Name	Hatim Sawai
UID No.	2021300108
Experiment No.	3

Experiment 3

Aim	1. Calculate bigrams from a given corpus , display bigram probability table and calculate probability of a sentence. 2. To apply add-one smoothing on sparse bigram table

1. Installation of NLTK and downloading the required corpus

In [ ]:
```
import nltk
import pandas as pd
import matplotlib.pyplot as plt
from nltk.corpus import wordnet
from prettytable import PrettyTable
from nltk.tokenize import word_tokenize
from pattern.text.en import pluralize, conjugate, comparative
from nltk.stem import WordNetLemmatizer, PorterStemmer, LancasterStemmer, SnowballS
```

```
C:\Users\hatim\AppData\Local\Temp\ipykernel_1572\1792132151.py:2: DeprecationWarnin
g:
Pyarrow will become a required dependency of pandas in the next major release of pan
das (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better inte
roperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
nltk.download("punkt")
nltk.download("averaged_perceptron_tagger")
nltk.download("wordnet")
```

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

```
Out[ ]: True
```

2. Preprocessing of the given corpus

```
In [ ]: def preprocess(data):
            data = data.lower()
            data = data.replace(",", "").replace(".", " eos").replace("!", " eos").replace(
            data = data.replace("'s", "")
            data = "eos " + data
            tokens = word_tokenize(data)
            return tokens
```

```
In [ ]: # Read the data from input.txt
        with open("input.txt", "r") as file:
            corpus = file.read()

        # Preprocess the data
        tokens = preprocess(corpus)
        print(tokens)
```

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

3. Making Bigram Probability Table

In [ ]:
```
# create bigram table using pretty table
unique = [""] + tokens
h = {}
for i in unique:
    h[i] = tokens.count(i)
unique = pd.unique(unique)
bigram_table = PrettyTable()
bigram_table.field_names = [i for i in unique]
unique = unique[1:]
for i in unique:
    row = [i]
    for j in unique:
        count = 0
        for k in range(len(tokens) - 1):
            if tokens[k] == i and tokens[k + 1] == j:
                count += 1
        p = count/h[i]
        row.append(p)
    bigram_table.add_row(row)

print(bigram_table)
```

```
+--------+-----+-----+------+-----+--------+------+------+
|        | 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  |
+--------+-----+-----+------+-----+--------+------+------+
```

4. Making Add-One Smoothed Bigram Table

In [ ]:
```
# create add one smoothing bigram table using pretty table
unique = [""] + tokens
unique = pd.unique(unique)
smooth_table = PrettyTable()
smooth_table.field_names = [i for i in unique]
unique = unique[1:]
for i in unique:
    row = [i]
    for j in unique:
        count = 0
        for k in range(len(tokens) - 1):
            if tokens[k] == i and tokens[k + 1] == j:
                count += 1
        p = (count+1)/(h[i]+len(unique))
        p = round(p, 2)
        row.append(p)
    smooth_table.add_row(row)

print(smooth_table)
```

```
+--------+------+------+------+------+--------+------+------+
|        | 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 |
+--------+------+------+------+------+--------+------+------+
```

5. Calculating Probability of a Sentence

In [ ]:
```
valid = "I book a flight."
invalid = "A flight book I."
```

In [ ]:
```
tokensa = preprocess(valid)
tokensb = preprocess(invalid)
print(tokensa)
print(tokensb)
```

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

In [ ]:
```
proba, probb = 1, 1
for i in range(len(tokensa) - 1):
    proba *= smooth_table.rows[smooth_table.field_names.index(tokensa[i])-1][smooth

for i in range(len(tokensb) - 1):
    probb *= smooth_table.rows[smooth_table.field_names.index(tokensb[i])-1][smooth

print("Probabilty of sentence A:", proba)
print("Probabilty of sentence B:", probb)
```

```
Probabilty of sentence A: 0.00026136
Probabilty of sentence B: 3.13632e-05
```

In [ ]:
```
if proba > probb:
    print("Sentence A is more probable")
else:
    print("Sentence B is more probable")
```

```
Sentence A is more probable
```

6. Curiosity Questions

Q1. Some Terminologies

1. **Corpus**: A corpus is a collection of written texts and is used for language research and development.
2. **Bigram**: A bigram is a sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words. A bigram is an n-gram for n=2.
3. **Add-one Smoothing**: Add-one smoothing is a simple method to smooth zero probabilities in a probability distribution. It is also known as Laplace smoothing.
4. **Sparse Table**: A sparse table is a table that has a large number of cells with zero values. It is a table in which most of the entries are zero.
5. **Word Form:** the inflected form as it actually appears in the corpus.
6. **Lemma:** an abstract form, shared by word forms having the same stem, part of speech, and word sense – stands for the class of words with stem.
7. **Types:** number of distinct words in a corpus (vocabulary size).
8. **Tokens:** total number of words in a corpus.

Q2. What is Bi-gram Model? How to calculate the probability of a sentence?

Ans: A bigram model is a type of n-gram language model where "n" equals 2. In simpler terms, it predicts the next word in a sentence based solely on the word that came before it. This assumption, termed the Markov assumption, states that the probability of a word depends only on the previous word, ignoring all preceding context. While simplistic, bigram models hold historical significance as the foundation for more complex language models.
Calculating Sentence Probability:
Here's how to calculate the probability of a sentence in a bigram model:

1. Tokenize the sentence: Divide the sentence into individual words (tokens).
2. Estimate bigram probabilities: Calculate the probability of each word following the previous word in the training corpus. This uses the formula: **P(w_i | w_(i-1)) = Count(w_(i-1), w_i) / Count(w_(i-1))**

where:
P(w_i | w_(i-1)) is the probability of word w_i given the previous word w_(i-1).
Count(w_(i-1), w_i) is the number of times the bigram (w_(i-1), w_i) appears in the training corpus.
Count(w_(i-1)) is the number of times the word w_(i-1) appears in the training corpus.
Multiply probabilities: Multiply the individual bigram probabilities for each word pair in the sentence.

3. Smoothing: Since most word pairs might not be present in the corpus, apply smoothing techniques like Laplace smoothing or Witten-Bell discounting to adjust probabilities and avoid zero probabilities.

6. Conclusion

In this experiment we learned about bigrams and add-one smoothing. We calculated bigrams from a given corpus and displayed the bigram probability table. We also calculated the probability of a sentence using bigram probabilities. We also applied add-one smoothing on the sparse bigram table.