МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Лабораторна робота № 10

ВИВЧЕННЯ БІБЛІОТЕКИ ПРИКЛАДНИХ ПРОГРАМ NLTK, ДЛЯ ОПРАЦЮВАННЯ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ. АВТОМАТИЧНИЙ МОРФОЛОГІЧНИЙ АНАЛІЗ (частина 2)

Виконала:
студентка групи ПРЛм-12
Іваськів М.Є.

Прийняв:
Дупак Б.П.

Львів 2015

МЕТА РОБОТИ

Вивчення основ програмування на мові Python. Ознайомлення з автоматичним морфологічним аналізом в NLTK.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Автоматичний морфологічний аналізатор по замовчуванню

Найпростіший можливий морфологічний аналізатор автоматично призначає той самий тег кожному слову. Це доволі дивний підхід, але він дозволяє зробити важливий початковий крок по створенню аналізатора. Для одержання максимально якісного результату кожне слово маркується найбільш уживаним тегом.

The Lookup Tagger Пошуковий морфологічний аналізатор

Слова, які мають високу частоту (часто зустрічаються в текстах), необов'язково мають тег *nn*. Спробуємо знайти найчастотніші слова та відповідні до них теги. Таку інформацію можна буде в подальшому використати, як модель для пошукового аналізатора "lookup tagger" в NLTK UnigramTagger.

Набагато кращий підхід полягає в тому, щоб побудувати словник, в якому встановлюється відповідність між цими словами і їх найвірогіднішими тегами. Це можна зробити налаштувавши функцію умовного частотного розподілу *cfd* для кожного промаркованого слова, тобто отримати частоту різних тегів, які зустрічаються з кожним словом. Тепер для будь-якого слова з цієї частини корпуса можна визначити найбільш вірогідний тег.

Уніграм аналізатор

Уніграм аналізатор (Unigram taggers) реалізовує простий статистичний алгоритм маркування слів. Кожному слову (tokens) ставиться у відповідність тег, який є найбільш імовірний для цього слова. Наприклад, згідно цього алгоритму тег *jj* буде поставлений у відповідність до кожного слова frequent в тексті, оскільки це слово частіше використовується, як прикметник (a frequent word) і рідко, як дієслово (I frequent this lecture).

Перед використанням уніграм аналізатора для аналізу тексту потрібно провести його тренування (навчання) на розміченому корпусі текстів. Аналізатор використовує корпус для визначення тегів, які властиві кожному слову. В наступному прикладі, здійснюється ініціалізація і тренування аналізатора **#1**. При створенні (ініціалізації) аналізатора промарковані речення вказуються, як параметр аналізатора, що і забезпечує тренування аналізатора. Процес тренування полягає в перегляді тегів кожного зі слів і збереження найбільш імовірних тегів (тегів, які найчастіше зустрічаються з кожним зі слів) у словнику, який зберігається в аналізаторі. Далі застосовується даний аналізатор для попередньо визначеного тексту і проводиться аналіз точності роботи створеного аналізатора

Поєднання (комбінування) аналізаторів

Одним з методів знаходження балансу між точністю аналізаторів та охопленням лексики є використання більш точних алгоритмів, коли можливо це зробити і повернення до алгоритмів з більшим охопленням лексики, коли це необхідно. Наприклад, можна комбінувати результати роботи біграм аналізатора, уніграм аналізатора та аналізатора по замовчуванню наступним чином:

1. Визначаємо теги за допомогою біграм аналізатора
2. Якщо біграм аналізатор не встановив тег для поточного слова – використовується унігам аналізатор.
3. Якщо уніграм аналізатор не встановив тег для поточного слова – використовується аналізатор по замовчуванню.

