

ميني پيکره فرضي (متن ورودي برنامه):

Language processing, also known as natural language processing (NLP), is a field of artificial intelligence that focuses on the interaction between computers and human language. It involves the development of algorithms and models to enable computers to understand, interpret, and generate human language. NLP encompasses a wide range of tasks, including speech recognition, sentiment analysis, machine translation, information extraction, and text generation. By leveraging techniques from linguistics, computer science, and statistics, language processing aims to bridge the gap between human communication and machine understanding.

برای محاسبه تعداد unigram و bigram در متن ورودی، مراحل زیر را انجام می دهیم:

- متن را به کلمات فردی (unigram) با حذف علامت‌های نگارشی و فاصله‌ها تقسیم می کنیم.
- تعداد تکرار هر کلمه یکتا را می شماریم تا تعداد unigram را تعیین کنیم.
- تمام جفت‌های ممکن از کلمات متوالی (bigram) در متن را تولید می کنیم.
- تعداد تکرار هر bigram یکتا را می شماریم تا تعداد bigram را تعیین کنیم.

برای ساختن یک رشته تصادفی از کلمات و محاسبه احتمال وقوع این رشته کلمات، با فرض unigram و bigram موجود، می‌توانید مراحل زیر را دنبال می کنیم:

- به صورت تصادفی یک کلمه از لیست کلمات موجود در متن انتخاب می کنیم (unigram).
- این کلمه را به عنوان نقطه شروع استفاده می کنیم.
- به صورت تصادفی کلمه بعدی را بر اساس توزیع احتمال bigram که با کلمه قبلی انتخاب شده شروع می‌شوند، انتخاب می کنیم.
- دو مرحله قبل را تا رسیدن به طول مورد نظر برای رشته تکرار می کنیم.
- احتمال رشته تولید شده را با ضرب احتمالات bigram های انتخاب شده محاسبه می کنیم.

برای محاسبه احتمال وقوع رشته تصادفی با استفاده از bigram، می‌توان از فرمول زیر استفاده کرد:

$$P(w_1, w_2, \dots, w_n) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_2) \times \dots \times P(w_n|w_{n-1})$$

که در آن $P(w_i|w_{i-1})$ احتمال شرطی کلمه w_i به شرط کلمه w_{i-1} است و $P(w_1)$ احتمال کلمه اول در رشته تصادفی تولید شده است.

سپس با استفاده از روابط زیر، احتمال پیشامد جمله رندوم ایجاد شده در مینی پیکره را به دست می

آوریم:

$$P(w_1^n) \approx \prod_{k=1}^n P(w_k|w_{k-1}) \quad P(w_n|w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$

توضیحات مربوط به کارکرد هر بخش از کد:

```
import random
import re
from decimal import Decimal
from collections import Counter
import pandas as pd
```

این بخش برای وارد کردن کتابخانه‌های مورد نیاز برای اجرای برنامه استفاده می‌شود.

```
def preprocess_text(text):
    """Preprocesses the text.

    Args:
        text (str): Text to be processed.

    Returns:
        str: Processed text.

    """
    # Add starting and ending tags to all the sentences
    tagged = re.sub('\.', " </s> <s>", text)
    tagged = "<s> " + tagged[:-4]
    # Convert all characters to lowercase
    tagged_lower = tagged.lower()
    # Define the pattern to match punctuation marks
    punctuation_pattern = r'[,.!;?]'
    # Remove punctuation marks using regex
    tagged_cleaned = re.sub(punctuation_pattern, ' ', tagged_lower)
    return tagged_cleaned
```

تابع `preprocess_text` پیش‌پردازش متن را انجام می‌دهد. این تابع تگ‌های شروع و پایان (`<s>` و `>/s<`) را به هر جمله اضافه می‌کند سپس متن را به حروف کوچک تبدیل می‌کند و با استفاده از عبارت‌های منظم، علامت‌های نگارشی را حذف می‌کند.

```
def create_ngrams(text, num):
    """Creates n-grams from the given text.

    Args:
        text (str): Text to generate n-grams from.
```

```

    num (int): Number of words per n-gram.

Returns:
    list: List of n-grams.

"""

# Split the text on space and trim extra whitespaces
splitted = [x.strip() for x in text.split(" ")]
# Create n-grams based on the input using a list comprehension
ngrams = [' '.join(splitted[i:i+num]) for i in
range(len(splitted) - num + 1)]
return ngrams

```

تابع `create_ngrams` با توجه به متن ورودی `n`-gram ایجاد می کند. یک متن و تعداد کلمات هر `n`-gram را به عنوان ورودی دریافت می کند و بر اساس تعداد واژگان داده شده، `n`-gram ها را ایجاد می کند. متن ورودی بر اساس `space` به واژگان تفکیک می شود و به نوعی فضاهای خالی حذف می شوند. سپس با استفاده از یک ترکیب لیستی، `n`-gram ها ایجاد می شوند.

