

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ КИЇВСЬКИЙ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА**

кафедра інформаційних систем та технологій

ЗВІТ

із лабораторної роботи №2

з дисципліни «**Технології програмування об'єктів лінгвістичної
предметної галузі**»

на тему: Аналіз подібності слів і тексту

Варіант № 2-06

Виконав:

Студент групи №2

Кличлієв К. С.

Перевірив:

доц. Костіков М. П.

Київ — 2022

Дані з індивідуального варіанту № 2-06

Іменник 1: love

Іменник 2: leave

N = 5

Файл: bronte_eyre.txt

Мета роботи

Метою лабораторної роботи №2 є насамперед аналіз подібності слів і тексту за допомогою WordNet (семантичного словника англійської мови, який є вбудованим у бібліотеку nltk), а саме

- дослідження усіх семантичних значень поданих іменників;
- визначення гіпонімів та гіперонімів цих іменників за допомогою методів WordNet;
- обчислення семантичної подібності методами Path Distance Similarity, Wu-Palmer Similarity, Leacock Chodorow Similarity;
- обрахування відстані Левенштейна і Дамерау-Левенштейна між двома словами;
- знаходження для певної лексеми N найближчих слів із наявного словника.

Крім цього, ціллю цієї ЛР є також дослідження частотності використання слів у художньому творі (“Джейн Ейр” Шарлотти Бронте), а також поновлення знань із роботи з файлами в МП Python.

Середовище розробки

Мова програмування: Python

Назва та версія IDE: PyCharm 2022.2.3 (Community Edition)

Назва та версія ОС: Linux Ubuntu 21.04

Мова ОС: англійська

Хід роботи

1. Створюємо новий консольний проєкт мовою Python, при запуску виводимо власне прізвище, ім'я, групу, номер ЛР. Імпортуємо до проєкту бібліотеку NLTK і корпус WordNet, який містить семантичний словник англійської мови, а також інші бібліотеки, необхідні для реалізації поставлених задач (Levenshtein, fastDamerauLevenshtein, difflib, seaborn)

```
1  from nltk.probability import FreqDist
2  from nltk.corpus import wordnet as wn
3  from Levenshtein import distance as lev
4  from fastDamerauLevenshtein import damerauLevenshtein
5  import difflib
6  import seaborn as sns
7  import nltk
8
9  print("Кличієв Кирило\nГрупа №2\nЛабораторна робота №2\n")
10
```

2. Виводимо в консоль визначення (тлумачення) для всіх семантичних значень іменника 1 (love) і іменника 2 (leave):

```

11 # Завдання 2: вивести в консоль тлумачення для всіх семантичних значень поданих іменників
12
13 love_noun_definitions = wn.synsets('love', pos=wn.NOUN)
14 leave_noun_definitions = wn.synsets('leave', pos=wn.NOUN)
15
16 print(f"Усі тлумачення іменника 'love':")
17 for i, w in enumerate(wn.synsets('love', pos=wn.NOUN)):
18     print(f"{i + 1}.", w.definition())
19
20 print(f"\nУсі тлумачення іменника 'leave':")
21 for i, w in enumerate(wn.synsets('leave', pos=wn.NOUN)):
22     print(f"{i + 1}.", w.definition())
23

```

```

Усі тлумачення іменника 'love':
1. a strong positive emotion of regard and affection
2. any object of warm affection or devotion
3. a beloved person; used as terms of endearment
4. a deep feeling of sexual desire and attraction
5. a score of zero in tennis or squash
6. sexual activities (often including sexual intercourse) between two people

Усі тлумачення іменника 'leave':
1. the period of time during which you are absent from work or duty
2. permission to do something
3. the act of departing politely

```

3. Виводимо в консоль усі гіпоніми та гіпероніми для цих же слів:

```

24 # Завдання 3: вивести в консоль усі гіпоніми та гіпероніми для цих слів
25
26 print("\nГіпоніми до всіх тлумачень слова 'love': ")
27 for i in love_noun_definitions:
28     x = i.hyponyms()
29     if len(x) == 0:
30         print(f"{i}: це тлумачення не має гіпонімів.")
31     else:
32         print(f"{i}: ", x)
33
34 print("\nГіпоніми до всіх тлумачень слова 'leave': ")
35 for i in leave_noun_definitions:
36     x = i.hyponyms()
37     if len(x) == 0:
38         print(f"{i}: це тлумачення не має гіпонімів.")
39     else:
40         print(f"{i}: ", x)
41
42 print("\nГіпероніми до всіх тлумачень слова 'love': ")
43 for i in love_noun_definitions:
44     x = i.hypernyms()
45     if len(x) == 0:
46         print(f"{i}: це тлумачення не має гіперонімів.")
47     else:
48         print(f"{i}: ", x)
49
50 print("\nГіпероніми до всіх тлумачень слова 'leave': ")
51 for i in leave_noun_definitions:
52     x = i.hypernyms()
53     if len(x) == 0:
54         print(f"{i}: це тлумачення не має гіперонімів.")
55     else:
56         print(f"{i}: ", x)
57

