МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

П. К. Ніколюк

ТЕХНОЛОГІЇ TEXTMINING AND WEBMINING

Методичні вказівки
до виконання самостійних та лабораторних робіт
для здобувачів вищої освіти ОС «Бакалавр»
спеціальності 122 Комп’ютерні науки ОП «Комп’ютерні науки»

Вінниця
2024

УДК 004.9+004.62](075.4)
Н 64

Рекомендовано до друку вченою радою
факультету інформаційних і прикладних технологій
Донецького національного університету імені Василя Стуса
(протокол № 11 від 21 травня 2024 р.)

Автор: *П. К. Ніколюк*, д-р фіз.-мат. наук, професор, професор кафедри інформаційних технологій Донецького національного університету імені Василя Стуса.

Рецензенти: *В. Г. Крижановський,* д-р техн. наук, професор, професор кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса;
В. І. Копитко, д-р екон. наук, професор, професор кафедри менеджменту та міжнародного бізнесу Львівського інституту менеджменту.

Ніколюк П. К.

Н 64 Технології TextMining and WebMining: методичні вказівки до виконання самостійних та лабораторних робіт для здобувачів вищої освіти ОС «Бакалавр» спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки». Вінниця: ДонНУ імені Василя Стуса, 2024. 72 с.

Методичні вказівки є навчально-методичним документом, який містить рекомендації для оптимального розв'язання прикладних задач під час виконання самостійних та лабораторних робіт з дисципліни «Технології TextMining and WebMining». До кожної лабораторної роботи наведено теоретичні відомості та методичні вказівки до виконання завдань.

УДК 004.9+004.62](075.4)

ЗМІСТ

Лабораторна робота № 1
Розбиття тексту на токени та речення
Теоретичні відомості

Токенізація, лематизація і стемінг – це три ключових етапи обробки тексту в області обробки природної мови (Natural Language Processing, NLP).

1. Токенізація. Це процес розбиття тексту на окремі фрагменти, які називаються токенами. Токени можуть бути словами, реченнями або навіть символами, залежно від вимог конкретного завдання. Наприклад, речення "Це речення для прикладу." може бути розкладене на токени: ["Це", "речення", "для", "прикладу", "."].

2. Лематизація. Це процес нормалізації слів до їх базової форми, яка називається лемою або лемою слова. Наприклад, лематизація перетворить слова "бігаю", "бігав", "бігає" на базову форму "бігати". Це допомагає зменшити розмір словника та покращує узагальнення в обробці тексту.

3. Стемінг. Це процес в обробці природної мови (Natural Language Processing, NLP), в якому слова зводяться до їхньої основи або кореня (stem), відкидаючи афікси. Основна мета стемінгу – зменшення слова до його базової форми, щоб забезпечити більш ефективний аналіз тексту.

Стемінг відрізняється від лематизації тим, що він не завжди повертає правильну форму слова. Наприклад, слова "running", "runs", "ran" після стемінгу можуть бути зведені до одного кореня "run". У результаті цього може виникати деяка втрата інформації, оскільки стемінг не завжди враховує контекст або граматичні правила.

Стемери – це програми або алгоритми, які виконують стемінг. Вони можуть бути побудовані на основі правил (наприклад, алгоритм Портера) або використовувати статистичні методи. Використання стемерів може бути корисним у випадках, коли швидкість обробки важливіша за точність або коли контекст не так важливий.

У випадках, коли потрібна більш точна нормалізація слів, лематизація зазвичай є більш важливою ніж стемінг. Однак обидва методи можуть бути корисними в різних випадках залежно від конкретних потреб задачі.

Розуміння та застосування токенізації, лематизації та стемінгу допомагає у розв’язанні багатьох завдань NLP, таких як аналіз сентиментів, класифікація тексту, машинний переклад, автоматичне розпізнавання мови та багато інших.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень понять про розбиття тексту на токени та речення.

Завдання до лабораторної роботи

1. Проведення досліджень розбиття тексту на токени та речення.

2. Ручне розбиття тексту на токени та речення.

3. Обробка результатів та їхнє графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджується розбиття тексту на токени та речення.

2. Реалізація розбиття тексту на токени та речення є двоступеневою процедурою.

– Перший ступінь – розбиття тексту на токени.

– Другий ступінь – розбиття тексту на речення.

Вибір пропозиції та порядку сортування з табл. 1.1 – відповідає номеру варіанта.

Таблиця 1.1

Індивідуальні завдання

№ варіанта	Пропозиції	Функція розбиття на токени
1	Гарні кекси коштують $3.88\ny Нью-Йорку. Будь ласка купіть мені три.\n\nДякую.	word_tokenize
2	Смачні гамбургери коштують $3.88\ny Лос-Анджелесі. Будь ласка купіть мені два.\n\nДякую.	word_tokenize
3	Прохолодна кола коштує $4.88\ny Чикаго. Будь ласка купіть мені дві.\n\nДякую.	word_tokenize
4	Гаряча піца коштує $6.25\ny Лас-Вегасі. Будь ласка купіть мені одну.\n\nДякую.	word_tokenize
5	Гостра паста коштує $7.15\ny Х'юстоні. Будь ласка купіть мені одну.\n\nДякую.	word_tokenize
6	Гарні кекси коштують $3.88\ny Нью-Йорку. Будь ласка купіть мені три.\n\nДякую.	wordpunct_tokenize
7	Смачні гамбургери коштують $3.88\ny Лос-Анджелесі. Будь ласка купіть мені два.\n\nДякую.	wordpunct_tokenize
8	Прохолодна кола коштує $4.88\ny Чикаго. Будь ласка купіть мені дві.\n\nДякую.	wordpunct_tokenize
9	Гаряча піца коштує $6.25\ny Лас-Вегасі. Будь ласка купіть мені одну.\n\nДякую.	wordpunct_tokenize
10	Гостра паста коштує $7.15\ny Х'юстоні. Будь ласка купіть мені одну.\n\nДякую.	wordpunct_tokenize
11	Гарні кекси коштують $3.88\ny Нью-Йорку. Будь ласка купіть мені три.\n\nДякую.	regexp_tokenize
12	Смачні гамбургери коштують $3.88\ny Лос-Анджелесі. Будь ласка купіть мені два.\n\nДякую.	regexp_tokenize
13	Прохолодна кола коштує $4.88\ny Чикаго. Будь ласка купіть мені дві.\n\nДякую.	regexp_tokenize

№ варіанта	Пропозиції	Функція розбиття на токени
14	Гаряча піца коштує $6.25\ny Лас-Вегасі. Будь ласка купіть мені одну.\n\nДякую.	regexp_tokenize
15	Гостра паста коштує $7.15\ny Х'юстоні. Будь ласка купіть мені одну.\n\nДякую.	regexp_tokenize

Порядок оформлення лабораторної роботи 1.1

1.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис розбиття тексту на токени та речення.
2. Результати розбиття тексту на токени та речення.
3. Аналіз закономірностей.

- Висновки.

1.2. Вступ має містити, окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

1.3. У звіті проводиться розбиття тексту на токени та речення.

1.4. У висновках мають бути підсумки роботи та порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

Наведемо приклад програми токенізації тексту.

Listing 1.1

```
!pip install nltk
import nltk
# викачування для пунктуації
nltk.download('punkt')
s = "Гарні кекси коштують $3.88\ny Нью-Йорку. Будь ласка купіть мені два.\n\nДякую."
# розбиття тексту на токени
print(nltk.tokenize.word_tokenize(text=s))
# ['Гарні', 'кекси', 'коштують', '$', '3.88', 'у', 'Нью-Йорку', '.', 'Будь', 'ласка', 'купіть', 'мені', 'два',
'.', 'Дякую', '.']
# розбиття тексту на токени (можна розбити спеціальні символи і дійсні числа)
print(nltk.tokenize.wordpunct_tokenize(text=s))
# ['Гарні', 'кекси', 'коштують', '$', '3', '.', '88', 'у', 'Нью', '-', 'Йорку', '.', 'Будь', 'ласка', 'купіть',
'мені', 'два', '.', 'Дякую', '.']
# розбиття тексту на речення
print(nltk.tokenize.sent_tokenize(text=s))
# ['Гарні кекси коштують $3.88\ny Нью-Йорку.', 'Будь ласка купіть мені два.', 'Дякую.']
# розбиття тексту на токени за допомогою вказаних регулярних виразів
print(nltk.tokenize.regexp_tokenize(text=s, pattern=r'\w+|\$[\d\.]+|\S+'))
```

```
# ['Гарні', 'кекси', 'коштують', '$3.88', 'у', 'Нью', '-Йорку.', 'Будь', 'ласка', 'куліть', 'мені', 'два', '.', 'Дякую', '.']
```

Наведемо програму, що застосовує процедуру стемінгу.

Listing 1.2

```
# процедура stemming – скорочення слова до його базової чи кореневої форми
import nltk
nltk.download('punk')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('snowball_data')
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
text = "NLTK is a great package for working with natural language data."
# Tokenize into words
words = word_tokenize(text)
# Stemming
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in words]
print(stemmed_words)
```

Лабораторна робота № 2
Морфологічний аналіз
Теоретичні відомості

Морфологічний аналіз – це процес аналізу мови з точки зору її морфологічних складових, таких як слова, їхні форми та структури. У лінгвістиці морфологія вивчає будову слова та способи, за допомогою яких вони утворюються і змінюються.

Основні завдання морфологічного аналізу містять:

1. Визначення частин мови (іменників, прикметників, дієслів тощо).

2. Виявлення граматичних категорій, таких як число, рід, відмінок, час, особа та інші.

3. Встановлення словозміни – тобто аналіз форм слів у різних контекстах та їхні взаємозв'язки.

Морфологічний аналіз є важливою складовою багатьох прикладних областей, таких як обробка природної мови, комп'ютерні переклади, інформаційний пошук, автоматична індексація текстів тощо. У цих областях аналіз морфології допомагає комп'ютерам розуміти та обробляти мовний контент, що відкриває широкі можливості для розробки інтелектуальних програм і систем.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень понять про проведення морфологічного аналізу.

Завдання до лабораторної роботи

1. Проведення досліджень морфологічного аналізу.

2. Ручне проведення морфологічного аналізу.

3. Обробка результатів і їхнє графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджується морфологічний аналіз.

2. Реалізація морфологічного аналізу є триступеневою процедурою.

– Перший ступінь – визначення леми.

– Другий ступінь – визначення частини мови.

– Третій ступінь – визначення морфологічних ознак.

Вибір двох слів із табл. 2.1 – відповідає номеру варіанта.

Таблиця 2.1

Індивідуальні завдання

№ варіанта	Перше слово	Друге слово
1	людина	розумна
2	тварина	дурна
3	розумна	сіла
4	дурна	встала
5	сіла	вона
6	встало	воно
7	вона	людина
8	воно	тварина
9	людина	дурна
10	тварина	розумна
11	розумна	встала
12	дурна	сіла
13	сіло	воно
14	встала	вона
15	вона	тварина
16	вона	людина

Порядок оформлення лабораторної роботи

2.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис морфологічного аналізу.
2. Результати морфологічного аналізу.
3. Аналіз закономірностей.

- Висновки.

2.2. Вступ має містити, окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

2.3. У звіті виконується морфологічний аналіз.

2.4. У висновках мають бути підсумки роботи та порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

У Listing 2.1 наведений приклад програми морфологічного аналізу.

Listing 2.1

```
!pip install pymorphy2
!pip install pymorphy2-dicts-uk
import pymorphy2
## стандартний морфологічний аналіз для української мови
ma=pymorphy2.MorphAnalyzer(lang='uk')
```

```
## дієслово
print(ma.parse('сіли'))
# [Parse(word='сіли', tag=OpencorporaTag('VERB,perf plur,past'), normal_form='сісти',
score=1.0, methods_stack=((DictionaryAnalyzer(), 'сіли', 1101, 14),))]
gr=ma.parse('сіли')[0]
# лема
print(gr.normal_form) # 'сісти'
print(gr.tag) # VERB,perf plur,past
# частина мови
print(gr.tag.POS) # VERB
# одухотвореність
print(gr.tag.animacy) # None
# доконаний і недоконаний вид
print(gr.tag.aspect) # perf
# відмінок
print(gr.tag.case) # None
# рід
print(gr.tag.gender) # None
# включеність того, хто говорить, у дію
print(gr.tag.involvement) # None
# спосіб (наказовий, дійсний)
print(gr.tag.mood) # None
# число
print(gr.tag.number) # plur
print(gr.tag.person) # None
# час
print(gr.tag.tense) # past
# перехідність
print(gr.tag.transitivity) # None
# стан (активний, пасивний)
print(gr.tag.voice) # None
```

Лабораторна робота № 3
Створення дерев синтаксичної залежності та фразової структури
Теоретичні відомості

Дерева синтаксичної залежності (Dependency Trees) є системою, яка відображає синтаксичну структуру речень. Принцип їхнього створення базується на встановленні зв'язків між словами у реченні та організації їх у відповідну ієрархічну структуру.

Основні кроки створення дерева синтаксичної залежності містять:

1. Токенізація: речення спочатку розбивається на окремі токени (слова), інколи із врахуванням інших елементів, наприклад пунктуації.

2. Морфологічний аналіз: кожен токен аналізується з морфологічної точки зору для визначення таких параметрів, як частина мови, відмінок, число та інші морфологічні властивості.

3. Синтаксичний аналіз: на основі результатів морфологічного аналізу та з використанням граматичних правил мови встановлюються синтаксичні відношення між словами. Ці відношення відображаються у вигляді графа або дерева.

4. Побудова дерева: використовуючи встановлені синтаксичні зв'язки, слова та речення організуються у вигляді дерева, де кожен вузол є одним словом, а ребра вказують на синтаксичні залежності між словами.

Складові дерева синтаксичної залежності містять:

1. Корінь дерева: це слово або фраза, яка не має синтаксичної залежності від інших слів у реченні. Вона може бути головним дієсловом речення чи іншою синтаксичною одиницею.

2. Залежності: це синтаксичні відношення між словами у реченні. Наприклад, підмет може залежати від головного дієслова у реченні, а прикметник може залежати від іменника.

3. Листя дерева: це слова, які не мають синтаксичних залежностей від інших слів у реченні.

Створення дерев синтаксичної залежності є важливою складовою для багатьох прикладних областей обробки природної мови, зокрема для машинного перекладу, аналізу тексту та автоматичного витягування інформації.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень понять про створення дерев синтаксичної залежності та дерев фразової структури складових.

Завдання до лабораторної роботи

1. Проведення досліджень дерев синтаксичної залежності та дерев фразової структури.

2. Ручне створення дерев синтаксичної залежності та дерев фразової структури.

3. Обробка результатів та їх графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджуються дерева синтаксичної залежності та дерева фразової структури.

2. Реалізація дерев синтаксичної залежності та дерев фразової структури є двоступеневою процедурою.

– Перший ступінь – створення дерева синтаксичної залежності.

– Другий ступінь – створення дерева фразової структури.

Вибір структури пропозиції з табл. 3.1 – відповідає номеру варіанта.

Таблиця 3.1

Індивідуальні завдання

№ варіанта	Валентність базової предикатної синтаксеми	Тип другої предикатної синтаксеми
1	3	адвербальна
2	4	адвербальна
3	5	адвербальна
4	6	адвербальна
5	7	адвербальна
6	3	модальна
7	4	модальна
8	5	модальна
9	6	модальна
10	7	модальна
11	3	атрибутивна
12	4	атрибутивна
13	5	атрибутивна
14	6	атрибутивна
15	7	атрибутивна

Порядок оформлення лабораторної роботи

3.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис дерев синтаксичної залежності та дерев фразової структури.

2. Результати аналізу дерев синтаксичної залежності та дерев фразової структури.

3. Аналіз закономірностей.

- Висновки.

3.2. Вступ має містити, окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

3.3. У звіті наводяться дерева синтаксичної залежності та безпосередні складові.

3.4. У висновках здійснюється порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

Приклади

На рис. 3.1 наведено приклад дерева синтаксичної залежності.

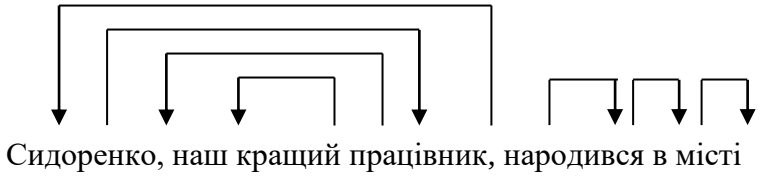

Рис. 3.1. Приклад дерева синтаксичної залежності

На рис. 3.2 наведено приклад дерева власне складових та дерева фразової структури.