این تابع دو آرگومان دریافت می کند: `text` (متن برای ایجاد `n`-gram) و `num` (تعداد کلمات در هر `n`-gram). خروجی این تابع، یک لیست از `n`-gram ها است.

```

def generate_random_sentence(tokens, size):
    """Generates a random sentence.

    Args:
        tokens (set): Set of words to be used as tokens.
        size (int): Size of the desired output.

    Returns:
        str: Randomly generated sentence.

"""

    # Randomly choose 'size' different samples from the set of words
    sentence_list = random.choices(list(tokens), k=size)
    # Create the sentence from the list and add starting and ending
    tags
    return "<s> " + " ".join(sentence_list) + " </s>"

```

تابع `generate_random_sentence` با انتخاب تصادفی تعداد مشخصی با توجه به متغیر `size` واژه از مجموعه داده شده (токن ها)، یک جمله تصادفی ایجاد می کند. واژگان انتخاب شده با هم پیوند داده می شوند تا یک جمله شکل بگیرد و سپس تگ های شروع و پایان (`<s>` و `</s>`) به جمله اضافه می شوند.

این تابع دو آرگومان دریافت می‌کند: tokens (مجموعه‌ای از کلمات که به عنوان توکن‌ها استفاده می‌شود) و size (اندازه مطلوب خروجی). خروجی این تابع، یک جمله تصادفی است.

```
corpus = "Language processing, also known as natural language processing (NLP), is a field of artificial intelligence that focuses on the interaction between computers and human language. It involves the development of algorithms and models to enable computers to understand, interpret, and generate human language. NLP encompasses a wide range of tasks, including speech recognition, sentiment analysis, machine translation, information extraction, and text generation. By leveraging techniques from linguistics, computer science, and statistics, language processing aims to bridge the gap between human communication and machine understanding."
```

```
output_text = ''
```

این بلوک، متغیر corpus را با یک متن (ورودی برنامه) نمونه شامل اطلاعاتی درباره پردازش زبان و هوش مصنوعی مقداردهی اولیه می‌کند. همچنین متغیر output_text را به عنوان یک رشته خالی مقداردهی اولیه می‌کند. این متغیر به این دلیل قرار داده شده که خروجی‌ها علاوه بر اینکه در print می‌شوند در فایل ذخیره می‌شوند و مواردی که قرار است در فایل ذخیره شوند به صورت رشته با این متغیر concat می‌شوند و متغیر در هر مرحله update می‌شود.

```
# Count the number of words in the text
word_count = len(corpus.split())
word_nums = f"\nThe text contains {word_count} words.\n"
print(word_nums)
output_text += word_nums
```

این بلوک تعداد کلمات در متن corpus را با تقسیم آن بر اساس فاصله‌ها محاسبه می‌کند. سپس یک رشته به نام word_nums تولید می‌شود که شامل اطلاعات درباره تعداد کلمات است. این رشته چاپ می‌شود و به متغیر output_text اضافه می‌شود. این کار را برای محاسبه تعداد کلمات متن ورودی برنامه انجام می‌دهیم.

```
# Process the input text to prepare it for generating n-grams
processed_text = preprocess_text(corpus)
```

این بلوک با فراخوانی تابع preprocess_text، متن corpus را پردازش می‌کند. هدف از این پردازش، آماده‌سازی متن برای تولید n-gram است. متن پردازش شده در متغیر processed_text ذخیره می‌شود.

```
# Generate unigrams from the processed text
unigrams = create_ngrams(processed_text, 1)
# Generate bigrams from the processed text
bigrams = create_ngrams(processed_text, 2)
```

این بخش با استفاده از تابع `create_ngrams` ۱-gram و ۲-gram را از `processed_text` تولید می‌کند. تابع را دو بار فراخوانی می‌کند، یک بار برای تولید `unigrams` و یک بار برای تولید `bigrams`. حاصل، به ترتیب در متغیرهای `unigrams` و `bigrams` ذخیره می‌شوند.

```
# Count the total number of unigrams and bigrams
total_unigrams = len(unigrams)
total_bigrams = len(bigrams)
# Print the counts
print("Total number of unigrams:", total_unigrams)
print("Total number of bigrams:", total_bigrams, '\n')
# Count the number of unique unigrams and bigrams
unique_unigrams = len(set(unigrams))
unique_bigrams = len(set(bigrams))
output_text += "\nTotal number of unigrams: " + str(total_unigrams) +
"\n"
output_text += "Total number of bigrams: " + str(total_bigrams) +
"\n"
# Print the counts
print("Number of unique unigrams:", unique_unigrams)
print("Number of unique bigrams:", unique_bigrams, '\n')
output_text += "\nNumber of unique unigrams: " + str(unique_unigrams) +
"\n"
output_text += "Number of unique bigrams: " + str(unique_bigrams) +
"\n"
```

این بخش تعداد کلیه `unigram` و `bigram` ها را با دریافت طول لیست‌های `unigrams` و `bigrams` محاسبه می‌کند و حاصل را در متغیرهای `total_unigrams` و `total_bigrams` ذخیره می‌کند. تعداد هر کدام چاپ می‌شود و به متغیر `output_text` اضافه می‌شوند.

