    meal   money      my      on   spend   tasty  \
chinese    0.0000  0.1481  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
delicious  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
eat        0.0000  0.1538  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
...
tasty           1   1   1       1      1   1     1
to              1   1   1       3      1   1     1
want            1   1   1       1      1   5     1
Probability of the bigram ('spend', 'money'): 0.2727272727272727
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

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