МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

*Кафедра “Системи автоматизованого проектування”*



Звіт

до лабораторної роботи №10

на тему: «ВИВЧЕННЯ БІБЛІОТЕКИ ПРИКЛАДНИХ ПРОГРАМ NLTK, ДЛЯ ОПРАЦЮВАННЯ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ.

АВТОМАТИЧНИЙ МОРФОЛОГІЧНИЙ АНАЛІЗ (частина2)*»*

з дисципліни “Комп’ютерна лінгвістика”

Виконала:

студентка групи ПРЛм-11

Павлів І.О.

Прийняв:

викладач

Дупак Б.П.

Львів-2015

**Мета роботи:**

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

**КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

## Розробка автоматичного морфологічного аналізатора

При виконанні цієї лабораторної роботи буде необхідність використовувати морфологічно розмічені корпуси текстів. При роботі з корпусом *Brown* доступ до речень з копусу слова в яких промарковані тегами морфологічних характеристик можна отримати наступним чином.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> from nltk.corpus import brown**  **>>> brown\_tagged\_sents = brown.tagged\_sents(categories='news')**  **>>> brown\_sents = brown.sents(categories='news')** | |

### 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. Для визначення такого тега використаємо просту програму:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> tags = [tag for (word, tag) in brown.tagged\_words(categories='news')]**  **>>> nltk.FreqDist(tags).max()**  **'NN'** | |

Тепер можна створити аналізатор *default\_tagger* , який позначить всі слова як *NN*.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> raw = 'I do not like green eggs and ham, I do not like them Sam I am!'**  **>>> tokens = nltk.word\_tokenize(raw)**  **>>> default\_tagger = nltk.DefaultTagger('NN')**  **>>> default\_tagger.tag(tokens)**  **[('I', 'NN'), ('do', 'NN'), ('not', 'NN'), ('like', 'NN'), ('green', 'NN'), ('eggs', 'NN'), ('and', 'NN'), ('ham', 'NN'), (',', 'NN'), ('I', 'NN'), ('do', 'NN'), ('not', 'NN'), ('like', 'NN'), ('them', 'NN'), ('Sam', 'NN'), ('I', 'NN'), ('am', 'NN'), ('!', 'NN')]** | |

Зауважимо, що аналізатор це генератор тегів слів. Роздрукувати його безпосередньо не можна, але можна перетворити у список і тоді роздрукувати, як показано у попередньому прикладі. Генератор ми можемо використати тільки один раз, але якщо результати його роботи зберегти у списку то їх можна використовувати і в подальшому.

Наведений метод маркування є дуже простий, але результат його роботи дуже поганий. Тільки приблизно одна восьма слів буде промаркована правильно в типовому корпусі.:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> default\_tagger.evaluate(brown\_tagged\_sents)**  **0.13089484257215028** | |

### Автоматичний морфологічний аналізатор на основі регулярних виразів

Аналізатор побудований на регулярних виразах встановлює відповідність тегів до слів на основі їх відповідності певним шаблонам. Наприклад можна передбачити, що кожне слово, яке закінчується на *–ed* це є *past participle* в дієсловах, а будь-яке слово, яке закінчується на *‘s* це є присвійний іменник. Подібні правила можна описати, як список регулярних виразів:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> patterns = [**  **... (r'.\*ing$', 'VBG'), # gerunds**  **... (r'.\*ed$', 'VBD'), # simple past**  **... (r'.\*es$', 'VBZ'), # 3rd singular present**  **... (r'.\*ould$', 'MD'), # modals**  **... (r'.\*\'s$', 'NN$'), # possessive nouns**  **... (r'.\*s$', 'NNS'), # plural nouns**  **... (r'^-?[0-9]+(.[0-9]+)?$', 'CD'), # cardinal numbers**  **... (r'.\*', 'NN') # nouns (default)**  **... ]** | |