ВИКОНАННЯ ЗАВДАНЬ

Завдання 1. Здійсніть тренування юніграм аналізатора на основі частини корпуса, який відповідає першій або другій літері прізвища студента та виконайте аналіз тексту з частини корпуса, яка відповідає першій або другій літері імені студента. Результати поясніть. Чому для деяких слів не встановлені теги.

```
import nltk
from nltk.corpus import brown
sf_tagged_sents = brown.tagged_sents(categories='science_fiction')
sf_sents = brown.sents(categories = 'science_fiction')[:5]
unigram_tagger = nltk.UnigramTagger(sf_tagged_sents)
res = unigram_tagger.tag(sf_sents[2])
evaluation = unigram_tagger.evaluate(sf_tagged_sents)
print 'tagged sentence from Brown corpus:'
print sf_tagged_sents[1]
print 'sentence tagged by the program:'
print res
print 'Evaluation:'
print evaluation
>>>
tagged sentence from Brown corpus:
[("Self's", 'NN$'), ('integrity', 'NN'), ('was', 'BEDZ'), ('and', 'CC'), ('is',
'BEZ'), ('and', 'CC'), ('ever', 'RB'), ('had', 'HVD'), ('been', 'BEN'), ('.', '.
')]
sentence tagged by the program:
[('Mike', 'NP'), ('stopped', 'VBD'), ('to', 'TO'), ('cherish', 'VB'), ('all', 'A
BN'), ('his', 'PP$'), ('brother', 'NN'), ('selves', 'NNS'), (',', ','), ('the',
'AT'), ('many', 'AP'), ('threes-fulfilled', 'JJ'), ('on', 'IN'), ('Mars', 'NP'),
 (',', ','), ('corporate', 'JJ'), ('and', 'CC'), ('discorporate', 'VB'), (',', '
,'), ('the', 'AT'), ('precious', 'JJ'), ('few', 'AP'), ('on', 'IN'), ('Earth', '
NN-TL'), ('--', '--'), ('the', 'AT'), ('unknown', 'JJ'), ('powers', 'NNS'), ('of
', 'IN'), ('three', 'CD'), ('on', 'IN'), ('Earth', 'NN-TL'), ('that', 'CS'), ('w
ould', 'MD'), ('be', 'BE'), ('his', 'PP$'), ('to', 'TO'), ('merge', 'VB'), ('wit
h', 'IN'), ('and', 'CC'), ('cherish', 'VB'), ('now', 'RB'), ('that', 'CS'), ('at
', 'IN'), ('last', 'AP'), ('long', 'JJ'), ('waiting', 'VBG'), ('he', 'PPS'), ('g
rokked', 'VBD'), ('and', 'CC'), ('cherished', 'VBN'), ('himself', 'PPL'), ('.',
'.')]
Evaluation:
0.95190048376
```

У тексті, на основі якого тестувався аналізатор, зустрічались усі слова, які були у реченні (про це свідчить аналіз точночті роботи аналізатора). Проте, Юніграм аналізатор ставить тег None всім словам, які не зустрічаються в текстах.

Завдання 2. Прочитати файл допомого про морфологічний аналізатор на основі афіксів (help(nltk.AffixTagger)). Напишіть програму, яка викликає аналізатор на основі афіксів в циклі, з різними значеннями довжини афіксів і мінімальними довжинами слів. При яких значеннях можна отримати кращі результати.

```
mport nltk
rom nltk.corpus import brown
rown_tag = brown.tagged_sents(categories = 'romance')
ent = brown.sents(categories='religion')[1]
ffix_size = [-1, -2, -3]
tem_size = [2,3]
or i in affix_size:
   for j in stem_size:
       tagger = nltk.AffixTagger(brown_tag, affix_length = i, min_stem_size = j)
       analysis = tagger.tag(sent)
       evaluate = tagger.evaluate(brown_tag)
       print 'analyzed sentense:'
       print analysis
       print 'Tagger - ', tagger, 'affix_size = ', i, 'min_stem_length = ', j
       print 'Evaluation: ', evaluate
>>>
An analyzed sentence:
[('Just', 'NN'), ('what', 'NN'), ('is', None), ('meant', 'NN'), ('by', None),
('``', None), ('spirit', 'NN'), ("''", None), ('and', 'VBD'), ('by', None), (
'``', None), ('matter', 'NN'), ("''", None), ('?', None), ('?', None)]
Tagger <AffixTagger: size=37>  affix_size = -1  min_stem_length = 2
Evaluation: 0.206078089743

An analyzed sentence:
[('Just', 'NN'), ('what', 'NN'), ('is', None), ('meant', 'NN'), ('by', None),
('``', None), ('spirit', 'NN'), ("''", None), ('and', None), ('by', None), ('
``', None), ('matter', 'NN'), ("''", None), ('?', None), ('?', None)]
Tagger <AffixTagger: size=35>  affix_size = -1  min_stem_length = 3
Evaluation: 0.170217931507

An analyzed sentence:
[('Just', 'RB'), ('what', 'CS'), ('is', None), ('meant', 'NN'), ('by', None),
('``', None), ('spirit', 'NN'), ("''", None), ('and', None), ('by', None), ('
``', None), ('matter', 'NN'), ("''", None), ('?', None), ('?', None)]
Tagger <AffixTagger: size=262>  affix_size = -2  min_stem_length = 2
Evaluation: 0.241324155265

An analyzed sentence:
[('Just', None), ('what', None), ('is', None), ('meant', 'NN'), ('by', None),
('``', None), ('spirit', 'NN'), ("''", None), ('and', None), ('by', None), ('
``', None), ('matter', 'NN'), ("''", None), ('?', None), ('?', None)]
Tagger <AffixTagger: size=226>  affix_size = -2  min_stem_length = 3
Evaluation: 0.171817428808

An analyzed sentence:
[('Just', None), ('what', None), ('is', None), ('meant', 'JJ'), ('by', None),
('``', None), ('spirit', 'NN'), ("''", None), ('and', None), ('by', None), ('
``', None), ('matter', 'NN'), ("''", None), ('?', None), ('?', None)]
Tagger <AffixTagger: size=1023>  affix_size = -3  min_stem_length = 2
Evaluation: 0.206692182457
```