سپس، تعداد `unigram` و `bigram` مختص به فرد را با تبدیل لیست‌ها به مجموعه‌ها و محاسبه طول مجموعه‌ها محاسبه می‌کند و حاصل را در متغیرهای `unique_unigrams` و `unique_bigrams` ذخیره می‌کند. تعداد هر کدام چاپ می‌شود و به متغیر `output_text` اضافه می‌شوند.

```
# Add all the unigrams to a set to have a list of unique words
```

```

tokens = {token for token in unigrams if token not in {"<s>",
"</s>"}}
# Count the length of the set to get the total number of unique
tokens
token_count = len(tokens)

```

این بلوک با استفاده از یک comprehension، تمامی tokens های منحصر به فرد از لیست unigrams را در یک مجموعه به نام tokens ایجاد می کند. tokens شمرده می شود تا تعداد کل توکن های منحصر مجموعه حذف می شوند. سپس طول مجموعه tokens شمرده می شود تا تعداد کل توکن های منحصر به فرد محاسبه شود و در متغیر token_count ذخیره می شود.

```

# Generate a random sentence using the tokens and a random length
between 2 and 5
sentence = generate_random_sentence(tokens, random.randint(2, 5))

```

این بلوک با استفاده از تابع generate_random_sentence یک جمله تصادفی تولید می کند. ورودی این تابع مجموعه tokens است و یک جمله با طول تصادفی بین 2 تا 5 تولید می شود. جمله تولید شده در متغیر sentence ذخیره می شود.

```

# Create unigrams for the generated sentence
sentence_unigrams = create_ngrams(sentence, 1)
# Create bigrams for the generated sentence
sentence_bigrams = create_ngrams(sentence, 2)

```

این بلوک با استفاده از تابع create_ngrams و unigram یک جمله تولید شده را ایجاد می کند. تابع دو بار فراخوانی می شود، یک بار با پارامتر 1 برای ایجاد unigram و یک بار با پارامتر 2 برای ایجاد bigram.

sentence_unigrams و sentence_bigrams مجموعه های حاصل در متغیرهای ذخیره شوند.

```

# Set this to print all rows of tables
pd.set_option('display.max_rows', None)

```

این بلوک یک گزینه در کتابخانه pandas را تنظیم می کند تا همه ردیف های جداول هنگام چاپ نمایش داده شوند.

```

# Count the number of unigrams and bigrams using Counter
unigram_counts = Counter(unigrams)
bigram_counts = Counter(bigrams)

```

این بلوک با استفاده از کلاس Counter از مازول collections، تعداد تکرار unigram و bigram در لیست‌های unigrams و bigrams را محاسبه می‌کند. تعدادها در دیکشنری‌های unigram_counts و bigram_counts ذخیره می‌شوند.

```
# Find the maximum length of strings in the Unigram and Bigram
columns
max_unigram_length = max(len(unigram) for unigram in
unigram_counts.keys())
max_bigram_length = max(len(bigram) for bigram in
bigram_counts.keys())
```

این بخش با استفاده از حلقه for روی کلیدهای دیکشنری‌های Unigram و Bigram محاسبه می‌کند و با استفاده از تابع max() بزرگترین طول رشته را برای ستون‌های Unigram و Bigram بدست می‌آورد. بزرگترین طول‌ها به ترتیب در متغیرهای max_unigram_length و max_bigram_length ذخیره می‌شوند. از این قسمت برای چاپ جدول مناسب با جدول استفاده خواهد شد که عرض هر سلول را برای طول بزرگترین داده قرار دهد. به طور کلی چندین بلوک بعدی برای ایجاد و چاپ جدول استفاده می‌شوند.

```
# Create a table to display the counts
unigram_table = pd.DataFrame({'Unigram': list(unigram_counts.keys()),
'Count': list(unigram_counts.values())})
bigram_table = pd.DataFrame({'Bigram': list(bigram_counts.keys()),
'Count': list(bigram_counts.values())})
```

