НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
Факультет Інформатики та обчислювальної техніки
Кафедра Інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №3

з дисципліни «Обробка та аналіз текстових даних на Python»

на тему «Аналіз тексту за допомогою NLTK»

Варіант №2

Виконала: студентка групи ІС-з21
Коломієць Катерина Миколаївна
23.05.2025

Перевірив: асист. Мягкий М. Ю.

Київ – 2025

GitHub: https://github.com/kate-miiets-uni/oatd-py

Теоретичні відомості

Токенізація (англ. tokenization) — це процес розбиття послідовності тексту на менші одиниці, які називаються токенами. Ці токени можуть бути словами, окремими символами, підсловами або іншими значущими частинами тексту, залежно від завдання. Токенізація є одним з перших кроків у багатьох задачах NLP, оскільки більшість алгоритмів обробки тексту працюють не з сирим текстом, а з окремими значущими одиницями. Токенізований текст легше аналізувати, обробляти та використовувати для подальших завдань.

Бібліотека NLTK (Natural Language Toolkit) — це потужна та широко використовувана бібліотека Python для роботи з обробкою природної мови (NLP). Деякі з корисних модулів NLTK:

-токенізація (nltk.tokenize): розбиття тексту на слова (word_tokenize), речення (sent_tokenize);

- стеммінг та лематизація (nltk.stem): стеммінг – це зведення слова до його кореневої форми шляхом відкидання суфіксів (наприклад, PorterStemmer, LancasterStemmer), лематизація – це зведення слова до його леми (словникової форми), враховуючи контекст (наприклад, WordNetLemmatizer);

- частотний аналіз (nltk.probability): інструменти для обчислення частоти слів та інших лінгвістичних одиниць (FreqDist).

Хід роботи:

1. Імпортуємо бібліотеки requests та nltk. Надсилаємо на сервер запит про отримання тексту.

```
import requests
import nltk



url = 'https://www.unian.ua/world/karti-groshi-p-yatero-kapibar-13017267.html'

# Надсилаємо запит на отримання тексту
response = requests.get(url)
```

2. Робимо перевірку чи позитивна відповідь на запит, якщо status_code == 200, все добре і йдемо далі.

```
if response.status_code == 200:
    text = response.text
```

3. Виконуємо токенізацію тексту. Для цього в першу чергу необхідно завантажити попередньо навчений токенізатор 'punkt', який використовується для розбиття тексту на

слова або речення. Далі використовуємо функцію .word_tokenize(), щоб розбити текст на токени – слова, номери, пунктуаційні символи.

```
# Токенізація тексту
nltk.download('punkt')
tokens = nltk.word_tokenize(text.lower()) # Приводимо до нижнього регістру для єдності
```

Для чого потрібен 'punkt'? Це набір попередньо навчених параметрів, які допомагають алгоритму токенізації приймати обґрунтовані рішення щодо того, де закінчується одне слово і починається інше, враховуючи контекст та типові патерни мови, у свою чергу не містить у собі правил розділу певного рядка, адже це було б нерозумно використовувати узагальнені правил для будь-яких ситуацій. Натомість, токенізатори використовують навчені статистичні моделі для більш інтелектуальної та точної токенізації, ніж це було б можливо з простим набором жорстких правил.

4. Формуємо біграми за допомогою функції nltk.bigrams(), яка формує список з пар слів, кожна біграма має два слова.

```
# Формування біграм
bigram_list = list(nltk.bigrams(tokens))
```

5. Визначаємо частоту появи кожної біграми за допомогою функції nltk.FreqDist(). Функція підраховує кількість появи кожної біграми в рядку та зберігає біграму разом з кількістю її появи в рядку у форматі словника.

```
# Визначення частоти появи біграм
freq_dist = nltk.FreqDist(bigram_list)
```

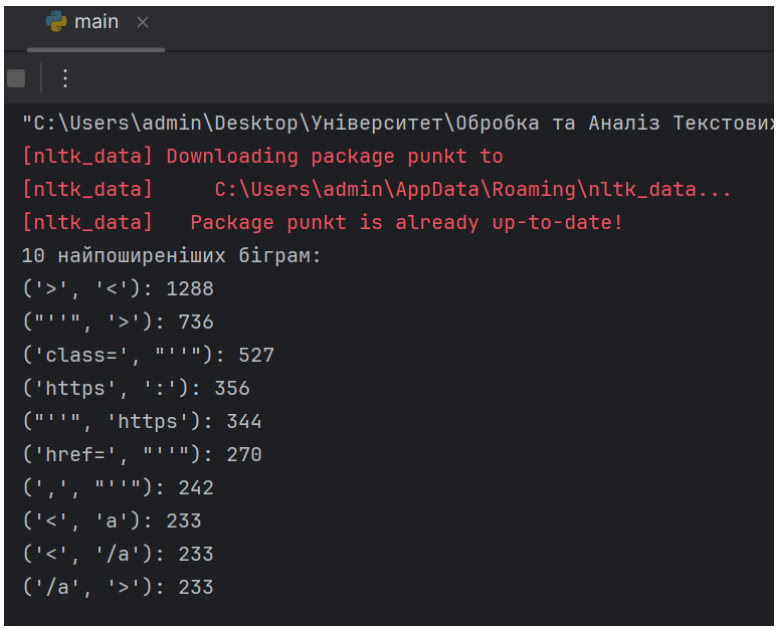
6. Виводимо на консоль 10 біграм, які з’являються в тексті найчастіше.

```
# Вивід 10 найпоширеніших біграм
for bigram, count in freq_dist.most_common(10):
    print(f"{bigram}: {count}")
```