```

Гіпоніми до всіх тлумачень слова 'love':

```

Synset('love.n.01'): [Synset('agape.n.01'), Synset('agape.n.02'), Synset('amorousness.n.01'), Synset('ardor.n.02')]
Synset('love.n.02'): це тлумачення не має гіпонімів.
Synset('beloved.n.01'): це тлумачення не має гіпонімів.
Synset('love.n.04'): це тлумачення не має гіпонімів.
Synset('love.n.05'): це тлумачення не має гіпонімів.
Synset('sexual_love.n.02'): це тлумачення не має гіпонімів.

```

Гіпоніми до всіх тлумачень слова 'leave':

```

Synset('leave.n.01'): [Synset('compassionate_leave.n.01'), Synset('furlough.n.01'), Synset('pass.n.02'), Synset('s')]
Synset('leave.n.02'): це тлумачення не має гіпонімів.
Synset('farewell.n.02'): [Synset('valediction.n.02')]

```

Гіпероніми до всіх тлумачень слова 'love':

```

Synset('love.n.01'): [Synset('emotion.n.01')]
Synset('love.n.02'): [Synset('object.n.04')]
Synset('beloved.n.01'): [Synset('lover.n.01')]
Synset('love.n.04'): [Synset('sexual_desire.n.01')]
Synset('love.n.05'): [Synset('score.n.03')]
Synset('sexual_love.n.02'): [Synset('sexual_activity.n.01')]

```

Гіпероніми до всіх тлумачень слова 'leave':

```

Synset('leave.n.01'): [Synset('time_off.n.01')]
Synset('leave.n.02'): [Synset('permission.n.01')]
Synset('farewell.n.02'): [Synset('departure.n.01')]

```

4. Знаходимо найнижчий у ієрархії понять спільний гіперонім для слів love і leave:

```
58 # Завдання 4: знайти найнижчий в ієрархії понять спільний гіперонім для іменника 1 і іменника 2
59
60 love = wn.synset('love.n.01')
61 leave = wn.synset('leave.n.01')
62 love_leave_hyponym = love.lowest_common_hyponyms(leave)
63 print(f"\nНайнижчий в ієрархії понять гіперонім іменників 1 і 2 є {love_leave_hyponym}")
64
```

```
Найнижчий в ієрархії понять гіперонім іменників 1 і 2 є [Synset('abstraction.n.06')]
```

5. Обчислюємо семантичну подібність іменників love і leave за допомогою методів:

- Path Distance Similarity;
- Wu-Palmer Similarity;
- Leacock Chodorow Similarity.

```
65 # Завдання 5: обчислити семантичну подібність іменника 1 і 2 за допомогою методів PDS, WPS, LCS
66
67 print(f"\nСемантична подібність поданих слів за метрикою PDS складає "
68       f"{love.path_similarity(leave)}")
69 print(f"Семантична подібність поданих слів за метрикою WPS складає "
70       f"{love.wup_similarity(leave)}")
71 print(f"Семантична подібність поданих слів за метрикою LCS складає "
72       f"{love.lch_similarity(leave)}")
73
```

```
Семантична подібність поданих слів за метрикою PDS складає 0.09090909090909091
Семантична подібність поданих слів за метрикою WPS складає 0.2857142857142857
Семантична подібність поданих слів за метрикою LCS складає 1.2396908869280152
```

6. Знаходимо відстань Левенштейна між іменниками love і leave за допомогою методу distance бібліотеки Levenshtein:

```
74 # Завдання 6: знайти відстань Левенштейна
75
76 print(f"\nВідстань Левенштейна між слова love та leave становить: " + str(lev('love', 'leave')))
77
```

```
Відстань Левенштейна між слова love та leave становить 2
```