این بلوک با استفاده از کتابخانه pandas دیتافریم‌های unigram_table و bigram_table ایجاد می‌کند. دیتافریم‌ها از دیکشنری‌های unigram_counts و bigram_counts ساخته می‌شوند و دارای ستون‌هایی به نام‌های 'Unigram' و 'Count' هستند که کلیدها و مقادیر دیکشنری‌ها را در خود نگه می‌دارند.

```
# Get the maximum length of each column
unigram_max_length =
max(unigram_table['Unigram'].astype(str).map(len).max(),
len('Unigram'))
count_max_length =
max(unigram_table['Count'].astype(str).map(len).max(), len('Count'))
bigram_max_length =
max(bigram_table['Bigram'].astype(str).map(len).max(), len('Bigram'))
```

این بخش بزرگترین طول رشته‌ها را در هر ستون از دیتافریم‌های unigram_table و bigram_table و

محاسبه می‌کند. مقادیر ستون‌ها را به رشته‌ها تبدیل می‌کند، (len() تابع map(str) را به هر رشته اعمال می‌کند و max() بزرگترین طول را پیدا می‌کند. بزرگترین طول‌ها به ترتیب در متغیرهای 'Unigram' برای ستون‌های bigram_max_length و count_max_length، unigram_max_length و 'Bigram' و 'Count' ذخیره می‌شوند. آرگومان دوم() max() برای مقایسه بزرگترین طول با طول رشته نام ستون استفاده می‌شود تا مطمئن شویم نام ستون به اندازه کافی عریض است.

```
# Calculate the total width of the tables
unigram_total_width = unigram_max_length + count_max_length + 16
bigram_total_width = bigram_max_length + count_max_length + 16
# Print the Unigram Counts table with separators
print('-' * unigram_total_width)
header = f"{'':^6} {'Unigram':^{unigram_max_length}} | \
{'Count':^{count_max_length}} |"
print(header)
print('-' * unigram_total_width)
output_text += '\n' + '-' * unigram_total_width + '\n' + header +
'\n' + '-' * unigram_total_width + '\n'
```

این بلوک با جمع کردن بزرگترین طول‌های ستون‌های مربوطه (unigram_max_length) و اضافه کردن 16 برای اندازه جداکننده‌های سطرها و ستون‌ها را محاسبه می‌کند. عرض کلی جداول در متغیرهای bigram_total_width و unigram_total_width ذخیره می‌شوند. سپس جدول Unigram Counts با جداکننده‌ها چاپ می‌شود و هدر و جداکننده‌های جدول به متغیر output_text برای نوشتن در فایل اضافه می‌شوند.

```
# Print the Bigram Counts table with separators
print("\n" + '-' * bigram_total_width)
header = f"{'':^6} {'Bigram':^{bigram_max_length}} | \
{'Count':^{count_max_length}} |"
print(header)
print('-' * bigram_total_width)
output_text += '-' * bigram_total_width + '\n' + header
```

این بخش جدول Bigram Counts را با جداکننده‌ها چاپ می‌کند. جداکننده‌ها با استفاده از کارکتر '-' که در طول bigram_total_width ضرب می‌شود چاپ می‌شوند. هدر جدول در متغیر header ذخیره شده و چاپ می‌شود. جداکننده‌ها و هدر جدول به متغیر output_text برای نوشتن در فایل اضافه می‌شوند.

```
for i, row in unigram_table.iterrows():
```

```

index = str(i).center(6)
unigram = str(row['Unigram']).center(unigram_max_length)
count = str(row['Count']).center(count_max_length)
print(f" {index} {unigram} {count} ")
print('-' * unigram_total_width)
output_text += f" {index} {unigram} {count} \n"
output_text += ' ' * unigram_total_width + "\n"

```

این بخش کد در هر مرحله از حلقه بر روی ردیف‌های جدول `unigram_table` با استفاده از تابع `iterrows()` اجرا می‌شود. هر مرحله، شامل بازیابی شاخص `(i)` و مقادیر ردیف `(row)` است. شاخص به رشتہ تبدیل شده و در میانه یک عرض ۶ کاراکتر قرار داده می‌شود و به متغیر `index` اختصاص داده می‌شود. مقدار 'Unigram' از ردیف به رشتہ تبدیل شده و در میانه عرضی که با `unigram_max_length` مشخص شده است، قرار داده می‌شود و به متغیر `unigram` اختصاص داده می‌شود. مقدار 'Count' از ردیف به رشتہ تبدیل شده و در میانه عرضی که با `count_max_length` مشخص شده است قرار داده می‌شود و به متغیر `count` اختصاص داده می‌شود. ردیف قالب‌بندی شده شامل `Index` و `Count` با استفاده از علامت خط عمودی `(|)` به همراه مشخصه `Unigram` چاپ می‌شود. یک خط افقی با استفاده از حروف خط `(-)` به طول `unigram_total_width` برای جدا کردن ردیف‌ها چاپ می‌شود. ردیف قالب‌بندی شده به شکل اشاره شده به متغیر `output_text` اضافه می‌شود و یک `(\n)` برای ایجاد خط جدید پس از آن قرار می‌گیرد. یک خط افقی با استفاده `(-)` به طولی برابر با `unigram_total_width` به متغیر `output_text` اضافه می‌شود، و یک `(\n)` برای ایجاد خط جدید پس از آن قرار می‌گیرد.

```

# Calculate the sentence probability of generating the generated
sentence
total_probability = 1
# Looping through the bigrams of the sentence
for i in range(len(sentence_bigrams)):
    # Counting occurrences of the first word in each bigram in the
    # corpus's unigrams
    unigram_count = unigrams.count(sentence_unigrams[i])
    # Counting occurrences of the bigram in the corpus's bigrams
    bigram_count = bigrams.count(sentence_bigrams[i])
    # Calculating the probability with add-1 smoothing
    probability = (bigram_count + 1) / (unigram_count + token_count)
    # Updating the total probability by multiplying it with each
    # calculated bigram probability

