Experiment 3

Name: Pranay Singhvi

UID:2021300126

Aim: Calculate bigrams from a given corpus , display bigram probability table and calculate probability of a sentence.

To apply add-one smoothing on sparse bigram table

Objective

1. Enter the corpus
2. Perform preprocessing. (Covert data to lowercase, and remove punctuation marks and stop words to remove noise.

use the eos tag to mark the beginning and end of the sentence.)

3. Tokenize the data.
4. Generate bigrams.
5. Calculate frequency of bigrams.
6. Generate bi-gram probability table.
7. Input text to calculate probability.
8. Repeat previous steps of pre-process, tokenize, generate bi-gram.
9. Caculate probability of word sequence ie. given sentence using bigram probability generated from the input corpus.

Theory

The experiment involves analyzing a corpus using natural language processing (NLP) techniques to generate bigrams and calculate their frequencies. Here's a detailed theoretical explanation:

1. **Corpus and Tokenization:** The corpus, a collection of text, is tokenized into individual words using NLTK's `word_tokenize` function. This step transforms the raw text into a format suitable for further analysis.
2. **Bigram Generation:** Bigrams are generated from the tokenized words using NLTK's `bigrams` function. A bigram is a pair of consecutive words that appear together in the text. This step establishes relationships between adjacent words.
3. **Frequency Calculation:** The `Counter` class is employed to calculate the frequency of each bigram in the corpus. This involves counting the occurrences of each unique bigram in the list of generated bigrams.
4. **Analysis and Insights:** The resulting list of bigrams and their frequencies provides insights into which word pairs commonly co-occur in the given text. This information is valuable for tasks such as language modeling, understanding contextual relationships, and improving predictive models.
5. **Importance of Tokenization:** Tokenization is a crucial pre-processing step as it breaks down the text into meaningful units (words). This allows for a more granular analysis of language structure and relationships.
6. **NLTK Library:** The experiment utilizes the Natural Language Toolkit (NLTK), a powerful library for NLP in Python. NLTK provides tools for tasks such as tokenization, bigram generation, and frequency counting, streamlining the NLP workflow.
7. **Application:** The generated bigrams and their frequencies can be applied in various NLP applications, including language modeling, sentiment analysis, and information retrieval. Understanding common word associations aids in developing more accurate and context-aware models.

Import Libraries

```
import pandas as pd
import nltk, re, pprint, string
from nltk import word_tokenize, sent_tokenize
from prettytable import PrettyTable
from nltk import FreqDist, bigrams
from collections import Counter
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Perform preprocessing. (Covert data to lowercase, and remove punctuation marks and stop words to remove noise.use the eos tag to mark the beginning and end of the sentence.)

```
corpus = "You book a flight. I read a book. You read!"
corpus = "eos " + corpus

corpus = corpus.lower()
corpus = corpus.replace(",", "").replace(".", " eos").replace("!", " eos").replace("?", " eos").replace(":", "").replace(";", "").replace("
print(corpus)
```

```
eos you book a flight eos i read a book eos you read eos
```

Tokenize the data.

```
tokens = word_tokenize(corpus)
print(tokens)
unique_words = [""] + tokens
unique_words = pd.unique(unique_words)
print(unique_words)
```

```
['eos', 'you', 'book', 'a', 'flight', 'eos', 'i', 'read', 'a', 'book', 'eos', 'you', 'read', 'eos']
['' 'eos' 'you' 'book' 'a' 'flight' 'i' 'read']
```

Frequency of Word

```
word_freq = FreqDist(tokens)
feq_table = PrettyTable()
feq_table.field_names = ["Word","Frequency"]
for word in unique_words:
  if word == 'eos' or word == '':
    continue
  feq_table.add_row([word,word_freq[word]])

print("Frequency Table")
print(feq_table)
```

```
Frequency Table
+--------+-----------+
|  Word  | Frequency |
+--------+-----------+
|  you   |     2     |
|  book  |     2     |
|   a    |     2     |
| flight |     1     |
|   i    |     1     |
|  read  |     2     |
+--------+-----------+
```

Generate bigrams

```
def generate_bigrams(corpus):
    tokens = word_tokenize(corpus.lower())
    bigram_list = list(bigrams(tokens))
    return bigram_list
```

Calculate frequency of bigrams.