7. Знаходимо відстань Дameraу — Левенштейна між іменниками love і leave за допомогою методу damerauLevenshtein модуля fastDamerauLevenshtein:

```
78 # Завдання 7: знайти відстань Дameraу Левенштейна між іменниками
79
80 dam_lev_sim = damerauLevenshtein('love', 'leave', similarity=True)
81 print(f"\nВідстань Дameraу-Левенштейна між слова love та leave становить " + str(dam_lev_sim))
82
```

Відстань Дameraу-Левенштейна між слова love та leave становить 0.6

8. За допомогою вбудованого методу input просимо користувача ввести довільне слово англійською, і знаходимо до цього слова 5 (N=5 за індивідуальним варіантом) найближчих слів із наявного словника в долученому до завдання текстовому файлі 1–1000.txt:

```
83 # Завдання 8: знайти до введеного словника найближчі 5 слів із файлу 1-1000.txt
84
85 user_inp = input("\nВведіть довільне слово англійською мовою: ")
86 f = open("/home/klychliieffx/Desktop/k/1-1000.txt", "r")
87 words = f.read()
88 x = words.split('\n')
89 c = difflib.get_close_matches(user_inp, x, n=5)
90 print(f"Найближчі 5 слів із файлу до слова '{user_inp}': {c}")
91
```

Введіть довільне слово англійською мовою: **gore**
Найближчі 5 слів файлу до слова 'gore': more, grew, gone, store, shore

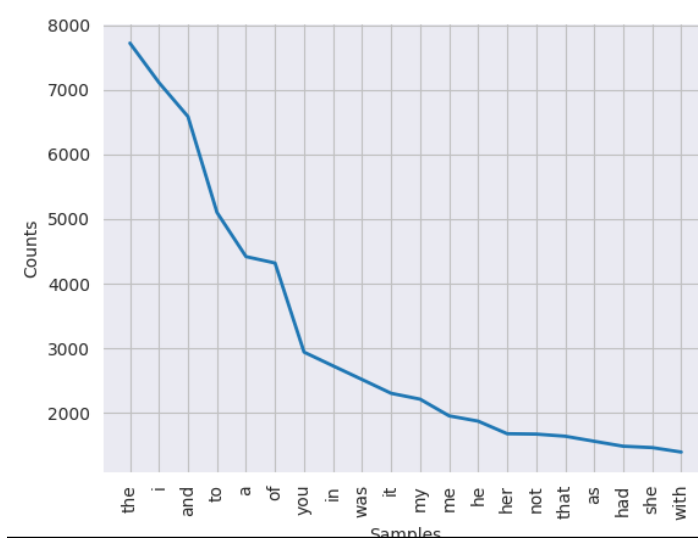
9. Беремо як джерело даних текстовий файл за індивідуальним варіантом (Jane_Eyre.txt), після чого:

- знаходимо всі слова, які зустрічаються в цьому файлі за допомогою методу `word_tokenize` бібліотеки `nltk`:

```
92 # Завдання 9: взяти як джерело даних текстовий файл
93
94 file = open("/home/klychliievfx/Desktop/k/Jane-Eyre.txt", "r")
95
96 # знайти всі слова, що зустрічаються в тексті
97
98 tokenized_text = nltk.word_tokenize(file.read())
99
```

- сортуємо слова за спаданням частотності використовуючи метод `FreqDist` і за допомогою бібліотеки `seaborn` візуалізуємо графік, на якому зображені 20 найчастотніших слів:

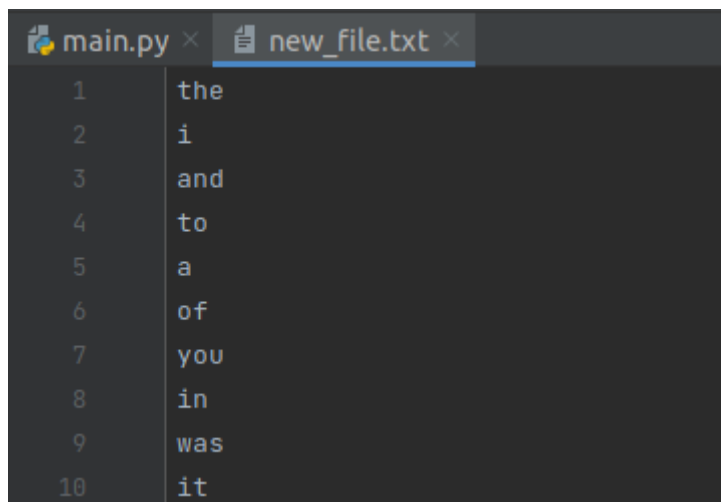
```
100 # відсортувати їх за спаданням частотності
101
102 words = []
103 for word in tokenized_text:
104     if word.isalpha():
105         words.append(word.lower())
106
107 sns.set_style('darkgrid')
108 nlp_words = FreqDist(words)
109 nlp_words.items()
110 nlp_words.plot(20)
111
```