```

```

total_probability *= probability

# Print the generated sentence and its probability
print("\nGenerated Sentence:", sentence)
print("Sentence Probability:", f"{Decimal(total_probability) * 100):.3E}\n")

```

این قسمت از کد، احتمال جمله داده شده را با حلقه‌ای به ازای bigram های آن محاسبه می‌کند.

روش add-1-smoothing برای کنترل bigram هایی که در مجموعه داده وجود ندارند، استفاده می‌کند. روند این بلوک به صورت زیر است:

۱. ابتدا احتمال کلی جمله را به یک تنظیم می‌کنیم.
۲. برای هر bigram در جمله، حلقه را اجرا می‌کنیم.
۳. برای هر bigram، تعداد تکرار کلمه اول bigram را در لیست unigram های مجموعه داده محاسبه می‌کنیم.
۴. تعداد تکرار bigram را در لیست bigram های مجموعه داده محاسبه می‌کنیم.
۵. احتمال وقوع bigram را با استفاده از روش add-1-smoothing محاسبه می‌کنیم. به تعداد تکرار bigram یک واحد اضافه کرده و آن را بر تعداد کل توکن‌ها در مجموعه داده جمع می‌کنیم.
۶. احتمال کلی را با ضرب آن در احتمال محاسبه شده برای bigram فعلی به روزرسانی می‌کنیم.
۷. مراحل ۲ تا ۶ را برای هر بیگرام در جمله تکرار می‌کنیم.
۸. در انتهای، متغیر total_probability شامل احتمال جمله بر اساس مدل bigram و روش add-1-smoothing می‌شود.

برای ایجاد جدول bigram به حالتی که شبیه به ماتریس است، از قطعه کد زیر استفاده می‌شود.

```

# Convert the set of unique words to a list
unique_words = list(set(processed_text.split()))
# Initialize an empty matrix with rows and columns for each unique word
matrix = pd.DataFrame(index=unique_words, columns=unique_words)
# Fill the matrix with zeros
matrix = matrix.fillna(0)
# Update the matrix with the counts of each bigram occurrence
for bigram in bigrams:
    word1, word2 = bigram.split()
    matrix.loc[word1, word2] += 1

```

```
# Print the Bigram Matrix
print("\nBigram Matrix:\n")
print(matrix)
```

- unique_words (لیست واژگان منحصر به فرد): مجموعه‌ی کلمات منحصر به فرد را از متن پیش‌پردازش شده به یک لیست تبدیل می‌کنیم.
 - matrix (ماتریس بایگرام): یک DataFrame خالی با استفاده از کتابخانه Pandas ایجاد می‌کنیم که ردیف‌ها و ستون‌های آن با واژگان منحصر به فرد برچسب‌گذاری شده‌اند. این یک ساختار ماتریس برای نگهداری تعداد تکرارهای بایگرام‌ها را فراهم می‌کند.
 - matrix.loc[word1, word2] += 1 (بروزرسانی ماتریس بایگرام): در هر بایگرام در لیست bigrams حلقه می‌زنیم و آن را به دو کلمه، word1 و word2 تقسیم می‌کنیم. سپس با افزایش تعداد، سلول متناظر را در ماتریس به روزرسانی می‌کنیم.
 - نمایش عنوان ماتریس بایگرام: ما یک عنوان را نشان می‌دهیم که شروع نمایش ماتریس بایگرام را نشان می‌دهد.
 - نمایش ماتریس بایگرام: خود ماتریس بایگرام را چاپ می‌کنیم که تعداد تکرارهای هر بایگرام را نشان می‌دهد.
 - اگر ماتریس بایگرام بسیار بزرگ باشد، ممکن است فضای زیادی را اشغال کند و خروجی را سخت‌تر قابل خواندن کند.
- سپس جمله تولید شده را با استفاده از متغیر sentence چاپ می‌کند. احتمال جمله را با استفاده از تابع Decimal چاپ می‌کند تا متغیر total_probability را به عنوان یک درصد با سه رقم اعشار قالب بندی کند.

```
# Write the output text to a file
with open("output.txt", "w") as file:
    file.write(output_text)
    file.write("\nGenerated Sentence: " + sentence)
    file.write("\nSentence Probability: " +
f"\n{Decimal(sentence_probability * 100):.3E}%)")
```