Потрібно звернути увагу на те, що при застосуванні такого шаблону відповідності будуть знайдені в порядку запису регулярних виразів. Тепер можна налаштувати аналізатор і використати його для аналізу деякого тексту.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> regexp\_tagger = nltk.RegexpTagger(patterns)**  **>>> regexp\_tagger.tag(brown\_sents[3])**  **[('``', 'NN'), ('Only', 'NN'), ('a', 'NN'), ('relative', 'NN'), ('handful', 'NN'), ('of', 'NN'), ('such', 'NN'), ('reports', 'NNS'), ('was', 'NNS'), ('received', 'VBD'), ("''", 'NN'), (',', 'NN'), ('the', 'NN'), ('jury', 'NN'), ('said', 'NN'), (',', 'NN'), ('``', 'NN'), ('considering', 'VBG'), ('the', 'NN'), ('widespread', 'NN'), ...]**  **>>> regexp\_tagger.evaluate(brown\_tagged\_sents)**  **0.20326391789486245** | |

Останній регулярний вираз «.\*» вказує на те, що всі слова, які не відповідають попереднім регулярним виразам будуть промарковані, як іменники. Цей регулярний вираз еквівалентний до аналізатора позамовчуванню.

### The Lookup Tagger Пошуковий морфологічний аналізатор

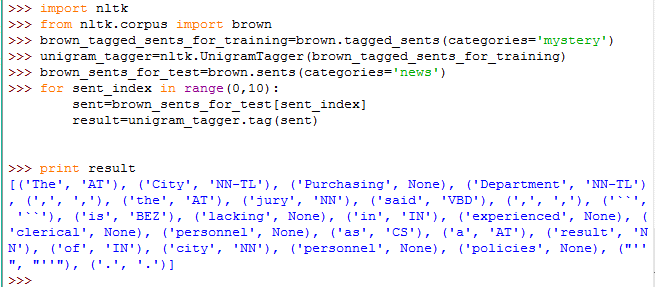
Слова, які мають високу частоту (часто зустрічаються в текстах), необов’язково мають тег *nn*. Спробуємо знайти найчастотніші слова та відповідні до них теги. Таку інформацію можна буде в подальшому використати, як модель для пошукового аналізатора "lookup tagger" в NLTK UnigramTagger.

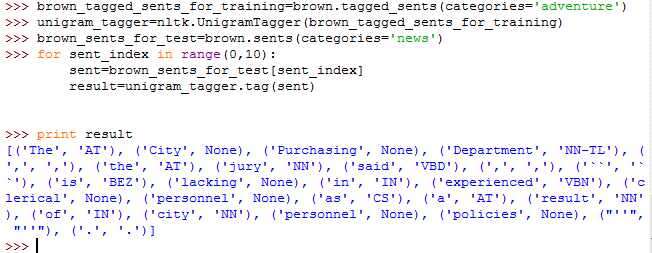
Наступна програма обробляє послідовність речень, підраховує частоту слів і виводить на екран 100 слів, які найчастіше зустрічаються :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> fd = nltk.FreqDist(brown.words(categories='news'))**  **>>> most\_freq\_words = fd.keys()[:100]**  **>>> most\_freq\_words**  **['the', ',', '.', 'of', 'and', 'to', 'a', 'in', 'for', 'The', 'that', '``', 'is', 'was', "''", 'on', 'at', 'with', 'be', 'by', 'as', 'he', 'said', 'his', 'will', 'it', 'from', 'are', ';', 'has', 'an', '--', 'had', 'who', 'have', 'not', 'Mrs.', 'were', 'this', 'would', 'which', 'their', 'been', 'they', 'He', 'one', 'I', 'its', 'but', 'or', 'more', ')', 'Mr.', 'up', '(', 'all', 'last', 'out', 'two', ':', 'other', 'new', 'first', 'year', 'than', 'A', 'about', 'there', 'when', 'home', 'after', 'In', 'also', 'over', 'It', 'into', 'no', 'But', 'made', 'her', 'only', 'years', 'time', 'three', 'them', 'some', 'can', 'New', 'him', 'state', '?', 'any', 'President', 'could', 'before', 'week', 'under', 'against',**  **'we', 'now']** | |