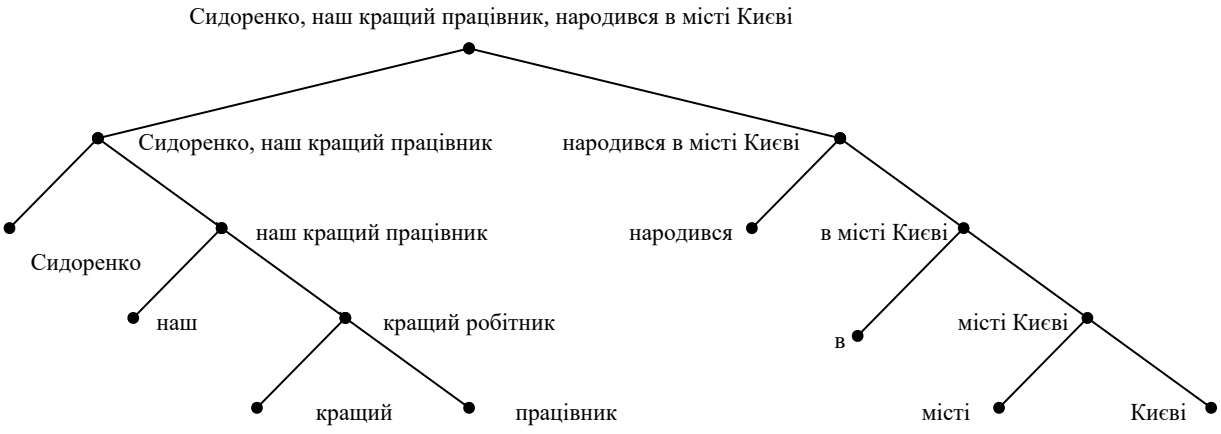

Рис. 3.2. Приклад дерева власне складових та дерева фразової структури

Далі наведемо (Listing 3.1) програми, що є варіантами дерева власне складових та дерева фразової структури.

Listing 3.1

```
import nltk
sample_text= "I am a coding ninja, and I am the best in coding."
tokenized=nltk.sent_tokenize(sample_text)
for i in tokenized:
words=nltk.word_tokenize(i)
tagged_words=nltk.pos_tag(words)
print(tagged_words)
chunkGram = r"""Chunk: {<RB.?>*<VB.?>*<NNP>+<NN>?}""" # this is the grammar that we define,
chunkParser=nltk.RegexpParser(chunkGram)
chunked=chunkParser.parse(tagged_words)
chunked.draw()
```

Наступна програма (Listing 3.2) є графічним зображенням дерева синтаксичної залежності.

Listing 3.2

```
import networkx as nx
import matplotlib.pyplot as plt
def draw_dependency_tree(dependency_tree):
G = nx.DiGraph()
for token in dependency_tree:
head, _, relation = token[6], token[0], token[7]
G.add_edge(head, token[0], label=relation)
pos = nx.spring_layout(G)
labels = {(i, j): f"{i}->{j}\n{label}" for i, j, label in G.edges(data='label')}
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.show()
# Приклад дерева синтаксичної залежності у вигляді списку кортежів (формат залежностей CoNLL)
dependency_tree = [
(1, 'The', '_', 'DET', '_', '_', 2, 'det'),
(2, 'cat', '_', 'NOUN', '_', '_', 4, 'nsubj'),
(3, 'sat', '_', 'VERB', '_', '_', 4, 'ROOT'),
(4, 'on', '_', 'ADP', '_', '_', 0, 'prep'),
(5, 'the', '_', 'DET', '_', '_', 6, 'det'),
(6, 'mat', '_', 'NOUN', '_', '_', 4, 'pobj')
]
draw_dependency_tree(dependency_tree)
```

На рис. 3.3 подано візуальну картину дерева синтаксичної залежності.

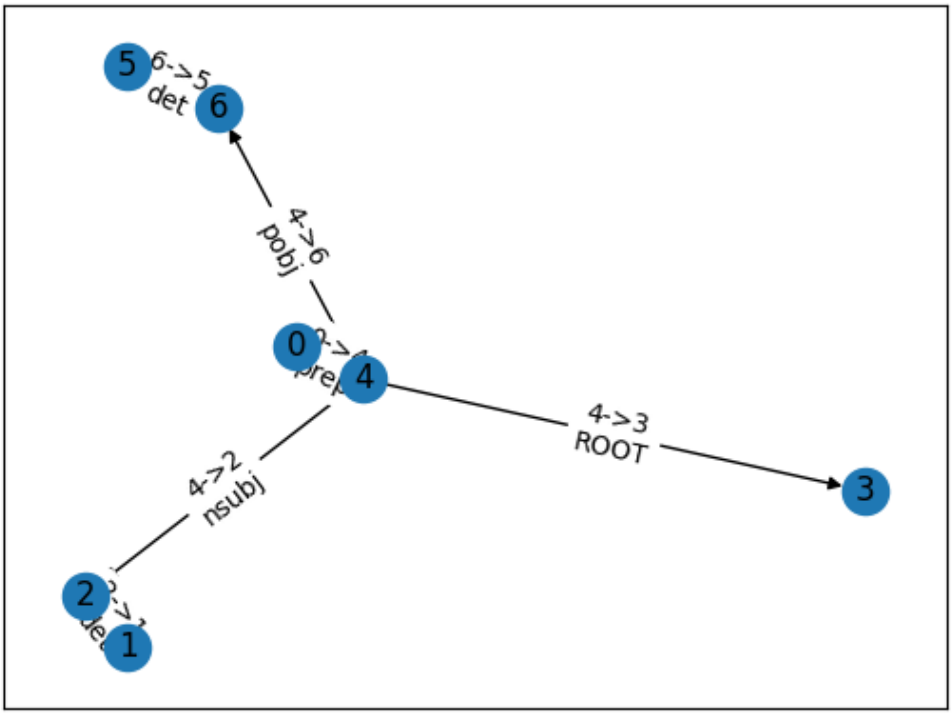

Рис. 3.3. Приклад дерева синтаксичної залежності у вигляді списку кортежів

Програма (Listing 3.2) створює граф (рис. 3.3) за допомогою бібліотеки NetworkX, де вершини є токенами речення, а ребра є залежностями між ними. Функція **draw_dependency_tree** відповідає за побудову та візуалізацію графа.

«nsubj» у лінгвістиці означає підмет (subject). У залежній структурі речення «nsubj» вказує на те, що дане слово є підметом у реченні. Наприклад, у реченні «The cat sat on the mat», «cat» є підметом, і його залежність позначається як «nsubj».

«pobj» у лінгвістиці означає об’єкт передбачення (prepositional object). У залежній структурі речення «pobj» вказує на те, що дане слово є об’єктом після прийменника у реченні. Наприклад, у реченні «The cat sat on the mat», «mat» є об’єктом передбачення після прийменника «on», і його залежність позначається як «pobj».

«ROOT» у залежній структурі речення вказує на кореневий вузол або кореневе слово речення. Це слово не має жодного граматичного або синтаксичного батьківського вузла. У залежній структурі дерева «ROOT» є початковим вузлом, від якого виходять всі інші залежності у реченні. Наприклад, у реченні «The cat sat on the mat», слово «sat» може бути позначене як «ROOT», оскільки воно є кореневим словом, від якого залежать інші слова у реченні.

«prep» у залежній структурі речення вказує на прийменникову фразу (prepositional phrase). У таких фразах прийменник виступає як головне слово, а додаткові слова, які йдуть після прийменника, є його залежними. Наприклад, у

реченні «The cat sat on the mat», «on» є прийменником, а «the mat» є прийменниковою фразою, де «the» – артикль, а «mat» – об’єкт після прийменника. Таким чином, в залежній структурі ця залежність позначається як «prep».

«det» у залежній структурі речення означає детермінант (determiner). У граматиці англійської мови детермінант – це слово, яке передує іншому слову і вказує на його кількість або належність. Наприклад, у реченні «The cat sat on the mat», «The» є детермінантом, який вказує на те, що «cat» належить до конкретного поняття (визначений артикль). Отже, у залежній структурі ця залежність позначається як «det».

Лабораторна робота № 4
Створення граматики власне складових
Теоретичні відомості

Створення граматики власне складових (Directly Constituent Grammar) передбачає визначення правил, які описують структуру речень мови саме на рівні слів (або складових), а не на рівні фраз або конституентів. Основні принципи створення такої граматики містять:

1. Визначення складових: визначення базових складових, із яких будуються речення у мові. Ці складові можуть бути окремими словами або комбінаціями слів, які не поділяються на більші конституенти в межах граматики.

2. Формулювання правил: створення правил, які описують можливі комбінації складових для створення речень. Ці правила мають відображати синтаксичні обмеження мови і визначати, які комбінації складових є валідними.

3. Урахування синтаксичних відношень: врахування синтаксичних відношень між складовими, таких як суб'єкт–дієслово, іменник–прикметник тощо. Правила граматики повинні відображати можливі синтаксичні конфігурації, які відповідають природній мові.

4. Рекурсивність: деякі граматики власне складових можуть дозволяти рекурсивні структури, коли складові можуть комбінуватися з іншими складовими, утворюючи більш складні структури.

5. Узагальнення: визначення правил, які дозволяють узагальнювати шаблони для структур, щоб охопити різноманітність речень мови.

6. Перевірка на коректність: перевірка граматики на коректність і повноту, щоб переконатися, що вона відображає всі синтаксичні конструкції мови і не допускає амбігвітетів. Амбігвітети – це ситуації, коли фраза або вислів мають більше ніж одне можливе значення чи інтерпретацію. У лінгвістиці амбігвітет виникає, коли одне й те саме слово чи конструкція може мати декілька різних семантичних і синтаксичних інтерпретацій у контексті. Наприклад: «Вона побачила кішку на столі». Це речення може мати дві можливі інтерпретації: «Вона побачила кішку, яка була на столі» або «Вона побачила кішку, коли сама була на столі».

Амбігвітети можуть виникати через полісемію (коли слово має декілька значень) або у випадку, коли синтаксичні конструкції можуть мати різні інтерпретації в залежності від контексту. У лінгвістиці та обробці природної мови розв'язання амбігвітетів є важливим завданням для точного розуміння тексту та побудови правильних синтаксичних та семантичних моделей.

Граматика власне складових може бути корисною для деяких застосувань, особливо коли важливо визначити структуру речень на основі окремих слів без

утворення більших конституентів. Однак вона може бути менш ефективною для деяких задач, оскільки не враховує багаторівневу структуру мови.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень понять про створення граматики власне складових.

Завдання лабораторної роботи

1. Проведення досліджень граматики власне складових.

2. Ручне трасування генерування мовної конструкції на підставі граматики власне складових.

3. Обробка результатів та їхнє графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджується граматика власне складових.

2. Реалізація граматики власне складових є триступеневою процедурою.

– Перший ступінь – створення множини термінальних символів.

– Другий ступінь – створення множини нетермінальних символів та визначення початкового нетермінального символу.

– Третій ступінь – створення множини правил.

Вибір структури пропозиції з табл. 4.1 – відповідає номеру варіанта.

Таблиця 4.1

Індивідуальні завдання

№ варіанта	Валентність базової предикатної синтаксеми	Тип другої предикатної синтаксеми
1	3	адвербальна
2	4	адвербальна
3	5	адвербальна
4	6	адвербальна
5	7	адвербальна
6	3	модальна
7	4	модальна
8	5	модальна
9	6	модальна
10	7	модальна
11	3	атрибутивна
12	4	атрибутивна
13	5	атрибутивна
14	6	атрибутивна
15	7	атрибутивна

Порядок оформлення лабораторної роботи

4.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис граматики власне складових.
2. Результати генерування мовної конструкції.
3. Аналіз закономірностей.

- Висновки.

4.2. Вступ має містити, окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

4.3. У звіті наводиться граматика власне складових.

4.4. У висновках наводиться порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

Приклад

Для речення «Сидоренко, наш кращий працівник, народився в місті Києві», БС-граматика буде мати вигляд:

$G=<N,\Sigma,P,S>$,

$N=\{NP, VP, PP, N, A, V, Prep\}$

$\Sigma=\{$Сидоренко, наш, кращий, працівник, народився, в, місті, Києві$\}$

$P=\{S \rightarrow NP\ VP$

$VP \rightarrow V\ PP$

$PP \rightarrow Prep\ NP$

$NP \rightarrow N\ NP \mid A\ NP \mid A\ N \mid N\ N$

$N \rightarrow$ Сидоренко | працівник | місті | Києві

$A \rightarrow$ наш | кращий

$V \rightarrow$ народився

$Prep \rightarrow$ в$\}$

Listing 4.1

```
!pip install nltk
!pip install pymorphy2
!pip install pymorphy2-dicts-uk
import nltk
import pymorphy2
# викачка для пунктуації
nltk.download('punkt')
```

```
## токенізація
sentence = "Сидоренко наш кращий працівник народився в місті Києві"
tokens = nltk.word_tokenize(sentence)
## морфологічний аналіз для української мови
ma=pymorphy2.MorphAnalyzer(lang='uk')
rule = """
S -> NP VP
VP -> VERB PP
PP -> PREP NP
NP -> NOUN NP | A NP | NOUN NOUN | A NOUN
"""
for t in tokens:
  gr=ma.parse(t)[0]
  pos = gr.tag.POS
  if (pos=="INTJ"):
     pos = "PREP"
  if (pos=="ADJF" or pos=="ADJS" or pos=="NPRO"):
     pos = "A"
  rule = rule + pos + """ -> """ + """""" + t + """""" + """\n"""
print(rule)
# S -> NP VP
# VP -> VERB PP
# PP -> PREP NP
# NP -> NOUN NP | A NP | NOUN NOUN | A NOUN
# NOUN -> 'Сидоренко'
# A -> 'наш'
# A -> 'кращий'
# NOUN -> 'працівник'
# VERB -> 'народився'
# PREP -> 'в'
# NOUN -> 'місті'
# NOUN -> 'Києві'
## синтаксичний аналіз на основі БС-граматики і динамічного програмування
grammar = nltk.CFG.fromstring(rule)
chat_parser = nltk.ChartParser(grammar)
for tree in chat_parser.parse(tokens):
    print(tree)
# (S (NP (NOUN Сидоренко) (NP (A наш) (NP (A кращий) (NOUN працівник)))) (VP (VERB
народився) (PP (PREP в) (NP (NOUN місті) (NOUN Києві)))))
```

Listing 4.2

```
import nltk

# Вхідне речення для аналізу
sentence = "The quick brown fox jumps over the lazy dog."
```

```
# Розділити речення на слова
words = nltk.word_tokenize(sentence)
# Виконати частиномовний аналіз
tagged_words = nltk.pos_tag(words)
# Вивести слова з тегами частин мови
print("Words with Part-of-Speech Tags:")
print(tagged_words)
# Визначити граматику власне складових
def immediate_constituents(words, tagged_words):
    constituents = []
    for i in range(len(tagged_words)):
        if i < len(tagged_words) - 1:
            constituents.append((tagged_words[i][0], tagged_words[i+1][0]))
    return constituents
# Вивести власне складові
print("\nImmediate Constituents:")
constituents = immediate_constituents(words, tagged_words)
for constituent in constituents:
    print(constituent)
```

У цьому Лістингу результат роботи програми має такий вигляд:

```
Words with Part-of-Speech Tags:
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'),
('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]
Immediate Constituents:
('The', 'quick')
('quick', 'brown')
('brown', 'fox')
('fox', 'jumps')
('jumps', 'over')
('over', 'the')
('the', 'lazy')
('lazy', 'dog')
('dog', '.')
```

Тут Immediate Constituents перекладається як *власне складові*.

Лабораторна робота № 5
Створення мережі переходів
Теоретичні відомості

Мережа переходів у граматиці (Transition Network) – це структура даних, яка використовується для опису синтаксичних правил та обробки тексту в комп'ютерних моделях мови. Основні принципи створення такої мережі містять:

1. Визначення станів: кожен вузол у мережі відповідає певному синтаксичному стану, який може виникати під час аналізу тексту. Наприклад, це може бути стан, в якому частина мови вже була розпізнана, аналізована та відповідним чином врахована.

2. Визначення переходів: дуги (або ребра) у мережі вказують на можливі переходи між різними синтаксичними станами. Кожен перехід може бути спричинений певною синтаксичною структурою в тексті або специфічною граматичною конструкцією.