```
def calculate_bigram_frequencies(bigram_list):
    bigram_counts = Counter(bigram_list)
    return bigram_counts

all_bigrams = generate_bigrams(corpus)
bigram_frequencies = calculate_bigram_frequencies(all_bigrams)
print(all_bigrams)
bi_feq_table = PrettyTable()
bi_feq_table.field_names = ["Bigram","Frequency"]
unique_bigrams = set(all_bigrams)
for bigram in unique_bigrams:
  bi_feq_table.add_row([bigram,bigram_frequencies[bigram]])

print("Bigram Frequency Table")
print(bi_feq_table)
```

```
[('eos', 'you'), ('you', 'book'), ('book', 'a'), ('a', 'flight'), ('flight', 'eos'), ('eos', 'i'), ('i', 'read'), ('read', 'a'), ('a'
Bigram Frequency Table
+-------------------+-----------+
|       Bigram      | Frequency |
+-------------------+-----------+
|   ('a', 'book')   |     1     |
|    ('eos', 'i')   |     1     |
|  ('you', 'book')  |     1     |
|  ('you', 'read')  |     1     |
|  ('a', 'flight')  |     1     |
|  ('book', 'eos')  |     1     |
|   ('i', 'read')   |     1     |
|   ('eos', 'you')  |     2     |
|  ('read', 'eos')  |     1     |
|   ('book', 'a')   |     1     |
| ('flight', 'eos') |     1     |
|   ('read', 'a')   |     1     |
+-------------------+-----------+
```

Generate bi-gram probability table.

```
def calculate_bigram_probabilities(bigram_list):
    unique_bigrams = set(bigram_list)
    total_bigrams = len(bigram_list)
    bigram_probabilities = {bigram: (bigram_frequencies[bigram]) / (word_freq[bigram[0]])
                            for bigram in unique_bigrams}
    return bigram_probabilities

print("Bigram Probability Table:")
bigram_probabilities = calculate_bigram_probabilities(unique_bigrams)
for bigram, prob in bigram_probabilities.items():
    print(f"{bigram}: {prob:.4f}")
```

```
Bigram Probability Table:
('a', 'book'): 0.5000
('eos', 'i'): 0.2500
('you', 'book'): 0.5000
('you', 'read'): 0.5000
('a', 'flight'): 0.5000
('book', 'eos'): 0.5000
('i', 'read'): 1.0000
('eos', 'you'): 0.5000
('read', 'eos'): 0.5000
('book', 'a'): 0.5000
('flight', 'eos'): 1.0000
('read', 'a'): 0.5000
```

```
table = PrettyTable()
unique_words=[""]+tokens
unique_words=pd.unique(unique_words)
print(unique_words)
h = {}
for token in tokens:
  h[token] = tokens.count(token)
print(h)
table.field_names=[i for i in unique_words]
unique_words=unique_words[1:]
for i in unique_words:
  row=[i]
  for j in unique_words:
    count=0
    for k in range(len(tokens)-1):
      if tokens[k]==i and tokens[k+1]==j:
        count+=1
    row.append(count/h[i])
  table.add_row(row)
print(table)
```

```
['' 'eos' 'you' 'book' 'a' 'flight' 'i' 'read']
{'eos': 4, 'you': 2, 'book': 2, 'a': 2, 'flight': 1, 'i': 1, 'read': 2}
+--------+-----+-----+------+-----+--------+------+------+
|        | eos | you | book |  a  | flight |  i   | read |
+--------+-----+-----+------+-----+--------+------+------+
|  eos   | 0.0 | 0.5 | 0.0  | 0.0 |  0.0   | 0.25 | 0.0  |
|  you   | 0.0 | 0.0 | 0.5  | 0.0 |  0.0   | 0.0  | 0.5  |
|  book  | 0.5 | 0.0 | 0.0  | 0.5 |  0.0   | 0.0  | 0.0  |
|   a    | 0.0 | 0.0 | 0.5  | 0.0 |  0.5   | 0.0  | 0.0  |
| flight | 1.0 | 0.0 | 0.0  | 0.0 |  0.0   | 0.0  | 0.0  |
|   i    | 0.0 | 0.0 | 0.0  | 0.0 |  0.0   | 0.0  | 1.0  |
|  read  | 0.5 | 0.0 | 0.0  | 0.5 |  0.0   | 0.0  | 0.0  |
+--------+-----+-----+------+-----+--------+------+------+
```