**Варіант №13**

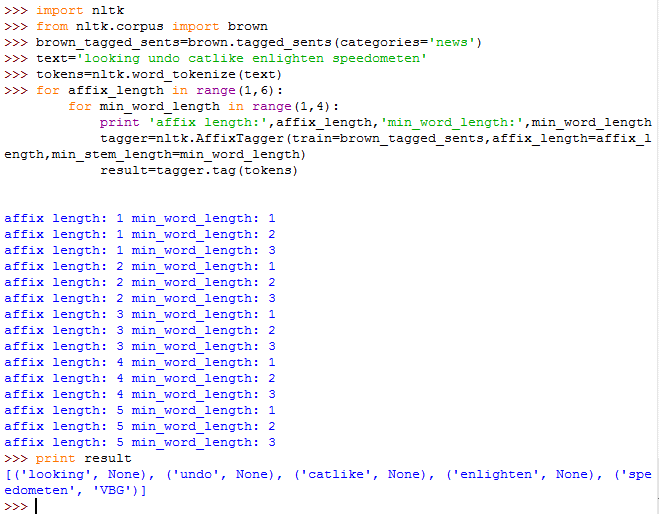
1. Здійсніть тренування юніграм аналізатора на основі частини корпуса, який відповідає першій або другій літері прізвища студента та виконайте аналіз тексту з частини корпуса, яка відповідає першій або другій літері імені студента. Результати поясніть. Чому для деяких слів не встановлені теги.



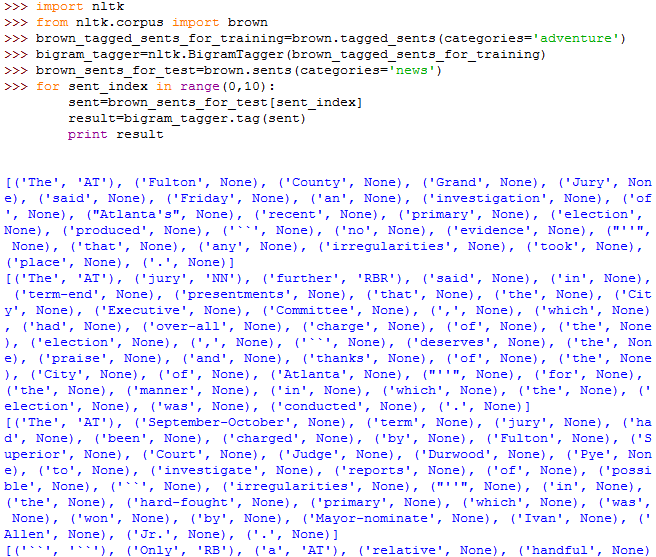


Деякі слова позначені тегом None, бо цих слів не було у реченнях, які були використані для тренування.

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.

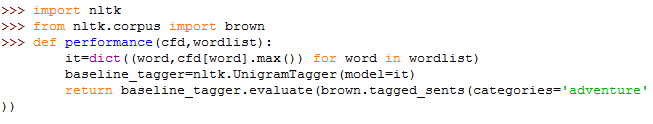


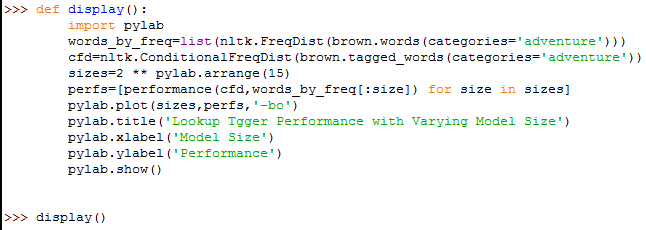
3. Здійсніть тренування біграм аналізатора на частинах корпуса з вправи 3.1 без backoff аналізатора. Перевірте його роботу. Що відбулося з продуктивністю аналізатора? Чому?