В результаті виконання програми бачимо, що найбільшу точність отримано у випадку, коли довжина афікса дорівнює 2 і мінімальна довжина кореня теж дорівнює 2. Ці показники зможемо використати в інших програмах.

Завдання 3. Здійсніть тренування біграм аналізатора на частинах корпуса з вправи 3.1 без backoff аналізатора. Перевірте його роботу. Що відбулося з продуктивністю аналізатора? Чому?

```
import nltk
from nltk.corpus import brown
sf_tagged_sents = brown.tagged_sents(categories='science_fiction')
sf_sents = brown.sents(categories = 'science_fiction')[:10]
bigram_tagger = nltk.BigramTagger(sf_tagged_sents)
res = bigram_tagger.tag(sf_sents[0])
evaluation = bigram_tagger.evaluate(sf_tagged_sents)
print 'tagged sentence from Brown corpus:'
print sf_tagged_sents[0]
print 'sentence tagged by the program:'
print res
print 'Evaluation:'
print evaluation
>>>
tagged sentence from Brown corpus:
[('Now', 'RB'), ('that', 'CS'), ('he', 'PPS'), ('knew', 'VBD'), ('himself', 'PPL
'), ('to', 'TO'), ('be', 'BE'), ('self', 'NN'), ('he', 'PPS'), ('was', 'BEDZ'),
('free', 'JJ'), ('to', 'TO'), ('grok', 'VB'), ('ever', 'QL'), ('closer', 'RBR'),
 ('to', 'IN'), ('his', 'PP$'), ('brothers', 'NNS'), (',', ','), ('merge', 'VB'),
 ('without', 'IN'), ('let', 'NN'), ('.', '.')]
sentence tagged by the program:
[('Now', 'RB'), ('that', 'CS'), ('he', 'PPS'), ('knew', 'VBD'), ('himself', 'PPL
'), ('to', 'TO'), ('be', 'BE'), ('self', 'NN'), ('he', 'PPS'), ('was', 'BEDZ'),
('free', 'JJ'), ('to', 'TO'), ('grok', 'VB'), ('ever', 'QL'), ('closer', 'RBR'),
 ('to', 'TO'), ('his', None), ('brothers', None), (',', None), ('merge', None),
('without', None), ('let', None), ('.', None)]
Evaluation:
0.843331029717
```

Коли у тексті зустрілося нове слово (his), аналізатор не може для нього встановити тег. Так само аналізатор не маркує наступне слово, навіть якщо воно зустрічалося при тренуванні, оскільки це слово в даних для тренування ніколи не зустрічалося після слова, тег якого None. Це приводить до того що і всі наступні слова в реченні не маркуються і точність роботи аналізатора падає.