```
table = PrettyTable()
unique_words=[""]+tokens
unique_words=pd.unique(unique_words)
print(unique_words)
h = {}
for token in tokens:
  h[token] = tokens.count(token)
print(h)
table.field_names=[i for i in unique_words]
unique_words=unique_words[1:]
for i in unique_words:
  row=[i]
  for j in unique_words:
    count=0
    for k in range(len(tokens)-1):
      if tokens[k]==i and tokens[k+1]==j:
        count+=1
    row.append(round((count+1)/(h[i] + len(unique_words)),2))
  table.add_row(row)
print(table)
```

```
['' 'eos' 'you' 'book' 'a' 'flight' 'i' 'read']
{'eos': 4, 'you': 2, 'book': 2, 'a': 2, 'flight': 1, 'i': 1, 'read': 2}
+--------+------+------+------+------+--------+------+------+
|        | eos  | you  | book |  a   | flight |  i   | read |
+--------+------+------+------+------+--------+------+------+
|  eos   | 0.09 | 0.27 | 0.09 | 0.09 |  0.09  | 0.18 | 0.09 |
|  you   | 0.11 | 0.11 | 0.22 | 0.11 |  0.11  | 0.11 | 0.22 |
|  book  | 0.22 | 0.11 | 0.11 | 0.22 |  0.11  | 0.11 | 0.11 |
|   a    | 0.11 | 0.11 | 0.22 | 0.11 |  0.22  | 0.11 | 0.11 |
| flight | 0.25 | 0.12 | 0.12 | 0.12 |  0.12  | 0.12 | 0.12 |
|   i    | 0.12 | 0.12 | 0.12 | 0.12 |  0.12  | 0.12 | 0.25 |
|  read  | 0.22 | 0.11 | 0.11 | 0.22 |  0.11  | 0.11 | 0.11 |
+--------+------+------+------+------+--------+------+------+
```

```
def N_Grams_smoothing(bigram_list):
    bigram_freq = FreqDist(bigram_list)
    # Add-one smoothing
    unique_bigrams = set(bigram_list)
    total_bigrams = len(bigram_list)
    bigram_probabilities = {bigram: (bigram_freq[bigram] + 1) / (total_bigrams + len(unique_bigrams))
                           for bigram in unique_bigrams}

    return bigram_probabilities


def calculate_sentence_probability(sentence, bigram_probabilities):
    probability = 1.0
    for bigram in sentence:
        if bigram_frequencies[bigram] == 0:
          bigram_probabilities = N_Grams_smoothing(list(unique_bigrams) +sentence)

        probability *= bigram_probabilities[bigram]

    return probability
```

Input text to calculate probability.

```
test_sentence = "You book a flight."
test_sentence = test_sentence.lower()

test_sentence_pro = test_sentence.replace(",", "").replace(".", " eos").replace("!", " eos").replace("?", " eos").replace(":", "").replace
test_sentence_pro = "eos " + test_sentence_pro
tokenized_test_sentence = word_tokenize(test_sentence_pro)
test_sentence_bigrams = list(bigrams(tokenized_test_sentence))

# Calculate probability of the test sentence
probability = calculate_sentence_probability(test_sentence_bigrams, bigram_probabilities)

# Display calculated probability for the test sentence
```

```
print(f"\nThe probability of the sentence '{test_sentence}' is: {probability:.6f}")
```

```
The probability of the sentence 'you book a flight.' is: 0.062500
```

```
test_sentence = "Are you in hospital."
test_sentence = test_sentence.lower()

test_sentence_pro = test_sentence.replace(",", "").replace(".", " eos").replace("!", " eos").replace("?", " eos").replace(":", "").replace
test_sentence_pro = "eos " + test_sentence_pro
tokenized_test_sentence = word_tokenize(test_sentence_pro)
test_sentence_bigrams = list(bigrams(tokenized_test_sentence))

# Calculate probability of the test sentence
probability = calculate_sentence_probability(test_sentence_bigrams, bigram_probabilities)

# Display calculated probability for the test sentence
print(f"\nThe probability of the sentence '{test_sentence}' is: {probability:.6f}")
```

```
The probability of the sentence 'are you in hospital.' is: 0.000001
```

Conclusion

In this experiment, I analyzed a corpus to generate bigrams and calculated their frequencies. Utilizing NLTK, the code successfully processed a string input, tokenized it into words, and produced a list of bigrams with their respective frequencies. The experiment demonstrates an effective approach to basic natural language processing tasks.