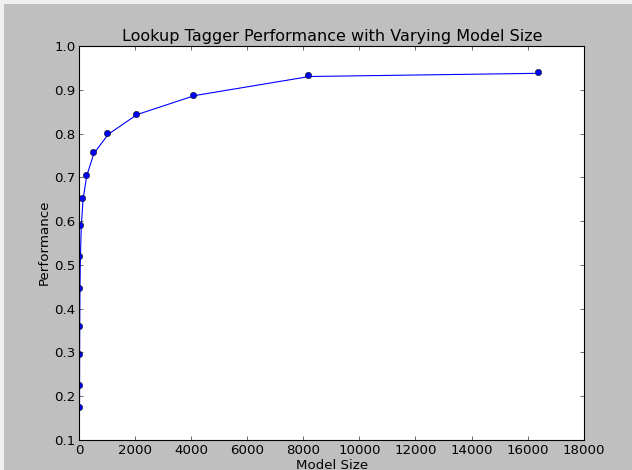


При аналізі речення з даних для тренування аналізатор промаркував кожне зі слів, але при аналізі невідомого тексту більшість слів не промарковано. Як тільки в тексті зустрілося нове слово, аналізатор не може для нього встановити тег. Так само аналізатор не маркує наступне слово навіть , якщо воно зустрічалося 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 надзвичайно низька.

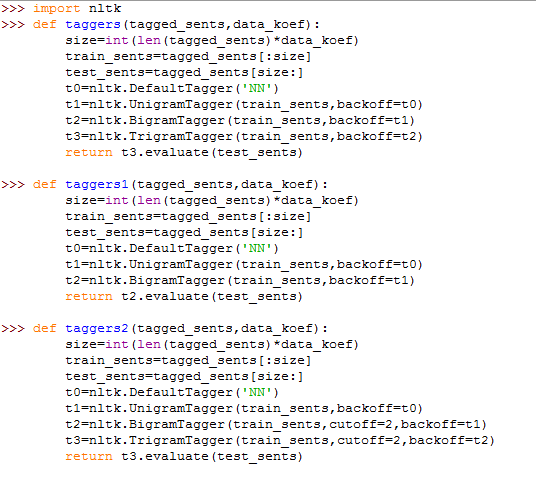
4. Дослідити наступні проблеми. що виникають при роботі з аналізатором на основі підстановок: що відбудеться з продуктивністю аналізатора, якщо опустити backoff аналізатор (дослідити на частині броунівського корпусу, яка відповідає першій або другій літері прізвища студента); на основі рис.1. та відповідного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