Завдання 4. Дослідити наступні проблеми. що виникають при роботі з аналізатором на основі підстановок: що відбудеться з продуктивністю аналізатора, якщо опустити backoff аналізатор (дослідити на частині броунівського корпусу, яка відповідає першій або другій літері прізвища студента); на основі рис.1. та відповідного фрагмента програми встановити точку максимальної продуктивності незважаючи на розмір списку (об'єм оперативної пам'яті) і точку достатньої продуктивності при мінімальному розмірі списку.

При застосуванні backoff аналізатора продукривність вища, ніж при застосуванні тільки юніграм аналізатора.

```
import nltk
from nltk.corpus import brown
romance_tagged_sents = brown.tagged_sents(categories = 'romance')
religion_sents = brown.sents(categories = 'religion')[:10]
test_religion_tagged_sents = brown.tagged_sents(categories = 'religion')
fd = nltk.FreqDist(brown.words(categories = 'romance'))
most_freq_words = fd.keys()[:100]
cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories = 'romance'))
likely_tags = dict((word, cfd[word].max()) for word in most_freq_words)
unigram_tagger = nltk.UnigramTagger(model = likely_tags)
uni_result = unigram_tagger.tag(religion_sents[0])
uni_mark = unigram_tagger.evaluate(test_religion_tagged_sents)
print '     The result of unigram tagger:'
print uni_result
print '     Evaluation:', uni_mark
print
uni_def_tagger = nltk.UnigramTagger(model = likely_tags,
                                    backoff = nltk.DefaultTagger('NN'))
uni_def_result = uni_def_tagger.tag(religion_sents[0])
uni_def_mark = uni_def_tagger.evaluate(test_religion_tagged_sents)
print '     The result of unigram + default tagger:'
print uni_def_result
print '     Evaluation:', uni_def_mark
```

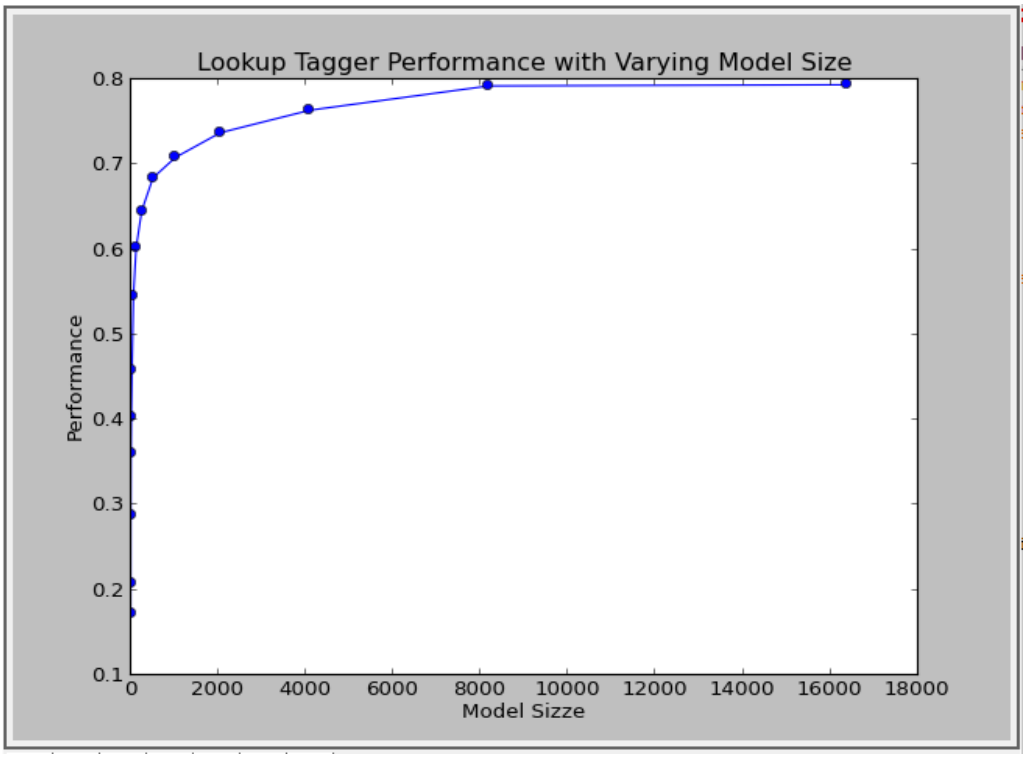
```
     The result of unigram tagger:
[('As', None), ('a', 'AT'), ('result', None), (',', ','), ('although', None), ('
we', 'PPSS'), ('still', None), ('make', None), ('use', None), ('of', 'IN'), ('th
is', 'DT'), ('distinction', None), (',', ','), ('there', 'RB'), ('is', 'BEZ'), (
'much', None), ('confusion', None), ('as', 'CS'), ('to', 'TO'), ('the', 'AT'), (
'meaning', None), ('of', 'IN'), ('the', 'AT'), ('basic', None), ('terms', None),
 ('employed', None), ('.', '.')]
     Evaluation: 0.461306124521

     The result of unigram + default tagger:
[('As', 'NN'), ('a', 'AT'), ('result', 'NN'), (',', ','), ('although', 'NN'), ('
we', 'PPSS'), ('still', 'NN'), ('make', 'NN'), ('use', 'NN'), ('of', 'IN'), ('th
is', 'DT'), ('distinction', 'NN'), (',', ','), ('there', 'RB'), ('is', 'BEZ'), (
'much', 'NN'), ('confusion', 'NN'), ('as', 'CS'), ('to', 'TO'), ('the', 'AT'), (
'meaning', 'NN'), ('of', 'IN'), ('the', 'AT'), ('basic', 'NN'), ('terms', 'NN'),
 ('employed', 'NN'), ('.', '.')]
     Evaluation: 0.582781288865
```