7. У випадку, якщо відповідь від сервера на запит невдала, повідомлення про це виводиться на консоль.

```
# Якщо запит невдалий
else:
    print("Не вдалося завантажити сторінку.")
```

Результат:



Висновок

Отже у ході лабораторної роботи я дізналася про бібліотеку для обробки природної мови, а також познайомилася з деякими з її функцій для токенізації рядка, групування токенів у біграми та визначення частоти появи біграм у рядку. У процесі ми отримали текст із сайту, поділили текст на токени та згенерували з них список біграм, після чого створили словник, який містить список біграм та відповідні їм значення частоти появи в рядку. У кінці ми вивели 10 найпоширеніших біграм на консоль.

Контрольні запитання

1. Що таке біграма?

Біграма (від грецького "bi" - два та "gramma" - літера, запис) — це послідовність із двох сусідніх елементів у більшій послідовності. Зазвичай під "елементами" розуміють літери, склади або слова. Найчастіше термін "біграма" використовується в лінгвістиці та обробці природної мови (NLP) для аналізу частоти зустрічальності пар літер або пар слів у тексті.

2. Як обчислювати частоту біграм?

Частоту біграм можна обчислити за допомогою функції FreqDist(), яка бере на вхід список біграм, та проходить по всьому рядку, шукаючи збіги з записаними біграмами, після чого створює словник, у якому зберігає список біграм та відповідні їм значення частот появи в рядку.

3. Для чого використовуються біграми в аналізі текстів?

Оскільки біграми надають інформацію про те, які слова або літери часто зустрічаються поруч, їх можуть застосовувати для визначення мови тексту, оскільки різні

мови мають різні типові послідовності літер або слів. Також, для статистичного аналізу мови, для перевірки правопису та граматики, для класифікації тексту та для вилучення ключових стіл з тексту.

4. Які методи є в бібліотеці NLTK для роботи з біграмами?

У бібліотеці NLTK має модуль util з функцією bigrams() для створення списку біграм. Також модуль probability, який містить функцію FreqDist() для підрахунку частоти появи кожної біграми в рядку.

5. Як перевірити якість токенізації?

Найпростішим методом оцінки якості буде візуальна інспекція певної вибірки токенів. За такої інспекції можна відібрати абсолютно різні елементи списку та сформувати уявлення про набір помилок, які варто вирішити. Також непоганим способом буде оцінка вирішення кінцевої задачі. Кінцеві результати можуть дати уявлення про якість токенізації.

6. Що означає контекст у текстовому аналізі?

У текстовому аналізі контекст відноситься до слів, фраз або навіть цілих речень, які оточують певне слово або вираз і допомагають зрозуміти його значення та інтерпретацію. Контекст є ключовим для розв'язання багатьох проблем NLP, оскільки одне й те саме слово може мати різні значення залежно від того, в якому оточенні воно вживається.

7. Які переваги біграм перед уніграмами?

Біграми, на відміну від окремих слів (уніграм), враховують контекст сусідніх слів, що дає краще розуміння словосполучень, стилю тексту та допомагає в багатьох задачах NLP (наприклад, автозавершення, розпізнавання мови).

8. Які можливі помилки при аналізі біграм?

При аналізі біграм можна отримати розріджені дані, втратити віддалений контекст, отримати спотворені результати через погану токенізацію та залежність від корпусу. Також можуть заважати стоп-слова та ігнорування різних форм слів.

9. Як вивести результат роботи у зручному форматі?

Для виведення даних словника у зручному форматі можна скористатися стороннім бібліотеками типу pandas, або просто використати цикл for для зручного для читання виведення даних у стовпчик.

10. Чим відрізняються токенізація на слова та на речення?

Токенізація на слова (word tokenization) розбиває текст на окремі слова (а також часто включає розділові знаки як окремі токени). Основна мета - виділити з тексту значущі лексичні одиниці. Токенізація на речення (sentence tokenization) розбиває текст на окремі речення. Основна мета - визначити межі між реченнями в тексті.