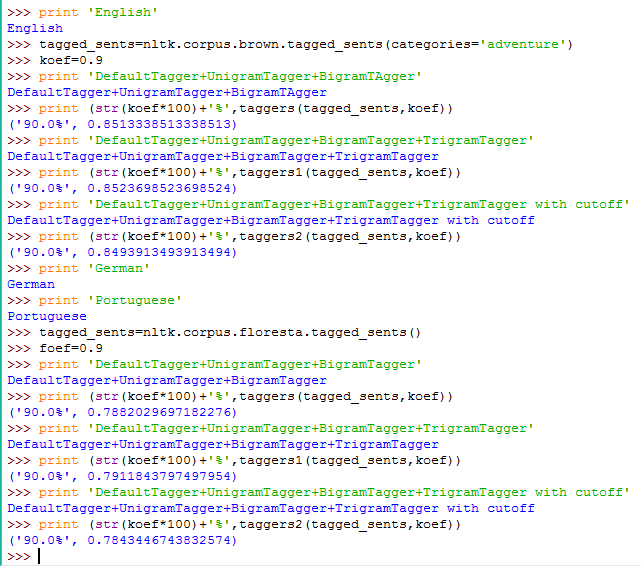


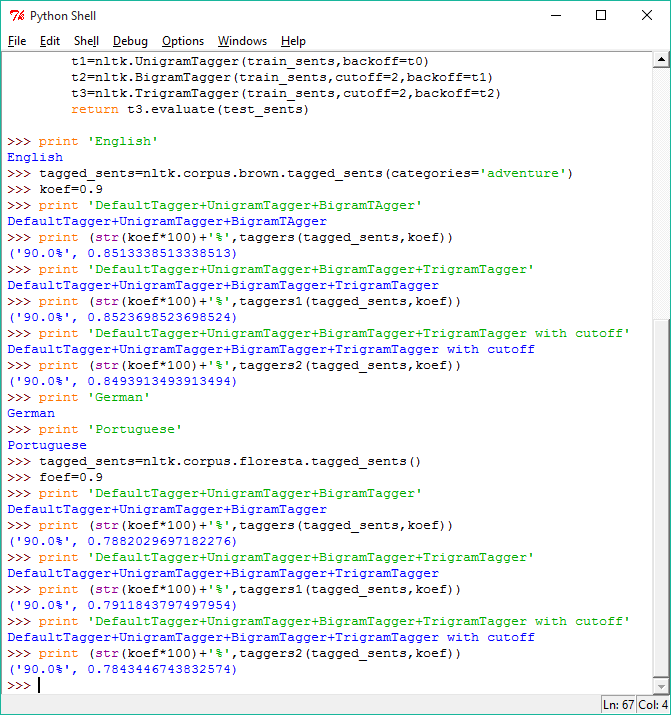




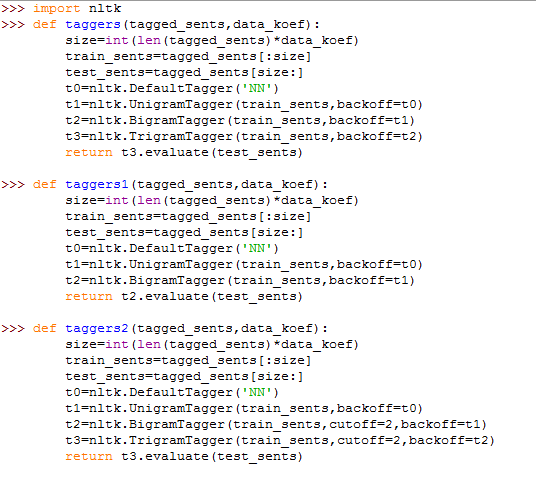
5. Знайдіть розмічені корпуси текстів для інших мов які вивчаєте або володієте (українська, польська, німецька, російська, італійська, японська). Здійсніть тренування та оцініть продуктивність роботи різних аналізаторів та комбінацій різних аналізаторів. Точність роботи аналізаторів порівняйте з точністю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