این بلوک متن خروجی، شامل جداول، جمله تولید شده و احتمال آن را در یک فایل به نام "output.txt" می‌نویسد. با استفاده از تابع open() و حالت "w"، فایل را در حالت نوشتن باز می‌کند. سپس محتوای متغیر output_text را با استفاده از متدهای write() در فایل می‌نویسد. در ادامه، جمله تولید شده و احتمال آن را در فایل قرار می‌دهد.

همانطور که اشاره شد، خروجی‌ها که شامل موارد گفته شده هستند، هم در print می‌شوند و هم

برای مشاهده و بررسی آسان تر در فایلی ذخیره می شوند.

خروجی:

```
The text contains 84 words.

Total number of unigrams: 92
Total number of bigrams: 91

Number of unique unigrams: 63
Number of unique bigrams: 85
```

در ابتداء تعداد کلمات متن ورودی چاپ می شود که طبق صورت پروژه باید بین ۵۰ تا ۱۰۰ کلمه باشد که در نمونه قرار داده شده برابر ۸۴ تا است.

سپس تعداد کل bigarm و unigram ها نمایش داده شده است که به ترتیب برابر ۹۲ و ۹۱ است. در دو خط آخر نیز تعداد bigarm و unigram های منحصر به فرد (بدون تکرار) قابل مشاهده است که به ترتیب برابر ۶۳ و ۸۵ است.

	Unigram	Count
0	<س>	4
1	language	5
2	processing	3
3	also	1
4	known	1
5	as	1
6	natural	1
7	(nlp)	1
8	is	1
9	a	2
10	field	1
11	of	3
12	artificial	1
13	intelligence	1
14	that	1
15	focuses	1
16	on	1
17	the	3

18	interaction	1
19	between	2
20	computers	2
21	and	6
22	human	3
23	</س>	4
24	it	1
25	involves	1
26	development	1
27	algorithms	1
28	models	1
29	to	3
30	enable	1
31	understand	1
32	interpret	1
33	generate	1
34	nlp	1
35	encompasses	1
36	wide	1

37 range 1
38 tasks 1
39 including 1
40 speech 1
41 recognition 1
42 sentiment 1
43 analysis 1
44 machine 2
45 translation 1
46 information 1
47 extraction 1
48 text 1
49 generation 1
50 by 1
51 leveraging 1
52 techniques 1
53 from 1
54 linguistics 1
55 computer 1

56 science 1
57 statistics 1
58 aims 1
59 bridge 1
60 gap 1
61 communication 1
62 understanding 1

همانطور که قابل مشاهده است پس محاسبه و نمایش bigram و unigram ها، یک جدول نمایش داده می شود که تعداد تکرار هر unigram را به همراه شاخص آن نشان می دهد. unigram ها به ترتیب ظاهر شدن در متن لیست شده اند. لازم به ذکر است همانطور که مشخص است جداول به صورت پیوسته هستند و برای قرار دادن در گزارش در چندین قسمت قرار داده شده اند.

	Bigram	Count
0	<س> language	1
1	language processing	3
2	processing also	1
3	also known	1
4	known as	1
5	as natural	1
6	natural language	1
7	processing (nlp)	1
8	(nlp) is	1
9	is a	1
10	a field	1
11	field of	1
12	of artificial	1
13	artificial intelligence	1
14	intelligence that	1
15	that focuses	1
16	focuses on	1
17	on the	1

18	the interaction	1
19	interaction between	1
20	between computers	1
21	computers and	1
22	and human	1
23	human language	2
24	language </س>	2
25	</س> <س>	3
26	<س> it	1
27	it involves	1

28	involves the	1
29	the development	1
30	development of	1
31	of algorithms	1
32	algorithms and	1
33	and models	1
34	models to	1
35	to enable	1
36	enable computers	1

37	computers to	1
38	to understand	1
39	understand interpret	1
40	interpret and	1
41	and generate	1
42	generate human	1
43	<س> nlp	1
44	nlp encompasses	1
45	encompasses a	1
46	a wide	1
47	wide range	1
48	range of	1
49	of tasks	1
50	tasks including	1
51	including speech	1
52	speech recognition	1
53	recognition sentiment	1
54	sentiment analysis	1
55	analysis machine	1

56 machine translation 1
57 translation information 1
58 information extraction 1
59 extraction and 1
60 and text 1
61 text generation 1
62 generation </s> 1
63 <s> by 1
64 by leveraging 1
65 leveraging techniques 1
66 techniques from 1
67 from linguistics 1
71 and statistics 1
72 statistics language 1
73 processing aims 1
74 aims to 1
75 to bridge 1
76 bridge the 1
77 the gap 1

78 gap between 1
79 between human 1
80 human communication 1
81 communication and 1
82 and machine 1
83 machine understanding 1
84 understanding </s> 1

جدول دیگری نمایش داده می‌شود که تعداد تکرار هر bigram را به همراه شاخص آن نشان می‌دهد. bigram ها به ترتیب ظاهر شدن در متن لیست شده‌اند.