3. Урахування синтаксичних правил: кожен перехід у мережі пов'язаний із синтаксичним правилом, яке визначає умови, за яких перехід може відбутися. Ці правила визначаються на основі граматики мови та інших синтаксичних правил.

4. Управління контекстом: мережа переходів може містити механізми для управління контекстом, наприклад, збереження та відновлення інформації про контекст аналізу для визначення правильних переходів.

5. Реакція на вхід: кожен перехід може мати свої умови виконання, які визначаються наявністю певних елементів в тексті. Ці умови виконання можуть об'єднувати семантичні, морфологічні та синтаксичні характеристики.

6. Реалізація в програмному коді: мережа переходів може бути реалізована у вигляді структури даних та алгоритмів, які дозволяють виконувати аналіз тексту та визначати його синтаксичну структуру.

Створення мережі переходів у граматиці – це складний процес, який вимагає ретельного аналізу мови та розробки ефективних алгоритмів для обробки тексту. Вона може бути використана у багатьох застосунках, включаючи машинний переклад, розпізнавання мови та автоматичну обробку тексту.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень понять про створення мережі переходів.

Завдання лабораторної роботи

1. Проведення досліджень мережі переходів.

2. Ручне трасування генерування мовної конструкції на підставі мережі переходів граматики.

3. Обробка результатів та їхнє графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджується мережа переходів.

2. Реалізація мережі переходів має вигляд триступеневої процедури.

– Перший ступінь – створення множини станів.

– Другий ступінь – створення множини дуг з мітками лексичних категорій.

– Третій ступінь – створення множини дуг з мітками найменування станів.

Вибір структури пропозиції з табл. 5.1 – відповідає номеру варіанта.

Таблиця 5.1

Індивідуальні завдання

№ варіанта	Валентність базової предикатної синтаксеми	Тип другої предикатної синтаксеми
1	3	адвербальна
2	4	адвербальна
3	5	адвербальна
4	6	адвербальна
5	7	адвербальна
6	3	модальна
7	4	модальна
8	5	модальна
9	6	модальна
10	7	модальна
11	3	атрибутивна
12	4	атрибутивна
13	5	атрибутивна
14	6	атрибутивна
15	7	атрибутивна

Порядок оформлення лабораторної роботи

5.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис мережі переходів.
2. Результати генерування мовної конструкції.
3. Аналіз закономірностей.

- Висновки.

5.2. Вступ має містити, окрім самостійної характеристикою теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

5.3. У звіті наводяться мережі переходів.

5.4. У висновках наводиться порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

Приклад

Речення «Буде оператор управляти агрегатом?» може бути описане множиною спрямованих підграфів (мереж переходів) із кінцевим числом станів і позначеними дугами (рис. 5.1)

Серед станів виділяються:

– початкова (наприклад, S_0);

– множина проміжних (наприклад, $S_1, S_2, S_3, S_4, S_8, S_9$);

– множина кінцевих станів (наприклад, $S_{5/1}, S_{6/1}, S_{7/1}, S_{10/1}$).

Мітки на дугах можуть бути:

– лексичними категоріями (наприклад, V – дієслово, MV – модальне дієслово, N – іменник, Par – частка, A – прикметник, Pred – прийменник);

– найменуваннями станів (наприклад, PP – прийменникова група іменників, VP – група дієслова, NP – група іменника).

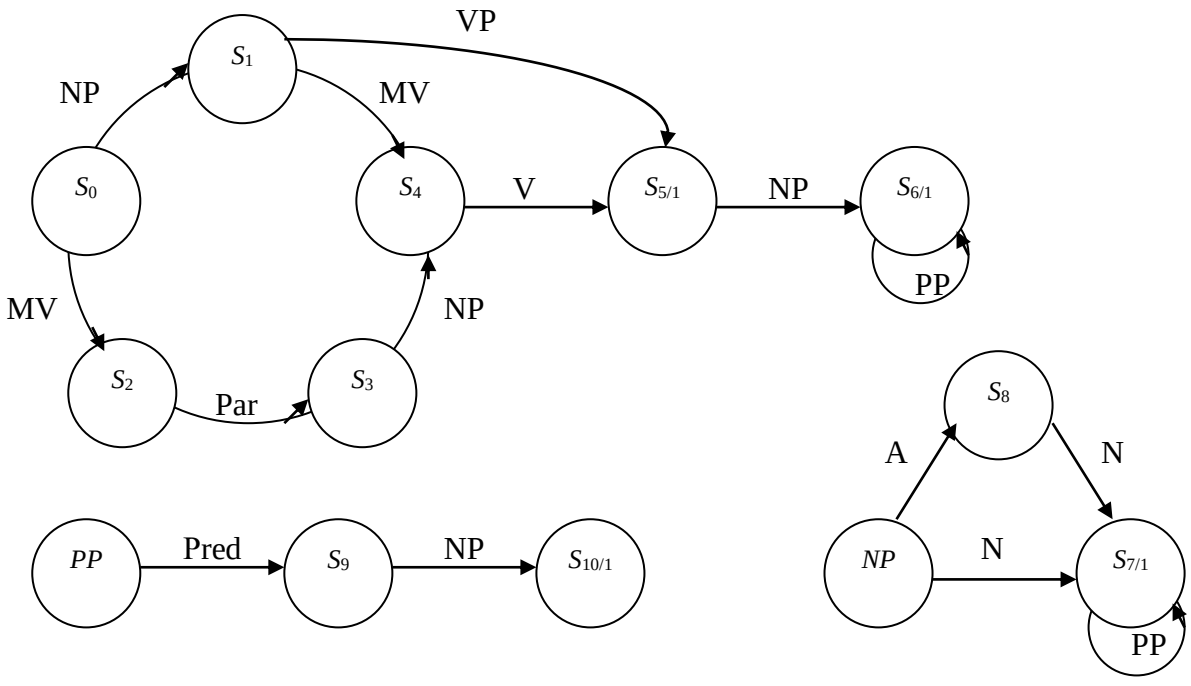

Рис. 5.1. Подання граматики у вигляді мережі переходів

Listing 5.1

```
import networkx as nx
import matplotlib.pyplot as plt

# Функція для створення граматичної мережі
def create_grammar_network(grammar):
    G = nx.DiGraph()
    # Додаємо ребра від початкового символу граматики до правил
    for non_terminal, rules in grammar.items():
        for rule in rules:
            G.add_edge(non_terminal, rule)
    return G
# Функція для візуалізації мережі
def visualize_grammar_network(G):
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_size=2000, node_color="skyblue", font_size=10,
font_weight="bold")
    plt.show()
# Приклад використання
grammar = {
    "<S>": ["<NP> <VP>", "<S> <CONJ> <S>"],
    "<NP>": ["<Det> <N>", "<N>"],
    "<VP>": ["<V>", "<V> <NP>"],
    "<Det>": ["the", "a"],
    "<N>": ["cat", "dog", "man", "woman"],
    "<V>": ["sings", "runs", "jumps"],
    "<CONJ>": ["and", "but", "or"]
}
#The man and woman sings but a dog or cat runs and jumps
grammar_network = create_grammar_network(grammar)
visualize_grammar_network(grammar_network)
```

На рис. 5.1 зображена мережа переходів, у якій введено ряд позначень, що розшифровуються так:

що означає <S>?

У лінгвістиці, <S> – це скорочення, що позначає синтаксичну категорію, яка відповідає за речення (sentence) або фразу, яка має структуру речення. У контексті мовного аналізу <S> використовується для позначення синтаксичного складу речення або фрази, в яких розглядається порядок слів, граматична правильність тощо. Наприклад, у реченні «Сонце сходить на сході», фраза «Сонце сходить» може бути позначена як <S>, це означає, що вона складається з окремого синтаксичного елементу, або є реченням.

що означає <det>, <conj>?

<det> – це скорочення, яке часто використовується в контексті лінгвістики для позначення визначеного артикля, тобто вказівки на конкретний предмет або особу в мовленні. Наприклад, у реченні «The cat is on the mat» слово «the» є визначеним артиклем, і його можна позначити як <det>.

<conj> – це скорочення, яке використовується в мовному аналізі для позначення сполучника. Сполучник – це частина мови, яка використовується для з'єднання двох або більше слів, фраз або речень у складну конструкцію. Наприклад, у реченні «Іван і Марія прийшли на вечірку», слово «і» є сполучником, і його можна позначити як <conj>.

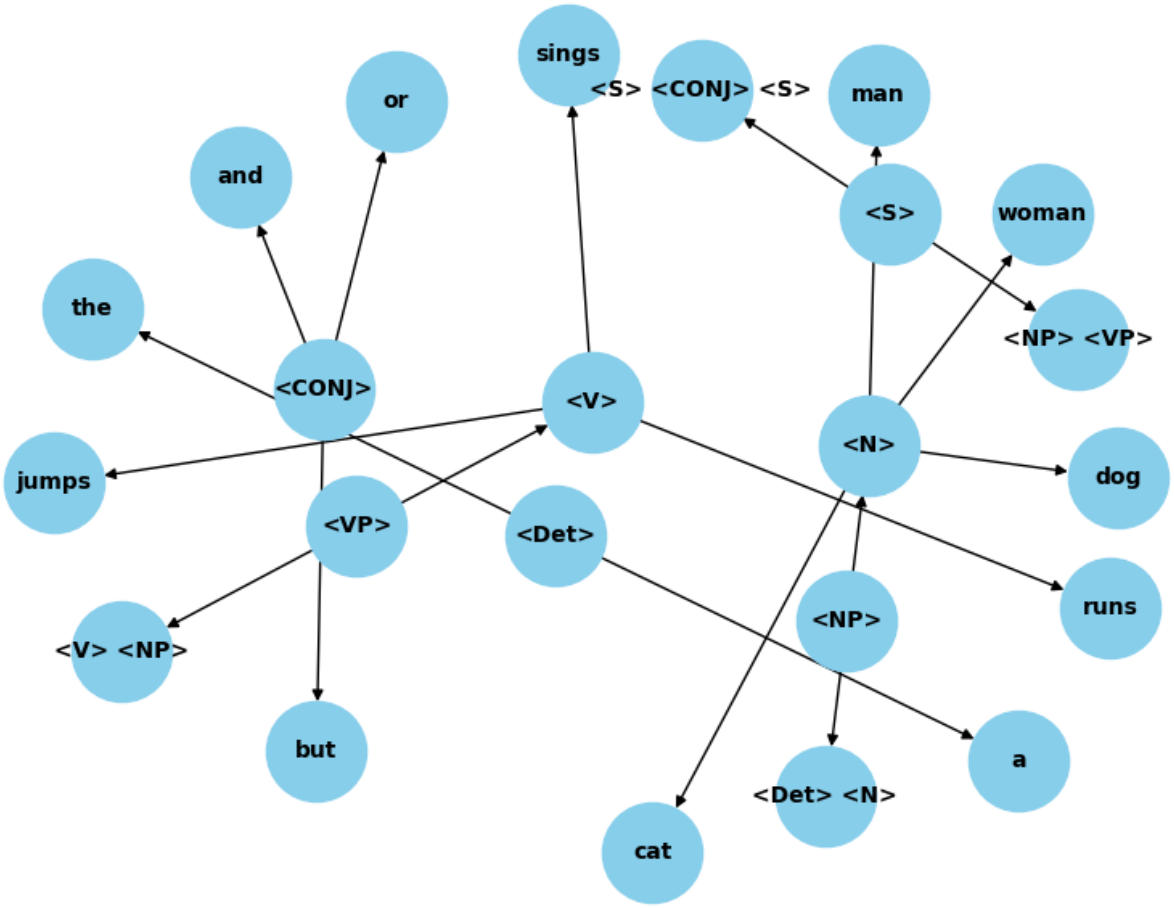

Рис. 5.2. Графічний вигляд мережі переходів

що означає <S>?

У лінгвістиці, <S> – це скорочення, що позначає синтаксичну категорію, яка відповідає за речення (sentence) або фразу, яка має структуру речення. У контексті мовного аналізу <S> використовується для позначення синтаксичного складу речення або фрази, в яких розглядається порядок слів, граматична правильність тощо. Наприклад, у реченні «Сонце сходить на сході», фраза «Сонце сходить»

може бути позначена як <S>, це означає, що вона складається з окремого синтаксичного елементу, або є реченням.

що означає <det>, <conj>?

<det> – це скорочення, яке часто використовується в контексті лінгвістики для позначення визначеного артикля, тобто вказівки на конкретний предмет або особу в мовленні. Наприклад, у реченні «The cat is on the mat» слово «the» є визначеним артиклем, і його можна позначити як <det>.

<conj> – це скорочення, яке використовується в мовному аналізі для позначення сполучника. Сполучник – це частина мови, яка використовується для з'єднання двох або більше слів, фраз або речень у складну конструкцію. Наприклад, у реченні «Іван і Марія прийшли на вечірку», слово «і» є сполучником, і його можна позначити як <conj>.

Лабораторна робота № 6
Обчислення частоти та ймовірності появи уніграм, біграм, триграм і поточкової спільної інформації
Теоретичні відомості

Принцип обчислення частоти та ймовірності появи уніграм, біграм та триграм часто використовується в обробці природної мови та у статистичному машинному навчанні для аналізу текстів.

1. Уніграми (unigrams): це окремі слова або токени, що складають текст. Щоб обчислити частоту появи уніграми, просто порахуйте кількість входжень кожного слова в текст.

Наприклад, у тексті «The cat sat on the mat», уніграми та їхні частоти будуть:

- «the»: 2
- «cat»: 1
- «sat»: 1
- «on»: 1
- «mat»: 1

2. Біграми (bigrams): це пари послідовних слів у тексті. Для обчислення частоти біграми, вираховуйте кількість разів, коли певна пара слів зустрічається поруч.

У тому ж тексті «The cat sat on the mat», біграми та їхні частоти будуть:

- «the cat»: 1
- «cat sat»: 1
- «sat on»: 1
- «on the»: 1
- «the mat»: 1

3. Триграми (trigrams): це трійки послідовних слів у тексті. Для обчислення частоти триграми, вираховуйте кількість разів, коли певна трійка слів зустрічається поруч.

Для того ж тексту «The cat sat on the mat», триграми та їхні частоти будуть:

- «the cat sat»: 1
- «cat sat on»: 1
- «sat on the»: 1
- «on the mat»: 1

Ймовірність появи випадкової послідовності слів (уніграм, біграм або триграм) можна обчислити, розділивши частоту входження цієї послідовності на загальну кількість послідовностей у тексті. Наприклад, для уніграми «the» вираховується ймовірність як кількість входжень «the» поділена на загальну кількість уніграм у тексті.

Обчислення ймовірностей появи уніграм, біграм та триграм має кілька важливих застосувань:

1. Мовне моделювання: це одне з основних застосувань. Мовні моделі використовуються для прогнозування наступного слова в послідовності слів. Наприклад, на основі ймовірностей біграм та триграм можна прогнозувати, яке слово має найбільшу ймовірність з'явитися після певної послідовності слів.

2. Автозавершення тексту та прогнозування наступного слова: на основі використання ймовірностей біграм та триграм можна створювати функціонал автозавершення тексту, який пропонує користувачеві можливі варіанти наступного слова або навіть завершує фразу.

3. Виправлення помилок: використання ймовірностей біграм та триграм може допомогти в розпізнаванні та виправленні помилок у тексті. Модель може враховувати ймовірність виникнення певної послідовності слів та використовувати цю інформацію для виправлення помилок у введенні тексту.

4. Машинний переклад: у машинному перекладі ймовірності біграм та триграм можуть використовуватися для вибору найбільш вірогідного перекладу фрази чи речення на основі величини ймовірності послідовностей слів.

5. Виділення ключових фраз та тематичний аналіз: аналізуючи ймовірності біграм та триграм, можна виявити ключові фрази або теми у тексті. Це може бути корисним для аналізу текстових даних та для знаходження суттєвої інформації.

Отже, обчислення ймовірностей появи уніграм, біграм та триграм є важливим етапом для багатьох завдань у обробці природної мови та у статистичному машинному навчанні.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень подань про уніграми, біграми, триграми та поточкову спільну інформацію.

Завдання лабораторної роботи

1. Проведення досліджень уніграм, біграм, триграм і поточкової спільної інформації.

2. Ручне обчислення частоти та ймовірності появи уніграм, біграм, триграм і поточкової спільної інформації.

3. Обробка результатів та їхнє графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджуються уніграми, біграми, триграми та поточкова спільна інформація.