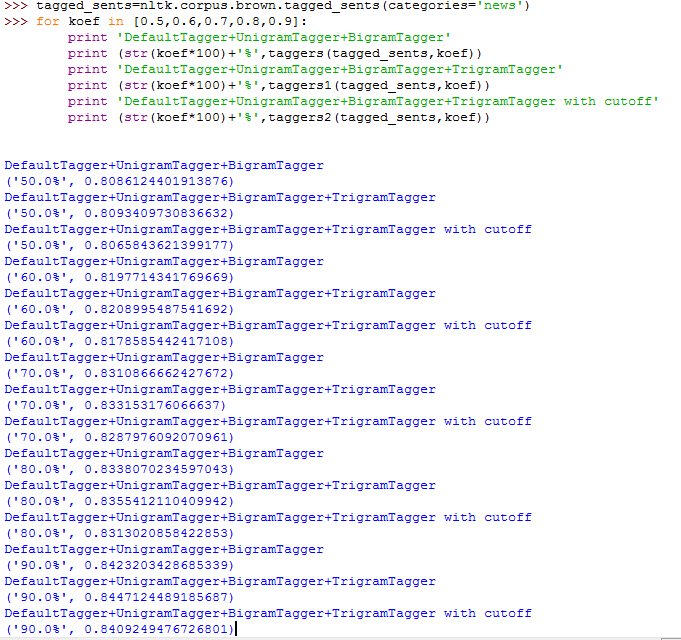




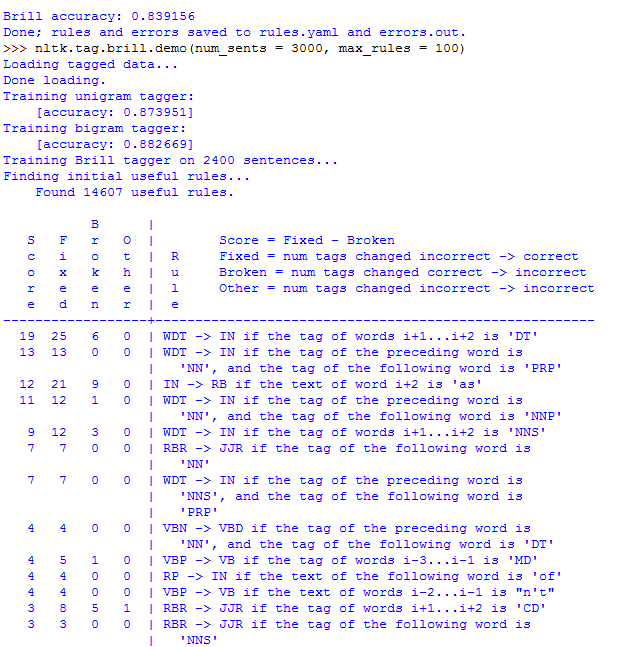
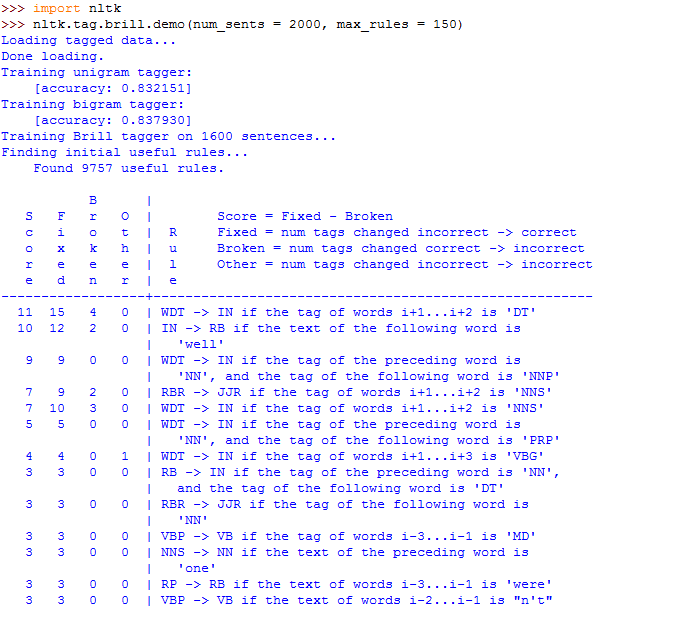