- створюємо новий текстовий файл (new_file.txt) і зберігаємо в ньому відсортовані слова за зразком 1–1000.txt, записуючи кожне наступне слово з нового рядка:

```
115 def convert_tuple(tup):
116     st = ''.join(map(str, tup[0]))
117     return st
118
119
120     f = open("new_file.txt", "w")
121     for word in nlp_words.most_common():
122         myStr = convert_tuple(word)
123         f.write(myStr + '\n')
124
125
126     f.close()
```

Так виглядають записи у новоствореному файлі (усіх записів 12161):



```
main.py × new_file.txt ×
1 the
2 i
3 and
4 to
5 a
6 of
7 you
8 in
9 was
10 it
```

- знаходимо для введенного користувачем слова 5 найближчих слів (із новоствореного файлу new_file.txt) використовуючи метод `get_close_matches` бібліотеки `difflib`:

```

128 # знайти найближчі 5 слів у новому файлі
129
130 user_inp = input("\nВведіть довільне слово англійською мовою: ")
131 f = open("new_file.txt", "r")
132 words = f.read()
133 words_list = words.split('\n')
134 close_matches = difflib.get_close_matches(user_inp, words_list, n=5)
135 print(f"Найближчі 5 слів із файлу до слова '{user_inp}': {close_matches}")

```

```

Введіть довільне слово англійською мовою: gore
Найближчі 5 слів до слова 'gore': gore, gorge, ore, gorged, george

```

Відповідь на контрольне запитання №6

Синсети WordNet та основні дії над ними.

WordNet – це семантично орієнтований словник англійської мови, базовою словникової одиницею в якого є не окреме слово, а так званий синонімічний ряд (синсет, англ. - synset), який об'єднує слова зі схожим значенням і за своєю суттю є вузлами мережі. Для зручності використання словника людиною кожен синсет доповнений значеннями і прикладами вживання слів в контексті. Слово або словосполучення може з'являтися більш ніж в одному синсеті і мати більше однієї категорії частини мови. Кожен синсет містить список синонімів або синонімічних словосполучень і покажчики, що описують відношення між ним та іншими синсетами. Слова, що мають кілька значень, включаються в кілька синсетов і можуть бути зараховані до різних синтаксичних і лексичних класів. NLTK включає англійську WordNet обсягом 155,287 слів і 117,659 наборів синонімів.

Синсет певної лексеми можна отримати за допомогою метода `wn.synsets()`, де `wn` - імпортований словник WordNet (`import wordnet as wn`), а `synsets` - функція, що повертає список синсету до вхідного слова (в нашому випадку 'evil'):

```
wn.synsets('evil')

[Synset('evil.n.01'),
 Synset('evil.n.02'),
 Synset('evil.n.03'),
 Synset('evil.a.01'),
 Synset('evil.s.02'),
 Synset('malefic.s.01')]
```

Список синонімів можна отримати за допомогою методу `lemma_names()`:

```
wn.synset('evil.n.01').lemma_names()

['evil', 'immorality', 'wickedness', 'iniquity']
```

Синсети мають визначення (definition) і приклади використання (examples):

```
wn.synset('evil.n.02').definition()

'that which causes harm or destruction or misfortune; - Shakespeare'
```

```
wn.synset('evil.n.02').examples()

['the evil that men do lives after them; the good is oft interred with their bones']
```

Якщо потрібно вивести тлумачення лише для певної частини мови можна додати аргумент `pos`:

```
for i in wn.synsets('evil', pos=wn.NOUN):
    print(i)
    print(i.definition())
    print()

Synset('evil.n.01')
morally objectionable behavior

Synset('evil.n.02')
that which causes harm or destruction or misfortune; - Shakespeare

Synset('evil.n.03')
the quality of being morally wrong in principle or practice
```