2. Реалізація обчислення частоти та ймовірності появи уніграм, біграм, триграм і поточкової спільної інформації є чотириступеневою процедурою.

– Перший ступінь – обчислення частоти та ймовірності появи уніграм.

– Другий ступінь – обчислення частоти та ймовірності появи біграм.

– Третій ступінь – обчислення частоти та ймовірності появи триграм.

– Четвертий ступінь – обчислення поточкової спільної інформації.

Вибір пропозиції та друк інформації про появу грам (частоти або ймовірності) з табл. 6.1 – відповідає номеру варіанта.

Таблиця 6.1

Індивідуальні завдання

№ варіанта	Пропозиції	Частота / ймовірність
1	гарні кекси коштують $3.88\ny Нью-Йорку гарні цукерки коштують $5.35\ny Нью-Йорку	частота
2	гарні гамбургери коштують $5.88\ny Нью-Йорку гарні чіпси коштують $4.35\ny Нью-Йорку	частота
3	гарна кола коштує $4.88\ny Нью-Йорку гарна пепсі коштує $4.88\ny Нью-Йорку	частота
4	гарна піца коштує $6.25\ny Нью-Йорку гарна паста коштує $7.15\ny Нью-Йорку	частота
5	смачні кекси коштують $3.88\ny Лос-Анджелесі смачні цукерки коштують $5.35\ny Лос-Анджелесі	частота
6	смачні гамбургери коштують $3.88\ny Лос-Анджелесі смачні чіпси коштують $5.35\ny Лос-Анджелесі	частота
7	смачна кола коштує $4.88\ny Лос-Анджелесі смачна пепсі коштує $4.88\ny Лос-Анджелесі	частота
8	смачна піца коштує $6.25\ny Лос-Анджелесі смачна паста коштує $7.15\ny Лос-Анджелесі	частота
9	гарні кекси коштують $3.88\ny Нью-Йорку гарні цукерки коштують $5.35\ny Нью-Йорку	ймовірність
10	гарні гамбургери коштують $5.88\ny Нью-Йорку гарні чіпси коштують $4.35\ny Нью-Йорку	ймовірність
11	гарна кола коштує $4.88\ny Нью-Йорку гарна пепсі коштує $4.88\ny Нью-Йорку	ймовірність
12	гарна піца коштує $6.25\ny Нью-Йорку гарна паста коштує $7.15\ny Нью-Йорку	ймовірність
13	смачні кекси коштують $3.88\ny Лос-Анджелесі смачні цукерки коштують $5.35\ny Лос-Анджелесі	ймовірність
14	смачні гамбургери коштують $3.88\ny Лос-Анджелесі смачні чіпси коштують $5.35\ny Лос-Анджелесі	ймовірність
15	смачна кола коштує $4.88\ny Лос-Анджелесі смачна пепсі коштує $4.88\ny Лос-Анджелесі	ймовірність
16	смачна піца коштує $6.25\ny Лос-Анджелесі смачна паста коштує $7.15\ny Лос-Анджелесі	ймовірність

Порядок оформлення лабораторної роботи

6.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис обчислення частоти та ймовірності появи уніграм, біграм, триграм і поточкової спільної інформації.

2. Результати обчислення частоти та ймовірності появи уніграм, біграм, триграм і поточкової спільної інформації.

3. Аналіз закономірностей.

- Висновки.

6.1. Вступ має містити, окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

6.2. У звіті наводяться обчислення частоти та ймовірності появи уніграм, біграм, триграм і поточкової спільної інформації.

6.3. У висновках наводиться порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

У наведених далі Лістингах запрограмовані різні варіанти аналізу уніграм, біграм та триграм.

Listing 6.1

```
!pip install nltk
import nltk
import math
# викачування для пунктуації
nltk.download('punkt')
s = "гарні кекси коштують $3.88\ny Нью-Йорку гарні цукерки коштують $5.35\ny Нью-Йорку"
# розбиття тексту на токени
token = nltk.tokenize.word_tokenize(text=s)
print(token)
# ['гарні', 'кекси', 'коштують', '$', '3.88', 'у', 'Нью-Йорку', 'гарні', 'цукерки', 'коштують', '$', '5.35', 'у', 'Нью-Йорку']
# обчислення частоти появи уніграм
unigram_fd = nltk.FreqDist(token)
u = unigram_fd.most_common()
print(u)
# [('гарні', 2), ('коштують', 2), ('$', 2), ('у', 2), ('Нью-Йорку', 2), ('кекси', 1), ('3.88', 1), ('цукерки', 1), ('5.35', 1)]
# обчислення ймовірності появи уніграм
```

```
pu = [(l, unigram_fd.freq(l)) for l in nltk.FreqDist(token)]
print(pu)
# [('гарні', 0.14285714285714285), ('коштують', 0.14285714285714285), ('$',
0.14285714285714285), ('у', 0.14285714285714285), ('Нью-Йорку', 0.14285714285714285),
('кекси', 0.07142857142857142), ('3.88', 0.07142857142857142), ('цукерки',
0.07142857142857142), ('5.35', 0.07142857142857142)]
# обчислення частоти появи біграм
bigram_fd = nltk.FreqDist(nltk.bigrams(token))
b = bigram_fd.most_common()
print(b)
# [(('коштують', '$'), 2), (('у', 'Нью-Йорку'), 2), (('гарні', 'кекси'), 1), (('кекси', 'коштують'), 1),
(('$', '3.88'), 1), (('3.88', 'у'), 1), (('Нью-Йорку', 'гарні'), 1), (('гарні', 'цукерки'), 1), (('цукерки',
'коштують'), 1), (('$', '5.35'), 1), (('5.35', 'у'), 1)]
# обчислення ймовірності появи біграм
pb = [(l, bigram_fd.freq(l)) for l in nltk.FreqDist(nltk.bigrams(token))]
print(pb)
[(('коштують', '$'), 0.15384615384615385), (('у', 'Нью-Йорку'), 0.15384615384615385), (('гарні',
'кекси'), 0.07692307692307693), (('кекси', 'коштують'), 0.07692307692307693), (('$', '3.88'),
0.07692307692307693), (('3.88', 'у'), 0.07692307692307693), (('Нью-Йорку', 'гарні'),
0.07692307692307693), (('гарні', 'цукерки'), 0.07692307692307693), (('цукерки', 'коштують'),
0.07692307692307693), (('$', '5.35'), 0.07692307692307693), (('5.35', 'у'), 0.07692307692307693)]
# обчислення частоти появи триграм
trigram_fd = nltk.FreqDist(nltk.trigrams(token))
t = trigram_fd.most_common()
print(t)
# [(('гарні', 'кекси', 'коштують'), 1), (('кекси', 'коштують', '$'), 1), (('коштують', '$', '3.88'), 1),
(('$', '3.88', 'у'), 1), (('3.88', 'у', 'Нью-Йорку'), 1), (('у', 'Нью-Йорку', 'гарні'), 1), (('Нью-Йорку',
'гарні', 'цукерки'), 1), (('гарні', 'цукерки', 'коштують'), 1), (('цукерки', 'коштують', '$'), 1),
(('коштують', '$', '5.35'), 1), (('$', '5.35', 'у'), 1), (('5.35', 'у', 'Нью-Йорку'), 1)]
# обчислення ймовірності появи триграм
pt = [(l, trigram_fd.freq(l)) for l in nltk.FreqDist(nltk.trigrams(token))]
print(pt)
# [(('гарні', 'кекси', 'коштують'), 0.08333333333333333), (('кекси', 'коштують', '$'),
0.08333333333333333), (('коштують', '$', '3.88'), 0.08333333333333333), (('$', '3.88', 'у'),
0.08333333333333333), (('3.88', 'у', 'Нью-Йорку'), 0.08333333333333333), (('у', 'Нью-Йорку',
'гарні'), 0.08333333333333333), (('Нью-Йорку', 'гарні', 'цукерки'), 0.08333333333333333),
(('гарні', 'цукерки', 'коштують'), 0.08333333333333333), (('цукерки', 'коштують', '$'),
0.08333333333333333), (('коштують', '$', '5.35'), 0.08333333333333333), (('$', '5.35', 'у'),
0.08333333333333333), (('5.35', 'у', 'Нью-Йорку'), 0.08333333333333333)]
# обчислення поточкової взаємної інформації
pmi = b
for j in range(len(b)):
    for i in range(len(u)):
        for k in range(len(u)):
            if (u[i][0] == b[j][0][1] and u[k][0] == b[j][0][0]):
                pmi[j] = (b[j][0], math.log((pb[j][1])/((pu[i][1])*(pu[k][1]))))
```

```
print(pmi)
# [(('коштують', '$'), 2.020018121209035), (('у', 'Нью-Йорку'), 2.020018121209035), (('гарні', 'кекси'), 2.020018121209035), (('кекси', 'коштують'), 2.020018121209035), (('$', '3.88'), 2.020018121209035), (('3.88', 'у'), 2.020018121209035), (('Нью-Йорку', 'гарні'), 1.32687094064909), (('гарні', 'цукерки'), 2.020018121209035), (('цукерки', 'коштують'), 2.020018121209035), (('$', '5.35'), 2.020018121209035), (('5.35', 'у'), 2.020018121209035)]]
```

Listing 6.2

```
import nltk
from nltk.util import ngrams
from collections import Counter
from nltk.probability import FreqDist
# Зразок тексту
text = "This is a sample text for calculating unigrams, bigrams, trigrams, and pointwise mutual information."
# Розбиття тексту на токени (слова)
tokens = nltk.word_tokenize(text.lower())
# Уніграми
unigrams = list(ngrams(tokens, 1))
# Біграми
bigrams = list(ngrams(tokens, 2))
# Триграми
trigrams = list(ngrams(tokens, 3))
# Обчислення частоти уніграм, біграм, триграм
unigram_freq = FreqDist(unigrams)
bigram_freq = FreqDist(bigrams)
trigram_freq = FreqDist(trigrams)
# Виведення результатів
print("Unigram frequencies:")
print(unigram_freq.most_common())
print("\nBigram frequencies:")
print(bigram_freq.most_common())
print("\nTrigram frequencies:")
print(trigram_freq.most_common())
```

Listing 6.3

```
nltk.download('stopwords')
# Зразок тексту
text = """
Python is an easy to learn, powerful programming language. It has efficient high-level data structures
and a simple but effective approach to object-oriented programming.
Python's elegant syntax and dynamic typing, together with its interpreted nature,
make it an ideal language for scripting and rapid application development in many areas on most platforms.
```

```
Python is also suitable as an extension language for customizable applications.
"""
# Розділення тексту на токени
tokens = nltk.word_tokenize(text.lower())
# Видалення знаків пунктуації та стоп-слів
stop_words = set(stopwords.words('english'))
tokens = [token for token in tokens if token.isalnum() and token not in stop_words]
# Обчислення уніграм
unigrams_freq = FreqDist(tokens)
total_unigrams = len(tokens)
# Виведення уніграм та їхньої частоти
print("Unigrams:")
for word, freq in unigrams_freq.items():
    print(f"{word}: {freq} (Probability: {freq / total_unigrams})")
# Обчислення біграм
bigrams = list(ngrams(tokens, 2))
bigrams_freq = Counter(bigrams)
total_bigrams = len(bigrams)
# Виведення біграм та їхньої частоти
print("\nBigrams:")
for bigram, freq in bigrams_freq.items():
    print(f"{' '.join(bigram)}: {freq} (Probability: {freq / total_bigrams})")
# Обчислення триграм
trigrams = list(ngrams(tokens, 3))
trigrams_freq = Counter(trigrams)
total_trigrams = len(trigrams)
# Виведення триграм та їхньої частоти
print("\nTrigrams:")
for trigram, freq in trigrams_freq.items():
    print(f"{' '.join(trigram)}: {freq} (Probability: {freq / total_trigrams})")
# Обчислення поточкової спільної інформації (pointwise mutual information)
def pmi(word1, word2, bigram_freq, unigram_freq, total_bigrams):
    p_word1 = unigram_freq[word1] / total_unigrams
    p_word2 = unigram_freq[word2] / total_unigrams
    p_bigram = bigram_freq[(word1, word2)] / total_bigrams
    return math.log2(p_bigram / (p_word1 * p_word2))
# Виведення поточкової спільної інформації для певних біграм
print("\nPointwise Mutual Information:")
print("PMI('powerful', 'programming'):", pmi('powerful', 'programming', bigrams_freq,
unigrams_freq, total_bigrams))
print("PMI('dynamic', 'typing'):", pmi('dynamic', 'typing', bigrams_freq, unigrams_freq,
total_bigrams))
```

Результат роботи програми такий:

Unigrams:

python: 3 (Probability: 0.07894736842105263)

easy: 1 (Probability: 0.02631578947368421)

learn: 1 (Probability: 0.02631578947368421)
powerful: 1 (Probability: 0.02631578947368421)
programming: 2 (Probability: 0.05263157894736842)
language: 3 (Probability: 0.07894736842105263)
efficient: 1 (Probability: 0.02631578947368421)
data: 1 (Probability: 0.02631578947368421)
structures: 1 (Probability: 0.02631578947368421)
simple: 1 (Probability: 0.02631578947368421)
effective: 1 (Probability: 0.02631578947368421)
approach: 1 (Probability: 0.02631578947368421)
elegant: 1 (Probability: 0.02631578947368421)
syntax: 1 (Probability: 0.02631578947368421)
dynamic: 1 (Probability: 0.02631578947368421)
typing: 1 (Probability: 0.02631578947368421)
together: 1 (Probability: 0.02631578947368421)
interpreted: 1 (Probability: 0.02631578947368421)
nature: 1 (Probability: 0.02631578947368421)
make: 1 (Probability: 0.02631578947368421)
ideal: 1 (Probability: 0.02631578947368421)
scripting: 1 (Probability: 0.02631578947368421)
rapid: 1 (Probability: 0.02631578947368421)
application: 1 (Probability: 0.02631578947368421)
development: 1 (Probability: 0.02631578947368421)
many: 1 (Probability: 0.02631578947368421)
areas: 1 (Probability: 0.02631578947368421)
platforms: 1 (Probability: 0.02631578947368421)
also: 1 (Probability: 0.02631578947368421)
suitable: 1 (Probability: 0.02631578947368421)
extension: 1 (Probability: 0.02631578947368421)
customizable: 1 (Probability: 0.02631578947368421)
applications: 1 (Probability: 0.02631578947368421)
Bigrams:
python easy: 1 (Probability: 0.02702702702702703)
easy learn: 1 (Probability: 0.02702702702702703)
learn powerful: 1 (Probability: 0.02702702702702703)
powerful programming: 1 (Probability: 0.02702702702702703)
programming language: 1 (Probability: 0.02702702702702703)
language efficient: 1 (Probability: 0.02702702702702703)
efficient data: 1 (Probability: 0.02702702702702703)
data structures: 1 (Probability: 0.02702702702702703)
structures simple: 1 (Probability: 0.02702702702702703)
simple effective: 1 (Probability: 0.02702702702702703)
effective approach: 1 (Probability: 0.02702702702702703)
approach programming: 1 (Probability: 0.02702702702702703)
programming python: 1 (Probability: 0.02702702702702703)
python elegant: 1 (Probability: 0.02702702702702703)

elegant syntax: 1 (Probability: 0.02702702702702703)
syntax dynamic: 1 (Probability: 0.02702702702702703)
dynamic typing: 1 (Probability: 0.02702702702702703)
typing together: 1 (Probability: 0.02702702702702703)
together interpreted: 1 (Probability: 0.02702702702702703)
interpreted nature: 1 (Probability: 0.02702702702702703)
nature make: 1 (Probability: 0.02702702702702703)
make ideal: 1 (Probability: 0.02702702702702703)
ideal language: 1 (Probability: 0.02702702702702703)
language scripting: 1 (Probability: 0.02702702702702703)
scripting rapid: 1 (Probability: 0.02702702702702703)
rapid application: 1 (Probability: 0.02702702702702703)
application development: 1 (Probability: 0.02702702702702703)
development many: 1 (Probability: 0.02702702702702703)
many areas: 1 (Probability: 0.02702702702702703)
areas platforms: 1 (Probability: 0.02702702702702703)
platforms python: 1 (Probability: 0.02702702702702703)
python also: 1 (Probability: 0.02702702702702703)
also suitable: 1 (Probability: 0.02702702702702703)
suitable extension: 1 (Probability: 0.02702702702702703)
extension language: 1 (Probability: 0.02702702702702703)
language customizable: 1 (Probability: 0.02702702702702703)
customizable applications: 1 (Probability: 0.02702702702702703)
Trigrams:
python easy learn: 1 (Probability: 0.027777777777777776)
easy learn powerful: 1 (Probability: 0.027777777777777776)
learn powerful programming: 1 (Probability: 0.027777777777777776)
powerful programming language: 1 (Probability: 0.027777777777777776)
programming language efficient: 1 (Probability: 0.027777777777777776)
language efficient data: 1 (Probability: 0.027777777777777776)
efficient data structures: 1 (Probability: 0.027777777777777776)
data structures simple: 1 (Probability: 0.027777777777777776)
structures simple effective: 1 (Probability: 0.027777777777777776)
simple effective approach: 1 (Probability: 0.027777777777777776)
effective approach programming: 1 (Probability: 0.027777777777777776)
approach programming python: 1 (Probability: 0.027777777777777776)
programming python elegant: 1 (Probability: 0.027777777777777776)
python elegant syntax: 1 (Probability: 0.027777777777777776)
elegant syntax dynamic: 1 (Probability: 0.027777777777777776)
syntax dynamic typing: 1 (Probability: 0.027777777777777776)
dynamic typing together: 1 (Probability: 0.027777777777777776)
typing together interpreted: 1 (Probability: 0.027777777777777776)
together interpreted nature: 1 (Probability: 0.027777777777777776)
interpreted nature make: 1 (Probability: 0.027777777777777776)
nature make ideal: 1 (Probability: 0.027777777777777776)
make ideal language: 1 (Probability: 0.027777777777777776)

ideal language scripting: 1 (Probability: 0.027777777777777776)
language scripting rapid: 1 (Probability: 0.027777777777777776)
scripting rapid application: 1 (Probability: 0.027777777777777776)
rapid application development: 1 (Probability: 0.027777777777777776)
application development many: 1 (Probability: 0.027777777777777776)
development many areas: 1 (Probability: 0.027777777777777776)
many areas platforms: 1 (Probability: 0.027777777777777776)
areas platforms python: 1 (Probability: 0.027777777777777776)
platforms python also: 1 (Probability: 0.027777777777777776)
python also suitable: 1 (Probability: 0.027777777777777776)
also suitable extension: 1 (Probability: 0.027777777777777776)
suitable extension language: 1 (Probability: 0.027777777777777776)
extension language customizable: 1 (Probability: 0.027777777777777776)
language customizable applications: 1 (Probability: 0.027777777777777776)
Pointwise Mutual Information:
PMI('powerful', 'programming'): 4.286401661258221
PMI('dynamic', 'typing'): 5.286401661258221