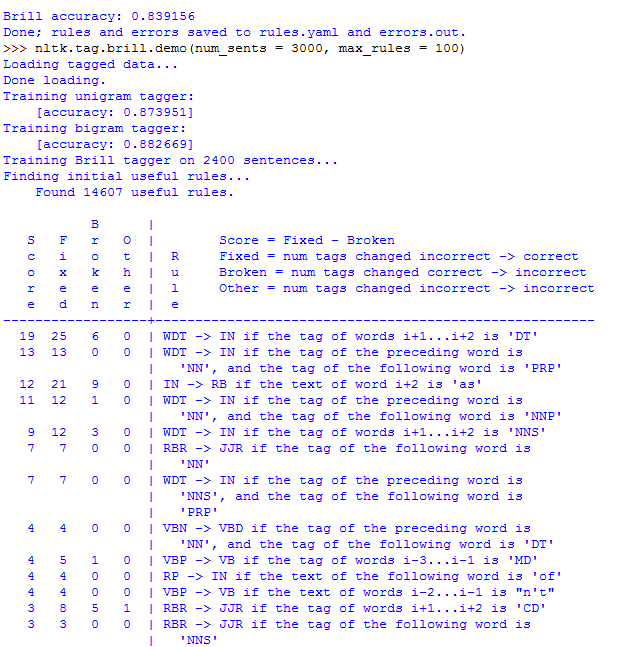
6. Створити аналізатор по замовчуванню та набір юніграм і n-грам аналізаторів. Використовуючи backoff здійсніть тренування аналізаторів на частині корпуса з вправи 3.2. Дослідіть три різні комбінації поєднання цих аналізаторів. Перевірте точність роботи аналізаторів. Визначіть комбінацію аналізаторів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