Точка максимальної продуктивності становить 0.796. Розмір списку приблизно 16350. Точку достатньої/максимальної продуктивності можна досягнути при розмірі списку приблизно 8150.

```
import nltk
from nltk.corpus import brown
def performance (cfd, wordlist):
    lt = dict((word, cfd[word].max()) for word in wordlist)
    baseline_tagger = nltk.UnigramTagger(model = lt,
                                         backoff = nltk.DefaultTagger('NN'))
    return baseline_tagger.evaluate(brown.tagged_sents(categories = 'religion'))

def display():
    import pylab
    words_by_freq = list(nltk.FreqDist(brown.words(categories = 'romance')))
    cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories = 'romance'))
    sizes = 2**pylab.arange(15)
    perfs = [performance(cfd, words_by_freq[:size]) for size in sizes]
    pylab.plot(sizes, perfs, '-bo')
    pylab.title('Lookup Tagger Performance with Varying Model Size')
    pylab.xlabel('Model Sizze')
    pylab.ylabel('Performance')
    pylab.show()
display()
```



Завдання 5. Знайдіть розмічені корпуси текстів для інших мов які вивчаєте або володієте (українська, польська, німецька, російська, італійська, японська). Здійсніть тренування та оцініть продуктивність роботи різних аналізаторів та комбінацій різних аналізаторів. Точність роботи аналізаторів порівняйте з точністю роботи аналізаторів для англійських корпусів. Результати поясніть.

Було проаналізовано розмічений корпус португальської мови. Для аналізу було використано різні аналізатори та різні комбінації аналізаторів. Найбільш продуктивною є комбінація “біграм аналізотор —> юніграм аналізатор —> аналізатор за замовчуванням -> триграм аналізатор. Найвища продуктивність становить 0.82, що є досить добрим результатом, але трохи нижчим від результатів аналізу англійських текстів (приблизно 0.9).

```
import nltk, re,pprint
from nltk.corpus import*
size=int(len(mac_morpho.tagged_sents())*0.9)
print size
train=mac_morpho.tagged_sents()[:size]
test=mac_morpho.tagged_sents()[size:]
u_t=nltk.UnigramTagger(train)
print u_t.evaluate(test)
b_t=nltk.BigramTagger(train)
print b_t.evaluate(test)
t_t=nltk.TrigramTagger
t_t=nltk.TrigramTagger(train)
print t_t.evaluate(test)
t0=nltk.DefaultTagger('NN')
t1=nltk.UnigramTagger(train,backoff=t0)
t2=nltk.BigramTagger(train,backoff=t1)
t3=nltk.TrigramTagger(train,backoff=t2)
print t3.evaluate(test)
>>>
46257
0.806675019738
0.208248944423
0.11788129484054786
0.8226459785108647
```

Завдання 6. Створити аналізатор по замовчуванню та набір юніграм і n-грам аналізаторів. Використовуючи backoff здійсніть тренування аналізаторів на частині корпуса з вправи 3.2. Дослідіть три різні комбінації поєднання цих аналізаторів. Перевірте точність роботи аналізаторів. Визначіть комбінацію аналізаторів з максимальною точністю аналізу. Змініть розмір даних на яких проводилось тренування. Повторіть експерименти для змінених даних для тренування. Результати порівняйти і пояснити.