Проаналізуємо результати роботи програми Listing 6.3 та встановимо, що означає *Pointwise Mutual Information?*

Pointwise Mutual Information (PMI) є мірою, яка використовується для визначення статистичного зв'язку між двома подіями. Вона оцінює ймовірність спільної появи двох подій порівняно з їхніми індивідуальними виникненнями. У сутності PMI вказує на те, наскільки часто дві події відбуваються разом порівняно з тим, як часто вони відбуваються окремо.

Формула PMI для двох подій A та B виглядає так:

$$PMI(A,B) = log_2(\frac{P(A,B)}{P(A) \cdot P(B)}),$$

де $P(A, B)$ – ймовірність *одночасного* виникнення подій A і B,

$P(A)$ та $P(B)$ – *індивідуальні* ймовірності виникнення подій A і B.

Якщо PMI додатне, це означає, що дві події співвідносяться частіше, ніж випадково (тобто, вони синоніми або мають деякий взаємозв'язок). Якщо PMI від'ємне, це означає, що вони відбуваються незалежно одне від одного.

Лабораторна робота № 7
Переклад за допомогою нейромережевого транслятора
Теоретичні відомості

Переклад за допомогою нейромережевого транслятора ґрунтується на використанні штучних нейромереж, щоб автоматично перекладати текст або мовлення з однієї мови на іншу. Основним принципом є використання великого обсягу даних для навчання моделі перекладу. Нейромережеві транслятори використовуються для автоматизації процесу перекладу, з метою зменшення залежності від ручного втручання та задля покращення якості перекладу. Такі системи постійно вдосконалюються за рахунок вивчення нових даних та за умови використання різних технік оптимізації, наприклад attention mechanisms та transformer architectures.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень понять про нейромережевий транслятор.

Завдання лабораторної роботи

1. Проведення досліджень нейромережевого транслятора.

2. Ручне виконання перекладу на підставі нейромережевого транслятора.

3. Обробка результатів та їхнє графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджується нейромережевий транслятор.

2. Реалізація перекладу на підставі нейромережевого транслятора є п’ятиступеневою процедурою.

– Перший ступінь – викачування набору даних.
– Другий ступінь – попередня підготовка даних.
– Третій ступінь – створення моделі транслятора.
– Четвертий ступінь – навчання та оцінка моделі.
– П’ятий ступінь – дешифрування тестової послідовності.

Вибір структури пропозиції з табл. 7.1 – відповідає номеру варіанта.

Таблиця 7.1

Індивідуальні завдання

№ варіанта	Транслятор	Мови
1	seq2seq на підставі SRN	fra-eng
2	seq2seq на підставі GRU	fra-eng
3	seq2seq на підставі LSTM	fra-eng

№ варіанта	Транслятор	Мови
4	Transformer	fra-eng
5	seq2seq на підставі SRN	spa-eng
6	seq2seq на підставі GRU	spa-eng
7	seq2seq на підставі LSTM	spa-eng
8	Transformer	spa-eng
9	seq2seq на підставі SRN	por-eng
10	seq2seq на підставі GRU	por-eng
11	seq2seq на підставі LSTM	por-eng
12	Transformer	por-eng
13	seq2seq на підставі SRN	ita-eng
14	seq2seq на підставі GRU	ita-eng
15	seq2seq на підставі LSTM	ita-eng
16	Transformer	ita-eng

Порядок оформлення лабораторної роботи

7.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш;
- Вступ:

1. Опис виконання перекладу на підставі нейромережевого транслятора.
2. Результати виконання перекладу на підставі нейромережевого транслятора.
3. Аналіз закономірностей.

- Висновки.

7.2. Вступ має містити окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

7.3. У звіті наводиться переклад на підставі нейромережевого транслятора.

7.4. У висновках наводиться порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

Приклад seq2seq на підставі SRN для мови fra-eng

Listing 7.1

```
import numpy, tensorflow
## Викачування набору даних
!curl -O http://www.manythings.org/anki/fra-eng.zip
!unzip -u fra-eng.zip
batch_size = 64 # розмір пакета
latent_dim = 256  # кількість нейронів у прихованому шарі
num_samples = 10000  # довжина навчальної вибірки
## Попередня підготовка даних
input_texts = []
target_texts = []
```

```
input_characters = set()
target_characters = set()
with open("fra.txt", "r", encoding="utf-8") as f:
    lines = f.read().split("\n")
for line in lines[: min(num_samples, len(lines) - 1)]:
    input_text, target_text, _ = line.split("\t")
    target_text = "\t" + target_text + "\n"
    input_texts.append(input_text)
    target_texts.append(target_text)
    for char in input_text:
        if char not in input_characters:
            input_characters.add(char)
    for char in target_text:
        if char not in target_characters:
            target_characters.add(char)
input_characters = sorted(list(input_characters))
target_characters = sorted(list(target_characters))
num_encoder_tokens = len(input_characters)
num_decoder_tokens = len(target_characters)
max_encoder_seq_length = max([len(txt) for txt in input_texts])
max_decoder_seq_length = max([len(txt) for txt in target_texts])
input_token_index = dict([(char, i) for i, char in enumerate(input_characters)])
target_token_index = dict([(char, i) for i, char in enumerate(target_characters)])
encoder_input_data = numpy.zeros((len(input_texts), max_encoder_seq_length,
num_encoder_tokens), dtype="float32")
decoder_input_data = numpy.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype="float32")
decoder_target_data = numpy.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype="float32")
for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
    for t, char in enumerate(input_text):
        encoder_input_data[i, t, input_token_index[char]] = 1.0
    encoder_input_data[i, t + 1 :, input_token_index[" "]] = 1.0
    for t, char in enumerate(target_text):
        decoder_input_data[i, t, target_token_index[char]] = 1.0
        if t > 0:
            decoder_target_data[i, t - 1, target_token_index[char]] = 1.0
    decoder_input_data[i, t + 1 :, target_token_index[" "]] = 1.0
    decoder_target_data[i, t:, target_token_index[" "]] = 1.0
## Створення моделі транслятора
encoder_inputs = tensorflow.keras.Input(shape=(None, num_encoder_tokens))
encoder = tensorflow.keras.layers.SimpleRNN(latent_dim, return_state=True)
encoder_outputs, state_h = encoder(encoder_inputs)
decoder_inputs = tensorflow.keras.Input(shape=(None, num_decoder_tokens))
decoder_srn = tensorflow.keras.layers.SimpleRNN(latent_dim, return_sequences=True,
return_state=True)
```

```
decoder_outputs, _ = decoder_srn(decoder_inputs, initial_state=state_h)
decoder_dense = tensorflow.keras.layers.Dense(num_decoder_tokens, activation="softmax")
decoder_outputs = decoder_dense(decoder_outputs)
model = tensorflow.keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)
encoder_inputs = model.input[0]
encoder_outputs, state_h_enc = model.layers[2].output
encoder_model = tensorflow.keras.Model(encoder_inputs, state_h_enc)
model.compile(optimizer="rmsprop", loss="categorical_crossentropy", metrics=["accuracy"])
decoder_inputs = model.input[1]
decoder_state_input_h = tensorflow.keras.Input(shape=(latent_dim,))
decoder_srn = model.layers[3]
decoder_outputs, state_h_dec = decoder_srn(decoder_inputs, initial_state=decoder_state_input_h)
decoder_dense = model.layers[4]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = tensorflow.keras.Model([decoder_inputs] + [decoder_state_input_h],
[decoder_outputs] + [state_h_dec])
## Навчання і оцінювання моделі
model.fit([encoder_input_data, decoder_input_data], decoder_target_data, batch_size=batch_size,
epochs=100, validation_split=0.2)
## Дешифрування тестової послідовності
reverse_input_char_index = dict((i, char) for char, i in input_token_index.items())
reverse_target_char_index = dict((i, char) for char, i in target_token_index.items())
for seq_index in range(20):
    input_seq = encoder_input_data[seq_index : seq_index + 1]
    states_value = encoder_model.predict(input_seq)
    target_seq = numpy.zeros((1, 1, num_decoder_tokens))
    target_seq[0, 0, target_token_index["\t"]] = 1.0
    stop_condition = False
    decoded_sentence = ""
    while not stop_condition:
        output_tokens, h = decoder_model.predict([target_seq] + [states_value])
        sampled_token_index = numpy.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char
        if sampled_char == "\n" or len(decoded_sentence) > max_decoder_seq_length:
            stop_condition = True
        target_seq = numpy.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.0
        states_value = [h]
    print("-")
    print("Input sentence:", input_texts[seq_index])
    print("Decoded sentence:", decoded_sentence)
```

Приклад seq2seq на підставі GRU для мови fra-eng

```
import numpy, tensorflow
## Викачування набору даних
!curl -O http://www.manythings.org/anki/fra-eng.zip
!unzip -u fra-eng.zip
batch_size = 64 # розмір пакета
latent_dim = 256  # кількість шлюзових рекурентних блоків
num_samples = 10000  # довжина навчальної вибірки
## Попередня підготовка даних
input_texts = []
target_texts = []
input_characters = set()
target_characters = set()
with open("fra.txt", "r", encoding="utf-8") as f:
    lines = f.read().split("\n")
for line in lines[: min(num_samples, len(lines) - 1)]:
    input_text, target_text, _ = line.split("\t")
    target_text = "\t" + target_text + "\n"
    input_texts.append(input_text)
    target_texts.append(target_text)
    for char in input_text:
        if char not in input_characters:
            input_characters.add(char)
    for char in target_text:
        if char not in target_characters:
            target_characters.add(char)
input_characters = sorted(list(input_characters))
target_characters = sorted(list(target_characters))
num_encoder_tokens = len(input_characters)
num_decoder_tokens = len(target_characters)
max_encoder_seq_length = max([len(txt) for txt in input_texts])
max_decoder_seq_length = max([len(txt) for txt in target_texts])
input_token_index = dict([(char, i) for i, char in enumerate(input_characters)])
target_token_index = dict([(char, i) for i, char in enumerate(target_characters)])
encoder_input_data = numpy.zeros((len(input_texts), max_encoder_seq_length,
num_encoder_tokens), dtype="float32")
decoder_input_data = numpy.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype="float32")
decoder_target_data = numpy.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype="float32")
for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
    for t, char in enumerate(input_text):
        encoder_input_data[i, t, input_token_index[char]] = 1.0
    encoder_input_data[i, t + 1 :, input_token_index[" "]] = 1.0
    for t, char in enumerate(target_text):
```

```
        decoder_input_data[i, t, target_token_index[char]] = 1.0
        if t > 0:
            decoder_target_data[i, t - 1, target_token_index[char]] = 1.0
    decoder_input_data[i, t + 1 :, target_token_index[" "]] = 1.0
    decoder_target_data[i, t:, target_token_index[" "]] = 1.0
## Створення моделі транслятора
encoder_inputs = tensorflow.keras.Input(shape=(None, num_encoder_tokens))
encoder = tensorflow.keras.layers.GRU(latent_dim, return_state=True)
encoder_outputs, state_h = encoder(encoder_inputs)
decoder_inputs = tensorflow.keras.Input(shape=(None, num_decoder_tokens))
decoder_gru = tensorflow.keras.layers.GRU(latent_dim, return_sequences=True,
return_state=True)
decoder_outputs, _ = decoder_gru(decoder_inputs, initial_state=state_h)
decoder_dense = tensorflow.keras.layers.Dense(num_decoder_tokens, activation="softmax")
decoder_outputs = decoder_dense(decoder_outputs)
model = tensorflow.keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)
encoder_inputs = model.input[0]
encoder_outputs, state_h_enc = model.layers[2].output
encoder_model = tensorflow.keras.Model(encoder_inputs, state_h_enc)
model.compile(optimizer="rmsprop", loss="categorical_crossentropy", metrics=["accuracy"])
decoder_inputs = model.input[1]
decoder_state_input_h = tensorflow.keras.Input(shape=(latent_dim,))
decoder_gru = model.layers[3]
decoder_outputs, state_h_dec = decoder_gru(decoder_inputs, initial_state=decoder_state_input_h)
decoder_dense = model.layers[4]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = tensorflow.keras.Model([decoder_inputs] + [decoder_state_input_h],
[decoder_outputs] + [state_h_dec])
## Навчання та оцінювання моделі
model.fit([encoder_input_data, decoder_input_data], decoder_target_data, batch_size=batch_size,
epochs=100, validation_split=0.2)
## Дешифрація тестової послідовності
reverse_input_char_index = dict((i, char) for char, i in input_token_index.items())
reverse_target_char_index = dict((i, char) for char, i in target_token_index.items())
for seq_index in range(20):
    input_seq = encoder_input_data[seq_index : seq_index + 1]
    states_value = encoder_model.predict(input_seq)
    target_seq = numpy.zeros((1, 1, num_decoder_tokens))
    target_seq[0, 0, target_token_index["\t"]] = 1.0
    stop_condition = False
    decoded_sentence = ""
    while not stop_condition:
        output_tokens, h = decoder_model.predict([target_seq] + [states_value])
        sampled_token_index = numpy.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char
```

```
        if sampled_char == "\n" or len(decoded_sentence) > max_decoder_seq_length:
            stop_condition = True
        target_seq = numpy.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.0
        states_value = [h]
    print("-")
    print("Input sentence:", input_texts[seq_index])
    print("Decoded sentence:", decoded_sentence)
```