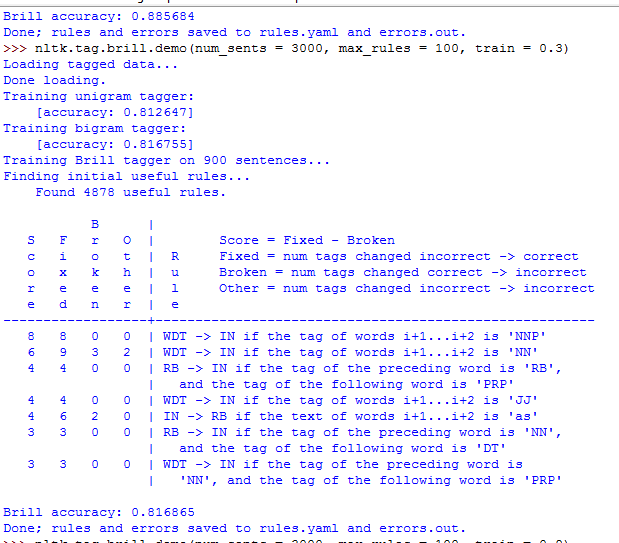


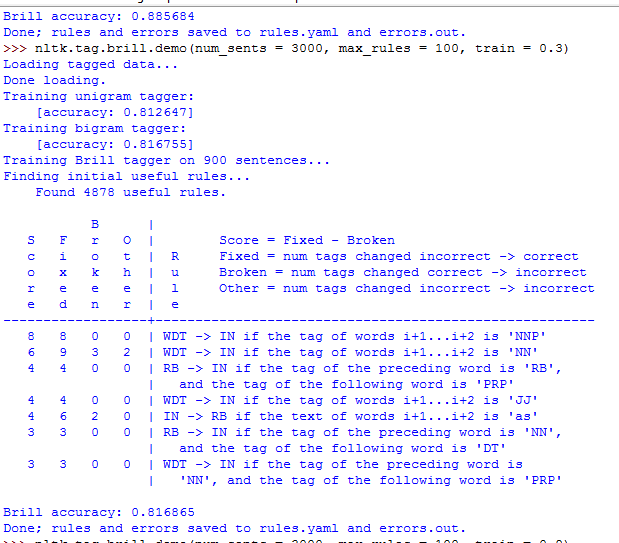


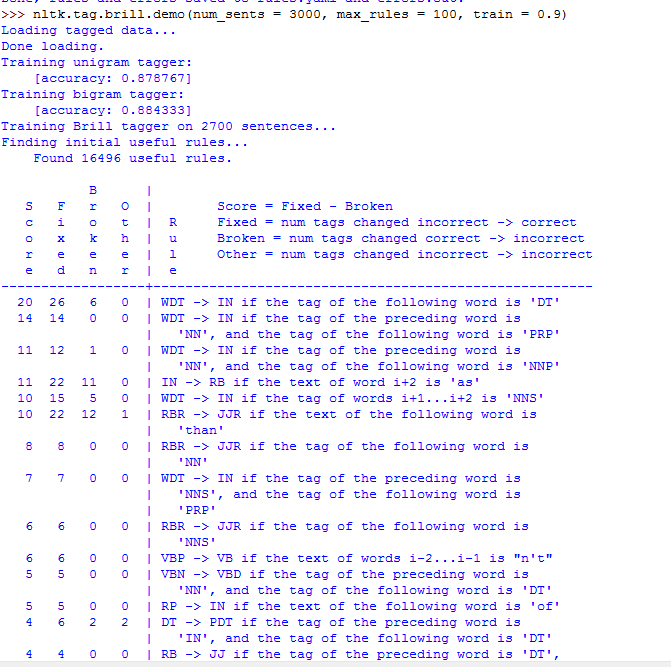
7. Прочитати стрічку документування функції demo Brill аналізатора. Здійснити експерименти з різними значення параметрів цієї функції. Встановити який взаємозв’язок є між часом тренування (навчання аналізатора) і точністю його роботи.

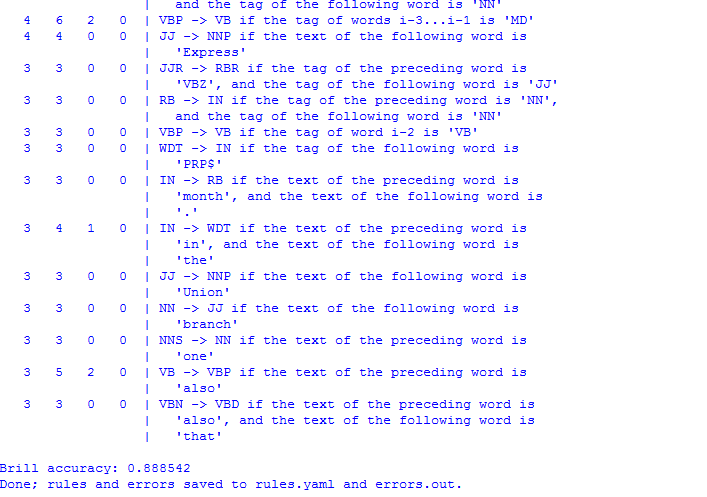


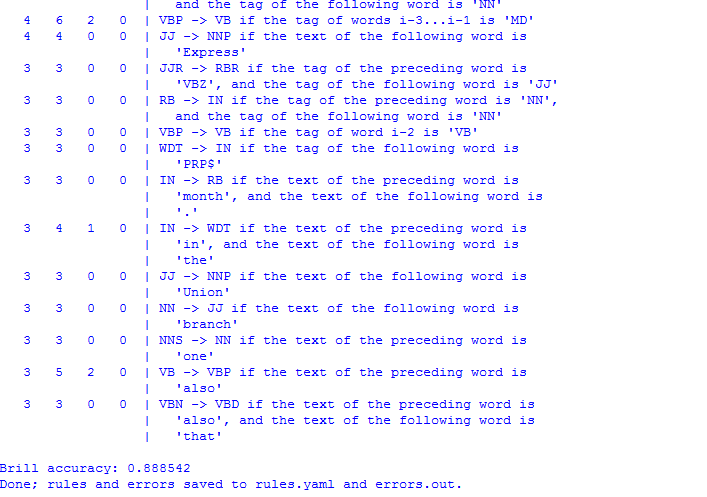












Чим більше речень, тим точність виконання буде більшою. Якщо встановлений вищий показник точності, то час тренування програми більший, а продукривність аналізаторів вища.

**Висновок:** на цій лабораторній роботі я вивчила основи структурного програмування мовою Python та ознайомилася з автоматичним морфологічним аналізом.