Для визначення комбінації аналізторів з максимальною точність аналізу було протестовано різні комбінації аналізаторів. Крім того, було змінено розмір даних для тренування. Комбінація “біграм аналізатор —> юніграм аналізатор —> аназілатор по замовчуванню” виявилася найбільш продуктивною.

```
import nltk
default_tagger = nltk.DefaultTagger('NN')
brown_a = nltk.corpus.brown.tagged_sents(categories='belles_lettres')
unigram_tagger = nltk.UnigramTagger(brown_b)
bigram_tagger = nltk.BigramTagger(brown_b, cutoff=0)
t0=nltk.DefaultTagger('NN')
t1=nltk.UnigramTagger(brown_b, backoff=t0)
t2=nltk.BigramTagger(brown_b, backoff=t1)
print nltk.tag.accuracy(t2, brown_b)
brown_a = nltk.corpus.brown.tagged_sents(categories='adventure')
print nltk.tag.accuracy(t2, brown_a)
t0=nltk.UnigramTagger(brown_b)
t1=nltk.BigramTagger(brown_b, backoff=t0)
t2=nltk.DefaultTagger('NN')
print nltk.tag.accuracy(t2, brown_a)
t0=nltk.DefaultTagger('NN')
t1=nltk.BigramTagger(brown_b, backoff=t0)
t2=nltk.UnigramTagger(brown_b, backoff=t2)
print nltk.tag.accuracy(t2, brown_a)
t0=nltk.DefaultTagger('NN')
t1=nltk.UnigramTagger(brown_a, backoff=t0)
t2=nltk.BigramTagger(brown_a, backoff=t1)
print nltk.tag.accuracy(t2, brown_a)
t0=nltk.DefaultTagger('NN')
unigram_tagger = nltk.UnigramTagger(brown_b)
unigram_tagger = nltk.UnigramTagger(brown_a)
brown_n = nltk.corpus.brown.tagged_sents(categories='news')
brown_r = nltk.corpus.brown.tagged_sents(categories='romance')
t0=nltk.DefaultTagger('NN')
t1=nltk.UnigramTagger(brown_n, backoff=t0)
t2=nltk.BigramTagger(brown_n, backoff=t1)
print nltk.tag.accuracy(t2, brown_a)
t0=nltk.DefaultTagger('NN')
t1=nltk.UnigramTagger(brown_r, backoff=t0)
t2=nltk.BigramTagger(brown_r, backoff=t1)
print nltk.tag.accuracy(t2, brown_a)
t0=nltk.DefaultTagger('NN')
t1=nltk.BigramTagger(brown_r, backoff=t0)
t2=nltk.UnigramTagger(brown_r, backoff=t1)
print nltk.tag.accuracy(t2, brown_a)
0.96992420390996903
0.85590262755617086
0.11610567909780509
0.84117850653283721
0.97332064261198115
0.83780392835510942
```

Завдання 7. Прочитати стрічку документування функції demo Brill аналізатора. Здійснити експерименти з різними значення параметрів цієї функції. Встановити який взаємозв'язок є між часом тренування (навчання аналізатора) і точністю його роботи.