Приклад seq2seq на підставі LSTM для мови fra-eng

Listing 7.2

```
import numpy, tensorflow
## Викачування набору даних
!curl -O http://www.manythings.org/anki/fra-eng.zip
!unzip -u fra-eng.zip
batch_size = 64 # розмір пакета
latent_dim = 256  # кількість однокоміркових блоків пам’яті
num_samples = 10000  # довжина навчальної вибірки
## Попередня підготовка даних
input_texts = []
target_texts = []
input_characters = set()
target_characters = set()
with open("fra.txt", "r", encoding="utf-8") as f:
    lines = f.read().split("\n")
for line in lines[: min(num_samples, len(lines) - 1)]:
    input_text, target_text, _ = line.split("\t")
    target_text = "\t" + target_text + "\n"
    input_texts.append(input_text)
    target_texts.append(target_text)
    for char in input_text:
        if char not in input_characters:
            input_characters.add(char)
    for char in target_text:
        if char not in target_characters:
            target_characters.add(char)
input_characters = sorted(list(input_characters))
target_characters = sorted(list(target_characters))
num_encoder_tokens = len(input_characters)
num_decoder_tokens = len(target_characters)
max_encoder_seq_length = max([len(txt) for txt in input_texts])
max_decoder_seq_length = max([len(txt) for txt in target_texts])
input_token_index = dict([(char, i) for i, char in enumerate(input_characters)])
target_token_index = dict([(char, i) for i, char in enumerate(target_characters)])
```

```
encoder_input_data = numpy.zeros((len(input_texts), max_encoder_seq_length,
num_encoder_tokens), dtype="float32")
decoder_input_data = numpy.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype="float32")
decoder_target_data = numpy.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype="float32")
for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
    for t, char in enumerate(input_text):
        encoder_input_data[i, t, input_token_index[char]] = 1.0
    encoder_input_data[i, t + 1 :, input_token_index[" "]] = 1.0
    for t, char in enumerate(target_text):
        decoder_input_data[i, t, target_token_index[char]] = 1.0
        if t > 0:
            decoder_target_data[i, t - 1, target_token_index[char]] = 1.0
    decoder_input_data[i, t + 1 :, target_token_index[" "]] = 1.0
    decoder_target_data[i, t:, target_token_index[" "]] = 1.0
## Створення моделі транслятора
encoder_inputs = tensorflow.keras.Input(shape=(None, num_encoder_tokens))
encoder = tensorflow.keras.layers.LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
encoder_states = [state_h, state_c]
decoder_inputs = tensorflow.keras.Input(shape=(None, num_decoder_tokens))
decoder_lstm = tensorflow.keras.layers.LSTM(latent_dim, return_sequences=True,
return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
decoder_dense = tensorflow.keras.layers.Dense(num_decoder_tokens, activation="softmax")
decoder_outputs = decoder_dense(decoder_outputs)
model = tensorflow.keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)
encoder_inputs = model.input[0]
encoder_outputs, state_h_enc, state_c_enc = model.layers[2].output
encoder_states = [state_h_enc, state_c_enc]
encoder_model = tensorflow.keras.Model(encoder_inputs, encoder_states)
model.compile(optimizer="rmsprop", loss="categorical_crossentropy", metrics=["accuracy"])
decoder_inputs = model.input[1]
decoder_state_input_h = tensorflow.keras.Input(shape=(latent_dim,))
decoder_state_input_c = tensorflow.keras.Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
decoder_lstm = model.layers[3]
decoder_outputs, state_h_dec, state_c_dec = decoder_lstm(decoder_inputs,
initial_state=decoder_states_inputs)
decoder_states = [state_h_dec, state_c_dec]
decoder_dense = model.layers[4]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = tensorflow.keras.Model([decoder_inputs] + decoder_states_inputs,
[decoder_outputs] + decoder_states)
## Навчання і оцінювання моделі
```

```
model.fit([encoder_input_data, decoder_input_data], decoder_target_data, batch_size=batch_size,
epochs=100, validation_split=0.2)
## Дешифрування тестової послідовності
reverse_input_char_index = dict((i, char) for char, i in input_token_index.items())
reverse_target_char_index = dict((i, char) for char, i in target_token_index.items())
for seq_index in range(20):
    input_seq = encoder_input_data[seq_index : seq_index + 1]
    states_value = encoder_model.predict(input_seq)
    target_seq = numpy.zeros((1, 1, num_decoder_tokens))
    target_seq[0, 0, target_token_index["\t"]] = 1.0
    stop_condition = False
    decoded_sentence = ""
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
        sampled_token_index = numpy.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char
        if sampled_char == "\n" or len(decoded_sentence) > max_decoder_seq_length:
            stop_condition = True
        target_seq = numpy.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.0
        states_value = [h, c]
    print("-")
    print("Input sentence:", input_texts[seq_index])
    print("Decoded sentence:", decoded_sentence)
```

Далі наводимо трансформер для мови spa-eng.

Listing 7.3

```
import random, string, re, numpy, tensorflow
vocab_size = 15000 # розмір словника
sequence_length = 20 # кількість слів
embed_dim = 256 # розмір шару Embedding для кожного слова
batch_size = 64 # розмір пакета
transformer_layers = 1 # кількість блоків трансформера
## Викачування набору даних
!curl -O http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip
!unzip -u spa-eng.zip
## Синтаксичний аналіз даних
with open("spa-eng/spa.txt") as f:
    lines = f.read().split("\n")[:-1]
text_pairs = []
for line in lines:
    eng, spa = line.split("\t")
    spa = "[start] " + spa + " [end]"
    text_pairs.append((eng, spa))
```

```
## Розбиття набору даних на навчальний, перевіряльний і тестовий
random.shuffle(text_pairs)
num_val_samples = int(0.15 * len(text_pairs))
num_train_samples = len(text_pairs) - 2 * num_val_samples
train_pairs = text_pairs[:num_train_samples]
val_pairs = text_pairs[num_train_samples : num_train_samples + num_val_samples]
test_pairs = text_pairs[num_train_samples + num_val_samples :]
## Векторизація набору даних
strip_chars = string.punctuation + "¿"
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")
def custom_standardization(input_string):
   lowercase = tensorflow.strings.lower(input=input_string)
   return tensorflow.strings.regex_replace(input=lowercase, pattern="[%s]" %
re.escape(strip_chars), rewrite="", replace_global=True)
eng_vectorization = tensorflow.keras.layers.TextVectorization( max_tokens=vocab_size,
output_mode="int", output_sequence_length=sequence_length)
spa_vectorization = tensorflow.keras.layers.TextVectorization( max_tokens=vocab_size,
output_mode="int", output_sequence_length=sequence_length + 1,
standardize=custom_standardization)
train_eng_texts = [pair[0] for pair in train_pairs]
train_spa_texts = [pair[1] for pair in train_pairs]
eng_vectorization.adapt(train_eng_texts)
spa_vectorization.adapt(train_spa_texts)
## Створення набору даних
def format_dataset(eng, spa):
   eng = eng_vectorization(eng)
   spa = spa_vectorization(spa)
   return ({"encoder_inputs": eng, "decoder_inputs": spa[:, :-1],}, spa[:, 1:])
def make_dataset(pairs):
   eng_texts, spa_texts = zip(*pairs)
   eng_texts = list(eng_texts)
   spa_texts = list(spa_texts)
   dataset = tensorflow.data.Dataset.from_tensor_slices((eng_texts, spa_texts))
   dataset = dataset.batch(batch_size)
   dataset = dataset.map(format_dataset)
   return dataset
train_ds = make_dataset(train_pairs)
val_ds = make_dataset(val_pairs)
## Клас шифрувальника
class TransformerEncoder(tensorflow.keras.layers.Layer):
   def __init__(self, embed_dim, **kwargs):
      super(TransformerEncoder, self).__init__(**kwargs)
      self.embed_dim = embed_dim
      self.attention = tensorflow.keras.layers.MultiHeadAttention(num_heads=8,
key_dim=embed_dim)
```

```
        self.dense_proj = tensorflow.keras.Sequential([tensorflow.keras.layers.Dense(
units=embed_dim, activation="relu"), tensorflow.keras.layers.Dense(units=embed_dim),])
        self.layernorm_1 = tensorflow.keras.layers.LayerNormalization()
        self.layernorm_2 = tensorflow.keras.layers.LayerNormalization()
        self.supports_masking = True # маска
    def call(self, inputs, mask=None):
        attention_output = self.attention(query=inputs, value=inputs, key=inputs,
attention_mask=None)
        proj_input = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)
        return self.layernorm_2(proj_input + proj_output)
## Клас Embedding позиційного шифрування
class PositionalEmbedding(tensorflow.keras.layers.Layer):
    def __init__(self, sequence_length, vocab_size, embed_dim, **kwargs):
        super(PositionalEmbedding, self).__init__(**kwargs)
        self.token_embeddings = tensorflow.keras.layers.Embedding( input_dim=vocab_size,
output_dim=embed_dim)
        self.position_embeddings = tensorflow.keras.layers.Embedding( input_dim=sequence_length,
output_dim=embed_dim)
        self.sequence_length = sequence_length
        self.vocab_size = vocab_size
        self.embed_dim = embed_dim
    def call(self, inputs):
        length = tensorflow.shape(inputs)[-1]
        positions = tensorflow.keras.backend.arange(start=0, stop=length, step=1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions
    def compute_mask(self, inputs, mask=None): # маска для Embedding
        return tensorflow.math.not_equal(inputs, 0)
## Клас дешифрувальника
class TransformerDecoder(tensorflow.keras.layers.Layer):
    def __init__(self, embed_dim, **kwargs):
        super(TransformerDecoder, self).__init__(**kwargs)
        self.embed_dim = embed_dim
        self.attention_1 = tensorflow.keras.layers.MultiHeadAttention( num_heads=8,
key_dim=embed_dim)
        self.attention_2 = tensorflow.keras.layers.MultiHeadAttention( num_heads=8,
key_dim=embed_dim)
        self.dense_proj = tensorflow.keras.Sequential(
[tensorflow.keras.layers.Dense(units=embed_dim, activation="relu"),
tensorflow.keras.layers.Dense(units=embed_dim)])
        self.layernorm_1 = tensorflow.keras.layers.LayerNormalization()
        self.layernorm_2 = tensorflow.keras.layers.LayerNormalization()
        self.layernorm_3 = tensorflow.keras.layers.LayerNormalization()
        self.supports_masking = True # маска
```

```
    def call(self, inputs, encoder_outputs, mask=None):
        causal_mask = self.get_causal_attention_mask(inputs)
        attention_output_1 = self.attention_1(query=inputs, value=inputs, key=inputs,
attention_mask=causal_mask)
        out_1 = self.layernorm_1(inputs + attention_output_1)
        attention_output_2 = self.attention_2(query=out_1, value=encoder_outputs,
key=encoder_outputs, attention_mask=None)
        out_2 = self.layernorm_2(out_1 + attention_output_2)
        proj_output = self.dense_proj(out_2)
        return self.layernorm_3(out_2 + proj_output)
    def get_causal_attention_mask(self, inputs):
        input_shape = tensorflow.keras.backend.shape(x=inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = tensorflow.keras.backend.arange(start=0, stop=sequence_length, step=1)[:,
tensorflow.newaxis]
        j = tensorflow.keras.backend.arange(start=0, stop=sequence_length, step=1)
        mask = tensorflow.keras.backend.greater_equal(x=i, y=j)
        mask = tensorflow.keras.backend.reshape(x=mask, shape=(1, input_shape[1], input_shape[1]))
        mult = tensorflow.keras.backend.concatenate(
tensors=[tensorflow.keras.backend.expand_dims(x=batch_size, axis=-1),
tensorflow.keras.backend.constant(value=[1, 1], dtype="int32")], axis=0)
        return tensorflow.keras.backend.tile(x=mask, n=mult)
## Створення моделі транслятора
encoder_inputs = tensorflow.keras.Input(shape=(None,), dtype="int64", name="encoder_inputs")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(encoder_inputs)
for i in range(transformer_layers):
    x = TransformerEncoder(embed_dim)(x)
encoder_outputs = x
encoder = tensorflow.keras.Model(encoder_inputs, encoder_outputs)
decoder_inputs = tensorflow.keras.Input(shape=(None,), dtype="int64", name="decoder_inputs")
encoded_seq_inputs = tensorflow.keras.Input(shape=(None, embed_dim))
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
for i in range(transformer_layers):
    x = TransformerDecoder(embed_dim)(x, encoded_seq_inputs)
decoder_outputs = tensorflow.keras.layers.Dense(vocab_size, activation="softmax")(x)
decoder = tensorflow.keras.Model([decoder_inputs, encoded_seq_inputs], decoder_outputs)
decoder_outputs = decoder([decoder_inputs, encoder_outputs])
transformer = tensorflow.keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)
transformer.compile("adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
transformer.summary()
## Навчання і оцінювання моделі
transformer.fit(train_ds, epochs=1, validation_data=val_ds)
## Дешифрування тестової послідовності
spa_vocab = spa_vectorization.get_vocabulary()
spa_index_lookup = dict(zip(range(len(spa_vocab)), spa_vocab))
max_decoded_sentence_length = sequence_length
```

```
def decode_sequence(input_sentence):
    tokenized_input_sentence = eng_vectorization([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = spa_vectorization([decoded_sentence])[:, :-1]
        predictions = transformer([tokenized_input_sentence, tokenized_target_sentence])
        sampled_token_index = numpy.argmax(predictions[0, i, :])
        sampled_token = spa_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token
        if sampled_token == "[end]":
            break
    return decoded_sentence
test_eng_texts = [pair[0] for pair in test_pairs]
for _ in range(10):
    input_sentence = random.choice(test_eng_texts)
    print(input_sentence)
    translated = decode_sequence(input_sentence)
    print(translated)
```

Лабораторна робота № 8
Класифікація тексту
Теоретичні відомості

Класифікація тексту – це процес призначення кожному текстовому документу одної чи кількох попередньо визначених категорій або міток, на основі його змісту або характеристик. Основні принципи класифікації тексту містять:

1. Навчання з учителем: модель навчається на попередньо визначених даних, де кожний текстовий документ має відому мітку або категорію.

2. Векторизація тексту: текстові дані перетворюються на числові вектори, щоб їх можна було подати у вигляді вхідних даних для моделей машинного навчання.

3. Вибір ознак: визначення, які аспекти тексту важливі для класифікації. Це можуть бути слова, фрази, частини мови та інші характеристики тексту.

4. Модель класифікації: використання алгоритму машинного навчання, як-от наївний Баєсів класифікатор, логістичної регресії, дерева рішень або нейронних мереж, для навчання моделі визначати категорії тексту.

5. Оцінка та налаштування моделі: використання метрик оцінки, як-от точність, відновлення, F1-показник, для оцінки ефективності моделі та налаштування параметрів моделі для поліпшення її результатів.

6. Перевірка на нових даних: після навчання моделі вона перевіряється на нових, невідомих даних, щоб оцінити її здатність до класифікації тексту, який не був використаний під час навчання.

Ці принципи допомагають створити ефективну систему класифікації тексту, яка може бути використана для різноманітних завдань, таких як аналіз настроїв, виявлення спаму, категоризація новинних статей тощо.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень поняття про класифікацію тексту.

Завдання лабораторної роботи

1. Проведення досліджень класифікації тексту.

2. Ручне виконання класифікації тексту.

3. Обробка результатів та їхнє графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджується класифікація тексту.

2. Реалізація класифікації тексту є шестиступеневою процедурою.

– Перший ступінь – викачування набору даних.

– Другий ступінь – попередня підготовка даних.
– Третій ступінь – формування векторів ознак.
– Четвертий ступінь – створення моделі класифікації.
– П’ятий ступінь – навчання та оцінка моделі.
– Шостий ступінь – класифікація тексту.

Вибір структури пропозиції з табл. 8.1 – відповідає номеру варіанта.

Таблиця 8.1

Індивідуальні завдання

№ варіанта	Методи класифікації	Система ознак
1	логістичнаа регресія	мішок слів
2	мультиноміальний наївний Баєс	мішок слів
3	Бернуллі наївний Баєс	мішок слів
4	C-SVM	мішок слів
5	ν-SVM	мішок слів
6	логістичнаа регресія	TF
7	мультиноміальний наївний Байес	TF
8	Бернуллі наївний Баєс	TF
9	C-SVM	TF
10	ν-SVM	TF
11	логістична регресія	TFIDF
12	мультиноміальний наївний Баєс	TFIDF
13	Бернуллі наївний Баєс	TFIDF
14	C-SVM	TFIDF
15	ν-SVM	TFIDF

Порядок оформлення лабораторної роботи

8.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис класифікації тексту.
2. Результати класифікації тексту.
3. Аналіз закономірностей.

- Висновки.

8.2. Вступ має містити, окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

8.3. У звіті наводиться класифікація тексту.

8.4. У висновках наводиться порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

Наведемо типовий варіант класифікації тексту.