Наступний приклад показує, як можна вивести всю інформацію про синсети (синонім, частину мову, пов'язані з ним лемми, визначення):

```

synsets = wn.synsets('evil')
for s in synsets:
    print()
    print ("Name:", s.name())
    print ("Lexical Type:", s.lexname())
    print ("Lemmas:", s.lemma_names())
    print ("Definition:", s.definition())

Name: evil.n.01
Lexical Type: noun.act
Lemmas: ['evil', 'immorality', 'wickedness', 'iniquity']
Definition: morally objectionable behavior

Name: evil.n.02
Lexical Type: noun.attribute
Lemmas: ['evil']
Definition: that which causes harm or destruction or misfortune; - Shakespeare

Name: evil.n.03
Lexical Type: noun.attribute
Lemmas: ['evil', 'evilness']
Definition: the quality of being morally wrong in principle or practice

Name: evil.a.01
Lexical Type: adj.all
Lemmas: ['evil']
Definition: morally bad or wrong

Name: evil.s.02
Lexical Type: adj.all
Lemmas: ['evil', 'vicious']
Definition: having the nature of vice

Name: malefic.s.01
Lexical Type: adj.all
Lemmas: ['malefic', 'malevolent', 'malign', 'evil']
Definition: having or exerting a malignant influence

```

Синсети у WordNet зв'язані між собою різними семантичними відношеннями:

- гіпероніми;
- гіпоніми;
- антоніми;
- мероніми тощо.

Гіпоніми для слова evil:

```

evil_noun = wn.synset('evil.n.03')
types = evil_noun.hyponyms()
print(types)

[Synset('error.n.05'), Synset('frailty.n.02'), Synset('maleficence.n.02'), Synset('malevolence.n.02'), Synset('malignity.n.02'), Synset('nefariousness.n.01'), Synset('perversity.n.02'), Synset('reprehensibility.n.01'), Synset('villainy.n.01'), Synset('worst.n.02')]

```

Найзагальніший гіперонім до аналізованого слова:

```
evil_noun.root_hypernyms()

[Synset('entity.n.01')]
```

Щоб знайти антоніми використовують метод `antonyms()`:

```
wn.lemma('staccato.r.01.staccato').antonyms()

[Lemma('legato.r.01.legato')]
```

Синсети з'єднані складною мережею лексичних відношень. З огляду на певний синсет ми можемо знайти, які слова семантично пов'язані один з одним. Існує досить багато різних метрик семантичної близькості між словами, які засновані на ієрархії WordNet. Всі вони враховують глибину синсета щодо головного кореня. Метод `min_depth()` повертає мінімальний шлях від кореня дерева до зазначеного синсету. Синсети з'єднані складною мережею лексичних відношень. З огляду на певний синсет ми можемо знайти, які слова семантично пов'язані один з одним. Існує досить багато різних метрик семантичної близькості між словами, які засновані на ієрархії WordNet. Всі вони враховують глибину синсета щодо головного кореня. Метод `min_depth()` повертає мінімальний шлях від кореня дерева до зазначеного синсету.

```
wn.synset('baleen_whale.n.01').min_depth()

14

wn.synset('whale.n.02').min_depth()

13

wn.synset('vertebrate.n.01').min_depth()

8
```

Визначаємо близькість між першим значенням слів `dog` і `shetland` за допомогою метрик WUP і PATH:

```
w1 = wn.synset('dog.n.01')
w2 = wn.synset('shetland.n.01')
print ('similarity WUP - ' + str(w1.wup_similarity(w2)))
print ('similarity PATH - ' + str(w1.path_similarity(w2)))

similarity WUP - 0.4
similarity PATH - 0.1
```

Висновок

Отже, у ході виконання Лабораторної роботи №2 було досягнуто всіх поставлених цілей (див. розділ Мета). Зокрема, було досліджено семантичну подібність між словами “love” і “leave”: виміряно між ними відстань Левенштейна (2) і Дамерау-Левенштейна (0.6), семантичну подібність методами PDS (0.09), WPS (0.29), LC (1.24), знайдено спільний гіперонім між ними (“abstraction”). Також записано у новий файл відсортовані за спаданням частоти слова з роману “Джейн Ейр” Шарлотти Бронте.

Додаток

Посилання на репозиторій із ресурсами з ЛР №2:

<https://github.com/klychliiev/Semantic-analysis.git>