Було проведено експерименти з різними даними параметрів функції і визначено, що точність аналізатора вища, якщо збільшити кількість речень і зменшити максимальну кількість правил.

```
>>> nltk.tag.brill.demo(num_sents = 1500, max_rules = 150)
Loading tagged data...
Done loading.
Training unigram tagger:
    [accuracy: 0.837607]
Training bigram tagger:
    [accuracy: 0.842965]
Training Brill tagger on 1200 sentences...
Finding initial useful rules...
    Found 6515 useful rules.

           B      |
   S   F   r   O  |        Score = Fixed - Broken
   c   i   o   t  |  R     Fixed = num tags changed incorrect -> correct
   o   x   k   h  |  u     Broken = num tags changed correct -> incorrect
   r   e   e   e  |  l     Other = num tags changed incorrect -> incorrect
   e   d   n   r  |  e
------------------+-------------------------------------------------------
   9  11   2   0  | WDT -> IN if the tag of words i+1...i+2 is 'NNP'
   8  11   3   2  | WDT -> IN if the tag of words i+1...i+2 is 'NN'
   7   9   2   0  | IN -> RB if the text of words i+1...i+2 is 'as'
   4   4   0   0  | RB -> IN if the tag of the preceding word is 'RB',
                  |   and the tag of the following word is 'PRP'
   4   4   0   0  | RBR -> JJR if the tag of the following word is
                  |   'NN'
   4   4   0   0  | WDT -> IN if the tag of words i+1...i+2 is 'JJ'
   4   4   0   0  | WDT -> IN if the tag of the preceding word is
                  |   'NN', and the tag of the following word is 'PRP'
   3   3   0   0  | RB -> IN if the tag of the preceding word is 'NN',
                  |   and the tag of the following word is 'DT'
   3   3   0   0  | WDT -> IN if the tag of words i+1...i+3 is 'VBG'
   3   3   0   0  | NNS -> NN if the text of the preceding word is
                  |   'one'

Brill accuracy: 0.842072
Done; rules and errors saved to rules.yaml and errors.out.
```

```
>>> nltk.tag.brill.demo(num_sents = 2500, max_rules = 50)
Loading tagged data...
Done loading.
Training unigram tagger:
    [accuracy: 0.858251]
Training bigram tagger:
    [accuracy: 0.865856]
Training Brill tagger on 2000 sentences...
Finding initial useful rules...
    Found 12300 useful rules.

           B      |
   S   F   r   O  |        Score = Fixed - Broken
   c   i   o   t  |  R     Fixed = num tags changed incorrect -> correct
   o   x   k   h  |  u     Broken = num tags changed correct -> incorrect
   r   e   e   e  |  l     Other = num tags changed incorrect -> incorrect
   e   d   n   r  |  e
------------------+-------------------------------------------------------
  13  19   6   0  | WDT -> IN if the tag of words i+1...i+2 is 'DT'
  13  19   6   1  | IN -> RB if the text of words i+1...i+2 is 'as'
  10  10   0   0  | WDT -> IN if the tag of the preceding word is
                  |   'NN', and the tag of the following word is 'NNP'
   8  11   3   0  | WDT -> IN if the tag of words i+1...i+2 is 'NNS'
   7   7   0   0  | WDT -> IN if the tag of the preceding word is
                  |   'NN', and the tag of the following word is 'PRP'
   6   6   0   0  | RBR -> JJR if the tag of the following word is
                  |   'NN'
   5   5   0   0  | WDT -> IN if the tag of the preceding word is
                  |   'NNS', and the tag of the following word is
                  |   'PRP'
   4   4   0   1  | WDT -> IN if the tag of words i+1...i+3 is 'VBG'
   3   3   0   0  | RBR -> JJR if the tag of the following word is
                  |   'NNS'
   3   3   0   0  | IN -> RB if the text of the preceding word is
                  |   'month', and the text of the following word is
                  |   '.'
   3   4   1   0  | IN -> WDT if the text of the preceding word is
                  |   'in', and the text of the following word is
                  |   'the'
   3   3   0   0  | JJ -> NNP if the text of the following word is
                  |   'Union'
   3   3   0   0  | NNS -> NN if the text of the preceding word is
                  |   'one'
   3   3   0   0  | RBR -> JJR if the text of words i-3...i-1 is
                  |   '*T*-1'
   3   3   0   0  | RP -> IN if the text of the following word is 'of'
   3   3   0   0  | RP -> RB if the text of words i-3...i-1 is 'were'
   3   3   0   0  | VBP -> VB if the text of words i-2...i-1 is "n't"

Brill accuracy: 0.867985
Done; rules and errors saved to rules.yaml and errors.out.
>>>
```

ВИСНОВОК

У цій лабораторній роботі я вивчила основи структурного програмування мовою Python та знайомилася з автоматичним морфологічним аналізом в NLTK.