Listing 8.1

```
!pip install nltk
import re, unicodedata
import nltk
import nltk.stem
import nltk.corpus
import sklearn.linear_model
# import sklearn.naive_bayes, sklearn.svm, sklearn.tree
import sklearn.datasets
import sklearn.feature_extraction.text
import sklearn.metrics
## Викачування
# викачування для стоп-слів
nltk.download('stopwords')
# викачування для пунктуації
nltk.download('punkt')
# викачування для лематизації
nltk.download('wordnet')
## Очищення тексту
def denoiser(text):
    # видалення url
    clean_text = re.sub(r"(?:\|http?\://|https?\://|www)\S+", "", text)
    # видалення email
    clean_text = re.sub(r'[\w.+-]+@[\w-]+\.[\w.-]+', '', clean_text)
    # видалення нагадувань
    clean_text = re.sub(r'@\w+', '', clean_text)
    # видалення цифр
    clean_text = re.sub(r'[0-9]+', '', clean_text)
    # видалення символів пунктуації
    clean_text = re.sub(r'[^\w\s]', '', clean_text)
    # перетворення у нижній регістр
    clean_text = clean_text.lower()
    # видалення кількох символів «нових рядків»
    clean_text = re.sub(r'[\r|\n|\r\n]+', ' ', clean_text)
    # видалення кількох символів пробілів
    clean_text = re.sub(' +', ' ', clean_text)
    # видалення стоп-слів
    sw = nltk.corpus.stopwords.words('english')
    text_list = clean_text.split(" ")
    text_list = [t for t in text_list if t not in sw]
    text_list = [t for t in text_list if t != '']
    # видалення ASCII символів
    clean_text = unicodedata.normalize('NFKD', ' '.join(text_list)).encode('ascii',
'ignore').decode('utf-8', 'ignore')
```

```
    return clean_text
# Токенізація і стемінг/лематизація
def tokenizer(text):
    # токенізація
    text = nltk.word_tokenize(denoiser(text))
    # стемінг
    # stemmer = nltk.stem.LancasterStemmer()
    # text = [stemmer.stem(t) for t in text]
    # лематизація
    lemmatizer = nltk.stem.WordNetLemmatizer()
    text = [lemmatizer.lemmatize(s) for s in text]
    return text
## Викачування набору даних
data_train = sklearn.datasets.fetch_20newsgroups(subset='train')
data_test = sklearn.datasets.fetch_20newsgroups(subset='test')
y_train, y_test = data_train.target, data_test.target
## Перетворення тексту у мішок слів (BoW)
# bow = sklearn.feature_extraction.text.CountVectorizer( tokenizer=tokenizer,
token_pattern=None, analyzer='word', max_features=5000)
# X_train = bow.fit_transform(data_train.data)
# X_test = bow.transform(data_test.data)

## Перетворення тексту до TF
# tf = sklearn.feature_extraction.text.TfidfVectorizer( tokenizer=tokenizer, analyzer='word',
token_pattern=None, max_features=5000, use_idf=False, sublinear_tf=True)
# X_train = tf.fit_transform(data_train.data)
# X_test = tf.transform(data_test.data)

## Перетворення тексту до TFIDF
tfidf = sklearn.feature_extraction.text.TfidfVectorizer( tokenizer=tokenizer, analyzer='word',
token_pattern=None, max_features=5000, use_idf=True, sublinear_tf=True)
X_train = tfidf.fit_transform(data_train.data)
X_test = tfidf.transform(data_test.data)
## Створення моделі класифікації
model = sklearn.linear_model.LogisticRegression()
# model = sklearn.naive_bayes.MultinomialNB()
# model = sklearn.naive_bayes.BernoulliNB()
# model = sklearn.svm.SVC()
# model = sklearn.svm.NuSVC()
# model = sklearn.tree.DecisionTreeClassifier()
## Навчання і оцінювання моделі
model.fit(X_train,y_train)
## Класифікація тексту
y_pred = model.predict(X_test)
print(sklearn.metrics.classification_report(y_test,y_pred)) # 0.76
```

Інший приклад класифікації тексту.

Listing 8.2

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
import joblib
# Приклад навчальних даних
texts = ["Це текст про собак", "Це текст про котів", "Собаки і коти - дуже симпатичні тварини"]
labels = ["собаки", "коти", "собаки_і_коти"]
# Ініціалізація TF-IDF векторизатора
tfidf_vectorizer = TfidfVectorizer()

# Векторизація навчальних даних
X_train_tfidf = tfidf_vectorizer.fit_transform(texts)
# Ініціалізація та навчання моделі класифікації (логістична регресія)
classifier = LogisticRegression()
classifier.fit(X_train_tfidf, labels)
# Збереження векторизатора та моделі
joblib.dump(tfidf_vectorizer, "tfidf_vectorizer.pkl")
joblib.dump(classifier, "classifier.pkl")
# Прийом нового тексту для класифікації
new_text = "Новий текст про котів"
# Векторизація нового тексту
new_text_tfidf = tfidf_vectorizer.transform([new_text])
# Прогнозування категорії для нового тексту
predicted_label = classifier.predict(new_text_tfidf)
print("Приблизна категорія для тексту:", predicted_label[0])
```

Наведемо ще один наочний приклад класифікації тексту.

Listing 8.3

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

# Навчальні дані (тексти та відповідні мітки)
texts = ["Це текст про собак", "Це текст про котів", "Собаки і коти - дуже симпатичні тварини"]
labels = ["собаки", "коти", "собаки_і_коти"]
# Визначення та навчання моделі
text_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('clf', MultinomialNB())
])
text_clf.fit(texts, labels)
# Тестування на навчальних даних
```

```
predicted_labels = text_clf.predict(texts)
# Виведення результатів
for text, predicted_label in zip(texts, predicted_labels):
    print(f"Текст: {text} | Передбачена категорія: {predicted_label}")
# Оцінка точності
accuracy = text_clf.score(texts, labels)
print(f"Точність на навчальних даних: {accuracy * 100:.2f}%")
```

Лабораторна робота № 9
Кластеризація тексту
Теоретичні відомості

Принципи кластеризації тексту – це методи і стратегії, за допомогою яких текстові дані можуть бути згруповані в різні категорії або кластери на основі їхнього змісту, семантики, теми або інших характеристик. Основні принципи кластеризації тексту містять:

1. Семантична схожість: кластери формуються на основі семантичної схожості текстів. Тексти, які мають схожі теми, вміст або ключові слова, можуть бути об'єднані в один кластер.

2. Векторне подання: текстові дані перетворюються в векторне подання, щоб їх можна було обробляти за допомогою алгоритмів машинного навчання. Це може передбачати використання методів, таких як TF-IDF (термін – частота / інверсія частоти документів) або векторизацію слова, наприклад, Word2Vec.

3. Алгоритми кластеризації: існує багато різних алгоритмів кластеризації, зокрема k-means, агломеративна кластеризація, DBSCAN тощо. Вибір конкретного алгоритму може залежати від обсягу даних, характеристик тексту та вимог до точності кластеризації.

4. Оцінка якості кластеризації: для оцінки якості кластеризації використовуються різні метрики, зокрема кошторис кластеризації, внутрішні та зовнішні індекси якості. Ці метрики оцінюють, наскільки добре кластери відображають структуру даних та розділяють текстові дані на правильні групи.

5. Параметризація алгоритмів: деякі алгоритми кластеризації мають параметри, які потрібно налаштувати для кращої продуктивності. Наприклад, в алгоритмі k-means потрібно вибрати кількість кластерів (k), а в DBSCAN – радіус і мінімальну кількість точок у кожному кластері.

6. Передобробка даних: до використання алгоритмів кластеризації зазвичай вимагається попереднє очищення і передобробка текстових даних, разом з токенізацією, видаленням стоп-слів, лематизацією тощо.

Ці принципи допомагають ефективно використовувати методи кластеризації для організації та розуміння великих обсягів текстових даних.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень поняття про кластеризацію тексту.

Завдання лабораторної роботи

1. Проведення досліджень кластеризації тексту.

2. Ручне виконання кластеризації тексту.

3. Обробка результатів та їхнє графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджується кластеризація тексту.

2. Реалізація класифікації тексту має вигляд шестиступеневої процедури.

– Перший ступінь – викачування набору даних.

– Другий ступінь – попередня підготовка даних.

– Третій ступінь – формування векторів ознак.

– Четвертий ступінь – створення моделі кластеризації.

– П’ятий ступінь – навчання та оцінка моделі.

– Шостий ступінь – кластеризація тексту.

Вибір структури пропозиції з табл. 9.1 – відповідає номеру варіанта.

Таблиця 9.1

Індивідуальні завдання

№ варіанта	Методи кластеризації	Система ознак
1	EM-алгоритм для суміші Гауса	мішок слів
2	EM-алгоритм для варіаційної суміші Гауса	мішок слів
3	алгоритм k-середніх	мішок слів
4	алгоритм зсуву середнього значення	мішок слів
5	алгоритм поширення близькості	мішок слів
6	EM-алгоритм для суміші Гауса	TF
7	EM-алгоритм для варіаційної суміші Гауса	TF
8	алгоритм k-середніх	TF
9	алгоритм зсуву середнього значення	TF
10	алгоритм поширення близькості	TF
11	EM-алгоритм для суміші Гауса	TFIDF
12	EM-алгоритм для варіаційної суміші Гауса	TFIDF
13	алгоритм k-середніх	TFIDF
14	алгоритм зсуву середнього значення	TFIDF
15	алгоритм поширення близькості	TFIDF

Порядок оформлення лабораторної роботи

9.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис кластеризації тексту.
2. Результати кластеризації тексту.
3. Аналіз закономірностей.

- Висновки.

9.2. Вступ має містити, окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

9.3. У звіті наводиться кластеризація тексту.

9.4. У висновках наводиться порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

Listing 9.1

```
!pip install nltk
import re, unicodedata
import nltk
import nltk.stem
import nltk.corpus
import sklearn.datasets
import sklearn.feature_extraction.text
import sklearn.cluster
# import sklearn.mixture
import sklearn.decomposition
## Викачування
# викачування для стоп-слів
nltk.download('stopwords')
# викачування для пунктуації
nltk.download('punkt')
# викачування для лематизації
nltk.download('wordnet')
## Очищення
def denoiser(text):
    # видалення url
    clean_text = re.sub(r"(?:\|http?\://|https?\://|www)\S+", "", text)
    # видалення email
    clean_text = re.sub(r'[\w.+-]+@[\w-]+\.[\w.-]+', '', clean_text)
    # видалення згадувань
    clean_text = re.sub(r'@\w+', '', clean_text)
    # видалення цифр
    clean_text = re.sub(r'[0-9]+', '', clean_text)
    # видалення символів пунктуації
    clean_text = re.sub(r'[^\w\s]', '', clean_text)
    # перетворення у нижній регістр
    clean_text = clean_text.lower()
    # видалення кількох символів «нових рядків»
    clean_text = re.sub(r'[\r|\n|\r\n]+', ' ', clean_text)
    # видалення кількох символів пробілів
    clean_text = re.sub(' +', ' ', clean_text)
    # видалення стоп-слів
```

```
    sw = nltk.corpus.stopwords.words('english')
    text_list = clean_text.split(" ")
    text_list = [t for t in text_list if t not in sw]
    text_list = [t for t in text_list if t != '']
    # видалення не ASCII символів
    clean_text = unicodedata.normalize('NFKD', ' '.join(text_list)).encode('ascii',
'ignore').decode('utf-8', 'ignore')
    return clean_text
# токенізація і стемінг/лематизація
def tokenizer(text):
    # токенізація
    text = nltk.word_tokenize(denoiser(text))
    # стемінг
    # stemmer = nltk.stem.LancasterStemmer()
    # text = [stemmer.stem(t) for t in text]
    # лематизація
    lemmatizer = nltk.stem.WordNetLemmatizer()
    text = [lemmatizer.lemmatize(s) for s in text]
    return text
## Викачування набору даних
data_train = sklearn.datasets.fetch_20newsgroups(subset='train')
data_test = sklearn.datasets.fetch_20newsgroups(subset='test')
y_train, y_test = data_train.target, data_test.target
## Перетворення тексту в мішок слів (BoW)
#bow = sklearn.feature_extraction.text.CountVectorizer( tokenizer=tokenizer, token_pattern=None,
analyzer='word', max_features=5000)
#X_train = bow.fit_transform(data_train.data)
#X_test = bow.transform(data_test.data)
## Перетворення тексту до TF
# tf = sklearn.feature_extraction.text.TfidfVectorizer( tokenizer=tokenizer, analyzer='word',
token_pattern=None, max_features=5000, use_idf=False, sublinear_tf=True)
# X_train = tf.fit_transform(data_train.data)
# X_test = tf.transform(data_test.data)
## Перетворення тексту у TFIDF
tfidf = sklearn.feature_extraction.text.TfidfVectorizer( tokenizer=tokenizer, analyzer='word',
token_pattern=None, max_features=5000, use_idf=True, sublinear_tf=True)
X_train = tfidf.fit_transform(data_train.data)
X_test = tfidf.transform(data_test.data)
## Перетворення розрідженої матриці BoW / TF / TFIDF у щільну матрицю
nmf = sklearn.decomposition.NMF(n_components=20, random_state=1, beta_loss='kullback-
leibler', solver='mu', max_iter=1000, l1_ratio=.5)
X_train = nmf.fit_transform(X_train)
X_test = nmf.fit_transform(X_test)
## Створення моделі кластеризації
# model = sklearn.mixture.GaussianMixture(n_components=20, covariance_type='full', tol=0.001,
reg_covar=0.000001, max_iter=100, n_init=1, init_params='kmeans', weights_init=None,
means_init=None, precisions_init=None, random_state=None, warm_start=False, verbose=0)
```

```
# model = sklearn.mixture.BayesianGaussianMixture( n_components=20, covariance_type='full',
tol=0.001, reg_covar=1e-06, max_iter=100, n_init=1, init_params='kmeans',
weight_concentration_prior_type='dirichlet_process', weight_concentration_prior=1/20,
mean_precision_prior=1, mean_prior=None, degrees_of_freedom_prior=n_features+1,
covariance_prior=None, random_state=None, warm_start=False, verbose=0)
# model = sklearn.cluster.KMeans(n_clusters=20, init='k-means++', n_init=10, max_iter=300,
tol=0.0001, verbose=0, copy_x=True, algorithm='auto')
# model = sklearn.cluster.MeanShift(bandwidth=2, seeds=None, bin_seeding=False,
cluster_all=True, n_jobs=1, max_iter=300)
model = sklearn.cluster.AffinityPropagation(damping=0.5, max_iter=200, convergence_iter=15,
copy=True, preference=None, affinity='euclidean', verbose=False, random_state=0)
## Навчання і оцінювання моделі
model.fit(X_train)
## Кластеризація тексту
print(model.predict(X_test))
```

Listing 9.2

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
# Зразок даних (можна замінити власними даними)
data = [
    "Python is a programming language",
    "Machine learning is fun",
    "Clustering is an unsupervised learning technique",
    "NLP stands for Natural Language Processing",
    "Text clustering groups similar documents together"
    "Code python is very simple"
]
# Векторизація тексту з використанням TF-IDF
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(data)

# Кластеризація тексту з використанням k-means
k = 2  # кількість кластерів
kmeans = KMeans(n_clusters=k)
kmeans.fit(X)
# Виведення результатів кластеризації
for i in range(k):
    print("Cluster ", i+1, ":")
    cluster_samples = [data[j] for j in range(len(data)) if kmeans.labels_[j] == i]
    for sample in cluster_samples:
        print("-", sample)
    print("\n")
```

Listing 9.3

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
# Приклад даних
sentences = [
    'apple',
    'banana',
    'orange',
    'lemon',
    'grape',
    'pear'
]
# Векторизація тексту за символами
vectorizer = CountVectorizer(analyzer='char')
X = vectorizer.fit_transform(sentences)
# Кластеризація методом KMeans
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)
# Виведення результатів кластеризації
for i in range(len(sentences)):
    print(sentences[i], '-> Cluster:', kmeans.labels_[i])
```

Лабораторна робота № 10
Вилучення та перетворення контенту вебсторінки
Теоретичні відомості

Принципи вилучення та перетворення контенту вебсторінки можуть варіюватися залежно від потреб і цілей. Ось кілька загальних принципів:

1. Визначення цілей: перш ніж вилучати або перетворювати контент, важливо чітко визначити свої цілі. Це допоможе зосередитися на необхідній інформації та забезпечити ефективність процесу.

2. Аналіз структури сторінки: розуміння структури HTML-коду сторінки допоможе здійснити вибіркове вилучення потрібних елементів. Зазвичай це робиться за допомогою CSS-селекторів або XPath.

3. Використання відкритих API: якщо вебсайт надає API для доступу до даних, це може бути найефективнішим шляхом отримання потрібної інформації.