بخشی از جدول :Bigram

Bigram Matrix:																		
leveraging	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
artificial	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
including	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
sentiment	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
of	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
(nlp)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
speech	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
algorithms	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
by	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
involves	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Generated Sentence: <s> encompasses sentiment understanding </s>
Sentence Probability: 1.291E-5%

در نهایت، جمله تولید شده و احتمال آن نمایش داده می‌شود.
کد کامل برنامه به صورت یکجا:

```
import random
import re
```

```

from decimal import Decimal
from collections import Counter
import pandas as pd

def preprocess_text(text):
    """Preprocesses the text.

    Args:
        text (str): Text to be processed.

    Returns:
        str: Processed text.
    """

    # Add starting and ending tags to all the sentences
    tagged = re.sub('.', " <s> <s>", text)
    tagged = "<s> " + tagged[:-4]

    # Convert all characters to lowercase
    tagged_lower = tagged.lower()

    # Define the pattern to match punctuation marks
    punctuation_pattern = r'[,.!;?]'

    # Remove punctuation marks using regex
    tagged_cleaned = re.sub(punctuation_pattern, '', tagged_lower)

    return tagged_cleaned

def create_ngrams(text, num):
    """Creates n-grams from the given text.

    Args:
        text (str): Text to generate n-grams from.
        num (int): Number of words per n-gram.

    Returns:
        list: List of n-grams.
    """

```

```

# Split the text on space and trim extra whitespaces
splitted = [x.strip() for x in text.split(" ")]

# Create n-grams based on the input using a list comprehension
ngrams = [' '.join(splitted[i:i+num]) for i in range(len(splitted) - num + 1)]

return ngrams

def generate_random_sentence(tokens, size):
    """Generates a random sentence.

    Args:
        tokens (set): Set of words to be used as tokens.
        size (int): Size of the desired output.

    Returns:
        str: Randomly generated sentence.
    """
    # Randomly choose 'size' different samples from the set of unique words
    sentence_list = random.choices(list(tokens), k=size)

    # Create the sentence from the list and add starting and ending tags
    return "<s> " + ".join(sentence_list) + " </s>"

corpus = "Language processing, also known as natural language processing (NLP), \
is a field of artificial intelligence that focuses on the interaction between \
computers and human language. It involves the development of algorithms and \
models to enable computers to understand, interpret, and generate human language. \
\
NLP encompasses a wide range of tasks, including speech recognition, sentiment \
analysis, machine translation, information extraction, and text generation. \
By leveraging techniques from linguistics, computer science, and statistics, \
language processing aims to bridge the gap between human communication and \
machine understanding."

output_text = ''

```

```

# Count the number of words in the text
word_count = len(corpus.split())
word_nums = f"\nThe text contains {word_count} words.\n"
print(word_nums)
output_text += word_nums

# Process the input text to prepare it for generating n-grams
processed_text = preprocess_text(corpus)

# Generate unigrams from the processed text
unigrams = create_ngrams(processed_text, 1)

# Generate bigrams from the processed text
bigrams = create_ngrams(processed_text, 2)

# Count the total number of unigrams and bigrams
total_unigrams = len(unigrams)
total_bigrams = len(bigrams)

# Print the counts
print("Total number of unigrams:", total_unigrams)
print("Total number of bigrams:", total_bigrams, '\n')

output_text += "\nTotal number of unigrams: " + str(total_unigrams) + "\n"
output_text += "Total number of bigrams: " + str(total_bigrams) + "\n"

# Count the number of unique unigrams and bigrams
unique_unigrams = len(set(unigrams))
unique_bigrams = len(set(bigrams))

output_text += "\nNumber of unique unigrams: " + str(unique_unigrams) + "\n"
output_text += "Number of unique bigrams: " + str(unique_bigrams) + "\n"

# Add all the unigrams to a set to have a List of unique words
tokens = {token for token in unigrams if token not in {"<s>", "</s>"}}

# Count the length of the set to get the total number of unique tokens
token_count = len(tokens)