4. Автоматизація процесу: для великої кількості сторінок або регулярного оновлення інформації ефективно використовувати скрипти або інструменти для автоматизації процесу вилучення та перетворення контенту.

5. Уважне ставлення до правил використання: під час вилучення контенту з вебсторінок важливо дотримуватися правил використання, особливо якщо це зроблено з комерційною метою.

6. Оновлення та підтримка: вебсторінки можуть змінюватися з часом, тому важливо періодично переглядати та оновлювати свої методи вилучення та перетворення контенту.

7. Збереження конфіденційності та прав доступу: під час обробки контенту важливо дотримуватися законів про захист персональних даних та збереження конфіденційності.

Ці принципи можуть бути вихідною точкою для розробки стратегії вилучення та перетворення контенту вебсторінки, але варто також враховувати конкретні вимоги та обмеження вашого проекту.

Мета роботи

Основною метою лабораторної роботи є формування в процесі досліджень поняття про вилучення та перетворення контенту вебсторінки.

Завдання лабораторної роботи

1. Проведення досліджень вилучення та перетворення контенту вебсторінки.

2. Ручне виконання вилучення та перетворення контенту вебсторінки.

3. Обробка результатів та їх графічне подання.

4. Аналіз отриманих результатів.

5. Оформлення результатів роботи.

Методичні вказівки до виконання лабораторної роботи

1. У лабораторній роботі досліджується вилучення та перетворення контенту вебсторінки.

2. Реалізація вилучення та перетворення контенту вебсторінки має вигляд чотириступеневої процедури.

– Перший ступінь – викачування набору даних.

– Другий ступінь – видалення HTML-тегів.

– Третій ступінь – попередня підготовка даних.

– Четвертий ступінь – формування векторів ознак.

Вибір діапазону речень контенту вебсторінки з табл. 10.1 – відповідає номеру варіанта.

Таблиця 10.1

Індивідуальні завдання

№ варіанта	Діапазон речень	Система ознак
1	01:001:001 – 01:001:005	мішок слів
2	01:001:006 – 01:001:010	мішок слів
3	01:001:011 – 01:001:015	мішок слів
4	01:001:016 – 01:001:020	мішок слів
5	01:001:021 – 01:001:025	мішок слів
6	01:001:001 – 01:001:005	TF
7	01:001:006 – 01:001:010	TF
8	01:001:011 – 01:001:015	TF
9	01:001:016 – 01:001:020	TF
10	01:001:021 – 01:001:025	TF
11	01:001:001 – 01:001:005	TFIDF
12	01:001:006 – 01:001:010	TFIDF
13	01:001:011 – 01:001:015	TFIDF
14	01:001:016 – 01:001:020	TFIDF
15	01:001:021 – 01:001:025	TFIDF

Порядок оформлення лабораторної роботи

10.1. Лабораторна робота має бути оформлена відповідно до ДСТУ-3008-95 і містити:

- Титульний аркуш.
- Вступ:

1. Опис вилучення та перетворення контенту вебсторінки.
2. Результати вилучення та перетворення контенту вебсторінки.
3. Аналіз закономірностей.

- Висновки.

10.2. Вступ має містити, окрім самостійної характеристики теми роботи, опис цілей, завдань лабораторної роботи з коротким теоретичним обґрунтуванням їхньої постановки для досліджень.

10.3. У звіті наводиться вилучення та перетворення контенту вебсторінки.

10.4. У висновках наводиться порівняння теоретичних положень й експериментальних результатів, отриманих у лабораторній роботі, оцінюється ступінь досягнення мети й виконання поставлених завдань.

Listing 10.1

```
!pip install nltk
import re, unicodedata, requests, bs4
import nltk
import nltk.stem
import sklearn.feature_extraction.text
## Викачування
# викачування для стоп-слів
nltk.download('stopwords')
# викачування для пунктуації
nltk.download('punkt')
# викачування для лематизації
nltk.download('wordnet')
## Очищення
def denoiser(text):
    # видалення url
    clean_text = re.sub(r"(?:\|http?\://|https?\://|www)\S+", "", text)
    # видалення email
    clean_text = re.sub(r'[\w.+-]+@[\w-]+\.[\w.-]+', '', clean_text)
    # видалення згадувань
    clean_text = re.sub(r'@\w+', '', clean_text)
    # видалення цифр
    clean_text = re.sub(r'[0-9]+', '', clean_text)
    # видалення символів пунктуації
    clean_text = re.sub(r'[^\w\s]', '', clean_text)
    # перетворення в нижній регістр
    clean_text = clean_text.lower()
    # видалення кількох символів «нових рядків»
    clean_text = re.sub(r'[\r|\n|\r\n]+', ' ', clean_text)
    # видалення кількох символів пробілів
    clean_text = re.sub(' +', ' ', clean_text)
    # видалення стоп-слів
    sw = nltk.corpus.stopwords.words('english')
    text_list = clean_text.split(" ")
    text_list = [t for t in text_list if t not in sw]
    text_list = [t for t in text_list if t != '']
    # видалення ASCII символів
```

```
    clean_text = unicodedata.normalize('NFKD', ' '.join(text_list)).encode('ascii', 'ignore').
decode('utf-8', 'ignore')
    return clean_text
# Токенізація і стемінг/лематизація
def tokenizer(text):
    # токенізація
    text = nltk.word_tokenize(denoiser(text))
    # стемінг
    # stemmer = nltk.stem.LancasterStemmer()
    # text = [stemmer.stem(t) for t in text]
    # лематизація
    lemmatizer = nltk.stem.WordNetLemmatizer()
    text = [lemmatizer.lemmatize(s) for s in text]
    return text
# Видалення HTML тегів
def strip_html_tags(text):
    soup = bs4.BeautifulSoup(text, "html.parser")
    [s.extract() for s in soup(['iframe', 'script'])]
    stripped_text = soup.get_text()
    stripped_text = re.sub(r'[\r|\n|\r\n]+', '\n', stripped_text)
    return stripped_text
data = requests.get( 'http://www.gutenberg.org/cache/epub/8001/pg8001.html')
content = data.content
print(content[7706:8535])
# <h1 id="id00002" style="margin-top: 5em">Book 01 Genesis</h1>\r\n\r\n<p
id="id00003">01:001:001 In the beginning God created the heaven and the earth.</p>\r\n\r\n<p
id="id00004" style="margin-left: 0%; margin-right: 0%">01:001:002 And the earth was without
form, and void; and darkness was\r\n upon the face of the deep. And the Spirit of God moved
upon\r\n the face of the waters.</p>\r\n\r\n<p id="id00005">01:001:003 And God said, Let there be
light: and there was light.</p>\r\n\r\n<p id="id00006">01:001:004 And God saw the light, that it
was good: and God divided
the<br>\r\n\r\n\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa
0\xc2\xa0light from the darkness.<br>\r\n</p>\r\n\r\n<p id="id00007">01:001:005 And God called
the light Day, and the darkness he
called<br>\r\n\r\n\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\
xa0\xc2\xa0Night. And the evening and the morning were the first day.<br>\r\n</p>
clean_content = strip_html_tags(content)[1018:1594]
print(clean_content)
# Book 01 Genesis
01:001:001 In the beginning God created the heaven and the earth.
01:001:002 And the earth was without form, and void; and darkness was upon the face of the deep.
And the Spirit of God moved upon the face of the waters.
01:001:003 And God said, Let there be light: and there was light.
01:001:004 And God saw the light, that it was good: and God divided the light from the darkness.
01:001:005 And God called the light Day, and the darkness he called Night. And the evening and
the morning were the first day.
```

```
print(tokenizer(clean_content))
# ['book', 'genesis', 'beginning', 'god', 'created', 'heaven', 'earth', 'earth', 'without', 'form', 'void', 'darkness', 'wa', 'upon', 'face', 'deep', 'spirit', 'god', 'moved', 'upon', 'face', 'water', 'god', 'said', 'let', 'light', 'light', 'god', 'saw', 'light', 'good', 'god', 'divided', 'the', 'light', 'darkness', 'god', 'called', 'light', 'day', 'darkness', 'called', 'night', 'evening', 'morning', 'first', 'day']
## Перетворення тексту на мішок слів (BoW)
# bow = sklearn.feature_extraction.text.CountVectorizer( tokenizer=tokenizer, token_pattern=None, analyzer='word', max_features=5000)
# X_train = bow.fit_transform([clean_content])
## Перетворення тексту до TF
# tf = sklearn.feature_extraction.text.TfidfVectorizer( tokenizer=tokenizer, analyzer='word', token_pattern=None, max_features=5000, use_idf=False, sublinear_tf=True)
# X_train = tf.fit_transform(data_train.data)
## Перетворення тексту до TFIDF
tfidf = sklearn.feature_extraction.text.TfidfVectorizer( tokenizer=tokenizer, analyzer='word', token_pattern=None, max_features=5000, use_idf=True, sublinear_tf=True)
X_train = tfidf.fit_transform([clean_content])
```

Надамо приклад інших варіантів роботи з вебсторінками:

Listing 10.2

```
import re, unicodedata, requests, bs4
import nltk
import nltk.stem
import sklearn.feature_extraction.text
## Екстракція
# екстракція для стоп-слів
nltk.download('stopwords')
# екстракція для пунктуації
nltk.download('punkt')
# екстракція для лематизації
nltk.download('wordnet')
## Очищення
def denoiser(text):
    # видалення url
    clean_text = re.sub(r"(?:\|http?\://|https?\://|www)\S+", "", text)
    # видалення.email
    clean_text = re.sub(r'[\w.+-]+@[\w-]+\.[\w.-]+', '', clean_text)
    # видалення згадувань
    clean_text = re.sub(r'@\w+', '', clean_text)
    # видалення цифр
    clean_text = re.sub(r'[0-9]+', '', clean_text)
    # видалення символів пунктуації
    clean_text = re.sub(r'[^\w\s]', '', clean_text)
    # перетворення у нижній регістр
    clean_text = clean_text.lower()
```

```
    # видалення кількох символів «нових рядків»
    clean_text = re.sub(r'[\r|\n|\r\n]+', ' ', clean_text)
    # видалення кількох символів пробілів
    clean_text = re.sub(' +', ' ', clean_text)
    # видалення стоп-слів
    sw = nltk.corpus.stopwords.words('english')
    text_list = clean_text.split(" ")
    text_list = [t for t in text_list if t not in sw]
    text_list = [t for t in text_list if t != '']
    # видалення ASCII символів
    clean_text = unicodedata.normalize('NFKD', ' '.join(text_list)).encode('ascii',
'ignore').decode('utf-8', 'ignore')
    return clean_text
# Токенізація і стемінг/лематизація
def tokenizer(text):
    # токенізація
    text = nltk.word_tokenize(denoiser(text))
    # стемінг
    # stemmer = nltk.stem.LancasterStemmer()
    # text = [stemmer.stem(t) for t in text]
    # лематизація
    lemmatizer = nltk.stem.WordNetLemmatizer()
    text = [lemmatizer.lemmatize(s) for s in text]
    return text
# Видалення HTML тегів
def strip_html_tags(text):
    soup = bs4.BeautifulSoup(text, "html.parser")
    [s.extract() for s in soup(['iframe', 'script'])]
    stripped_text = soup.get_text()
    stripped_text = re.sub(r'[\r|\n|\r\n]+', '\n', stripped_text)
    return stripped_text
data = requests.get( 'http://www.wipo.int/')
content = data.content
print(content[7706:8535])
# <h1 id="id00002" style="margin-top: 5em">Book 01 Genesis</h1>\r\n\r\n<p
id="id00003">01:001:001 In the beginning God created the heaven and the earth.</p>\r\n\r\n<p
id="id00004" style="margin-left: 0%; margin-right: 0%">01:001:002 And the earth was without
form, and void; and darkness was\r\n upon the face of the deep. And the Spirit of God moved
upon\r\n the face of the waters.</p>\r\n\r\n<p id="id00005">01:001:003 And God said, Let there
be light: and there was light.</p>\r\n\r\n<p id="id00006">01:001:004 And God saw the light, that
it was good: and God divided
the<br>\r\n\r\n\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa
0\xc2\xa0light from the darkness.<br>\r\n</p>\r\n\r\n<p id="id00007">01:001:005 And God
called the light Day, and the darkness he
called<br>\r\n\r\n\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\
```

```
xa0\xc2\xa0Night. And the evening and the morning were the first day.<br>\r\n</p>
clean_content = strip_html_tags(content)[1018:1594]
print(clean_content)
print(tokenizer(clean_content))
tfidf = sklearn.feature_extraction.text.TfidfVectorizer( tokenizer=tokenizer, analyzer='word',
token_pattern=None, max_features=5000, use_idf=True, sublinear_tf=True)
X_train = tfidf.fit_transform([clean_content])
```

Listing 10.3

```
import requests
from bs4 import BeautifulSoup
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
# Функція для отримання тексту з вебсторінки
def get_text_from_url(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    # Вилучаємо тільки текстовий вміст
    text = ' '.join([p.text for p in soup.find_all('p')])
    return text
# Функція для обробки тексту та виведення інформації про слова, що найбільш часто
зустрічаються
def analyze_text(text):
    # Розбиваємо текст на слова
    words = word_tokenize(text.lower())  # Переводимо в нижній регістр
    # Видаляємо стоп-слова
    stop_words = set(stopwords.words('english'))
    filtered_words = [word for word in words if word.isalnum() and word not in stop_words]
    # Обчислюємо частоту появи кожного слова
    freq_dist = FreqDist(filtered_words)
    # Виводимо інформацію про слова, що найбільш часто зустрічаються
    print("Most common words:")
    for word, frequency in freq_dist.most_common(10):
        print(f"{word}: {frequency}")
    # Графік розподілу частоти слів
    freq_dist.plot(20, cumulative=False)
    plt.show()
# Посилання на вебсторінку для аналізу
url = "https://example.com"
# Отримання тексту з вебсторінки
web_text = get_text_from_url(url)
# Аналіз тексту
analyze_text(web_text)
```

Література

1. Бахрушин В. Є. Методи аналізу даних: навчальний посібник для студентів. Запоріжжя: КПУ, 2011. 268 с.

2. Марченко О. О., Россада Т. В. Актуальні проблеми Data Mining: навчально-методичний посібник. Київ: КНУ імені Тараса Шевченка, 2017. 150 с.

3. Сопронюк Т. М. Системне програмування. Частина І. Елементи теорії формальних мов: навчальний посібник у двох частинах. Чернівці: ЧНУ імені Юрія Федьковича, 2008. 84 с.

4. Сопронюк Т. М. Системне програмування. Частина ІІ. Елементи теорії компіляції: навчальний посібник у двох частинах. Чернівці: ЧНУ імені Юрія Федьковича, 2008. 84 с.

5. Ланде Д. В., Субач І. Ю., Бояринова Ю. Є. Основи теорії і практики інтелектуального аналізу даних у сфері кібербезпеки: навчальний посібник. Київ: КПІ імені Ігоря Сікорського, 2018. 297 с.

6. Dipanjan Sarkar. Text Analytics with Python. A Practitioners Guide to Natural Language Processing. New York: Apress Media, LLC, 2019. 674 p.

7. Bird S., Klein E., Loper E. Natural Language Processing with Python. USA: O'Reilly, 2009. 502 p.

8. Інтелектуальні технології Text Mining. *Pidru4niki*. URL: https://pidru4niki.com/1722020647790/informatika/intelektualni_tehnologiyi_text_mining (дата звернення: 24.07.2025).

9. Верес О. М., Оливко Р. М. Класифікація методів аналізу великих даних. *Вісник Національного університету «Львівська політехніка».* 2017. № 872. С. 84–92. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2018/jun/13005/ilovepdfcom-84-92.pdf

ДЛЯ НОТАТОК

Навчальне видання

Ніколюк Петро Карпович

ТЕХНОЛОГІЇ TEXTMINING AND WEBMINING

Методичні вказівки
до виконання самостійних та лабораторних робіт
для здобувачів вищої освіти ОС «Бакалавр»
спеціальності 122 Комп'ютерні науки ОП «Комп'ютерні науки»

Редактор О. А. Солдатова
Технічний редактор Т. О. Важеніна-Гопрак

Підписано до друку 22.01.2025.
Формат 60 × 84/16. Папір офсетний.
Друк – цифровий. Умовн. друк. арк. 4,18.
Тираж 30. Зам. 52.

Донецький національний університет імені Василя Стуса
21021, м. Вінниця, 600-річчя, 21.
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру
серія ДК № 5945 від 15.01.2018