```

```

# Generate a random sentence using the tokens and a random length between 2 and 5
sentence = generate_random_sentence(tokens, random.randint(2, 5))

# Create unigrams for the generated sentence
sentence_unigrams = create_ngrams(sentence, 1)

# Create bigrams for the generated sentence
sentence_bigrams = create_ngrams(sentence, 2)

# Set this to print all rows of tables
pd.set_option('display.max_rows', None)

# Count the number of unigrams and bigrams using Counter
unigram_counts = Counter(unigrams)
bigram_counts = Counter(bigrams)

# Find the maximum length of strings in the Unigram and Bigram columns
max_unigram_length = max(len(unigram) for unigram in unigram_counts.keys())
max_bigram_length = max(len(bigram) for bigram in bigram_counts.keys())

# Create a table to display the counts
unigram_table = pd.DataFrame({'Unigram': list(unigram_counts.keys()), 'Count': list(unigram_counts.values())})
bigram_table = pd.DataFrame({'Bigram': list(bigram_counts.keys()), 'Count': list(bigram_counts.values())})

# Get the maximum length of each column
unigram_max_length = max(unigram_table['Unigram'].astype(str).map(len).max(),
len('Unigram'))
count_max_length = max(unigram_table['Count'].astype(str).map(len).max(),
len('Count'))
bigram_max_length = max(bigram_table['Bigram'].astype(str).map(len).max(),
len('Bigram'))

# Calculate the total width of the tables
unigram_total_width = unigram_max_length + count_max_length + 16
bigram_total_width = bigram_max_length + count_max_length + 16

```

```

# Print the Unigram Counts table with separators
print('-' * unigram_total_width)
header = f"{'':^6} {'Unigram':^{unigram_max_length}} |"
{'Count':^{count_max_length}} |"
print(header)
print('-' * unigram_total_width)

output_text += '\n' + '-' * unigram_total_width + '\n' + header + '\n' + '-' * unigram_total_width + '\n'

for i, row in unigram_table.iterrows():
    index = str(i).center(6)
    unigram = str(row['Unigram']).center(unigram_max_length)
    count = str(row['Count']).center(count_max_length)
    print(f"{'':^6} {'{unigram}':^{unigram_max_length}} |")
    print('{'Count':^{count_max_length}} |")
    print('-' * unigram_total_width)
    output_text += f"{'':^6} {'{unigram}':^{unigram_max_length}} | {count} |\n"
    output_text += '-' * unigram_total_width + "\n"

# Print the Bigram Counts table with separators
print("\n" + '-' * bigram_total_width)
header = f"{'':^6} {'Bigram':^{bigram_max_length}} |"
{'Count':^{count_max_length}} |"
print(header)
print('-' * bigram_total_width)

output_text += '\n' + '-' * bigram_total_width + '\n' + header + '\n' + '-' * bigram_total_width + '\n'

for i, row in bigram_table.iterrows():
    index = str(i).center(6)
    bigram = str(row['Bigram']).center(bigram_max_length)
    count = str(row['Count']).center(count_max_length)
    print(f"{'':^6} {'{bigram}':^{bigram_max_length}} |")
    print('{'Count':^{count_max_length}} |")
    print('-' * bigram_total_width)
    output_text += f"{'':^6} {'{bigram}':^{bigram_max_length}} | {count} |\n"
    output_text += '-' * bigram_total_width + "\n"

```

```

# Convert the set of unique words to a list
unique_words = list(set(processed_text.split()))

# Initialize an empty matrix with rows and columns for each unique word
matrix = pd.DataFrame(index=unique_words, columns=unique_words)

# Fill the matrix with zeros
matrix = matrix.fillna(0)

# Update the matrix with the counts of each bigram occurrence
for bigram in bigrams:
    word1, word2 = bigram.split()
    matrix.loc[word1, word2] += 1

# Print the Bigram Matrix
print("\nBigram Matrix:\n")
print(matrix)

# Calculate the sentence probability of generating the generated sentence
total_probability = 1

# Looping through the bigrams of the sentence
for i in range(len(sentence_bigrams)):
    # Counting occurrences of the first word in each bigram in the corpus's
    unigrams
    unigram_count = unigrams.count(sentence_unigrams[i])
    # Counting occurrences of the bigram in the corpus's bigrams
    bigram_count = bigrams.count(sentence_bigrams[i])
    # Calculating the probability with add-1 smoothing
    probability = (bigram_count + 1) / (unigram_count + token_count)
    # Updating the total probability by multiplying it with each calculated
    bigram probability
    total_probability *= probability

# Print the generated sentence and its probability
print("\nGenerated Sentence:", sentence)
print("Sentence Probability:", f"{{Decimal(total_probability * 100):.3E}}%\n")

```

```
# Write the output text to a file
with open("output.txt", "w") as file:
    file.write(output_text)
    file.write("\nGenerated Sentence: " + sentence)
    file.write("\nSentence Probability: " + f"{Decimal(total_probability * 100):.3E}%")


# Add the matrix to the output text
output_text += "\n\nBigram Matrix:\n\n" + str(matrix)

print('\nOutput text:\n')
print(output_text)
```