Московский Авиационный Институт
(Научный Исследовательский Институт)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Отчёт по лабораторной работе №1 по курсу «Обработка текстов на естественном языке»

Москва 2020

ЛР1: Токенизация

Постановка задачи:

Нужно реализовать процесс разбивания текстов документов на токены, который потом будет использоваться при индексации. Для этого потребуется выбрать правила по которым текст делится на токены. Необходимо описать их в отчете, указать достоинства и недостатки выбранного метода. Привести примеры токенов, которые были выделены неудачно, объяснить, как можно было бы поправить правила, чтобы исправить найденные проблемы.

В результатах выполнения работы нужно указать следующие статистические данные:

- Количество токенов.
- Среднюю длину токена.

Кроме того, нужно привести время выполнения программы, указать зависимость времени от объема входных данных. Указать скорость токенизации в расчете на килобайт входного текста. Является ли эта скорость оптимальной? Как ее можно ускорить?

Этапы выполнения:

- Выбрать инструменты для работы с json и строками
- Составить набор правил, для выделени токенов
- Выделить токены
- Анализ качества полученного решения
- Подсчет статистической информации
- Измерение и оценка оптимальности скорости выполнения
- Возможности ускорения

Выполнение:

В качестве инструментов используем:

- модуль gc для контроля над сборщиком мусора при замерах
- модуль re для ругулярных выражений
- модуль time для замеров времени

```
@logging('Tokenization...')
def tokenization(docs):
    pattern = r"\w+(?:'\w+)?|[^\w\s]"
    return map(lambda doc: re.findall(pattern, doc), docs)
```

Аналогично можно разбивать входную последовательность по знакам препинания используя операцию *split*, вместо *findall*, но тогда в выдаче могут возникнуть пустые строки.

Подсчет статистической информации

Всего в тексте: 36937361 Средняя длина токена в тексте: 6.1050526739207445

Анализ качества полученного решения

Полученное решение лишь выделяет токены из всего текста - которые совпадают с заданным шаблоном. Представляется это все конечным автоматом

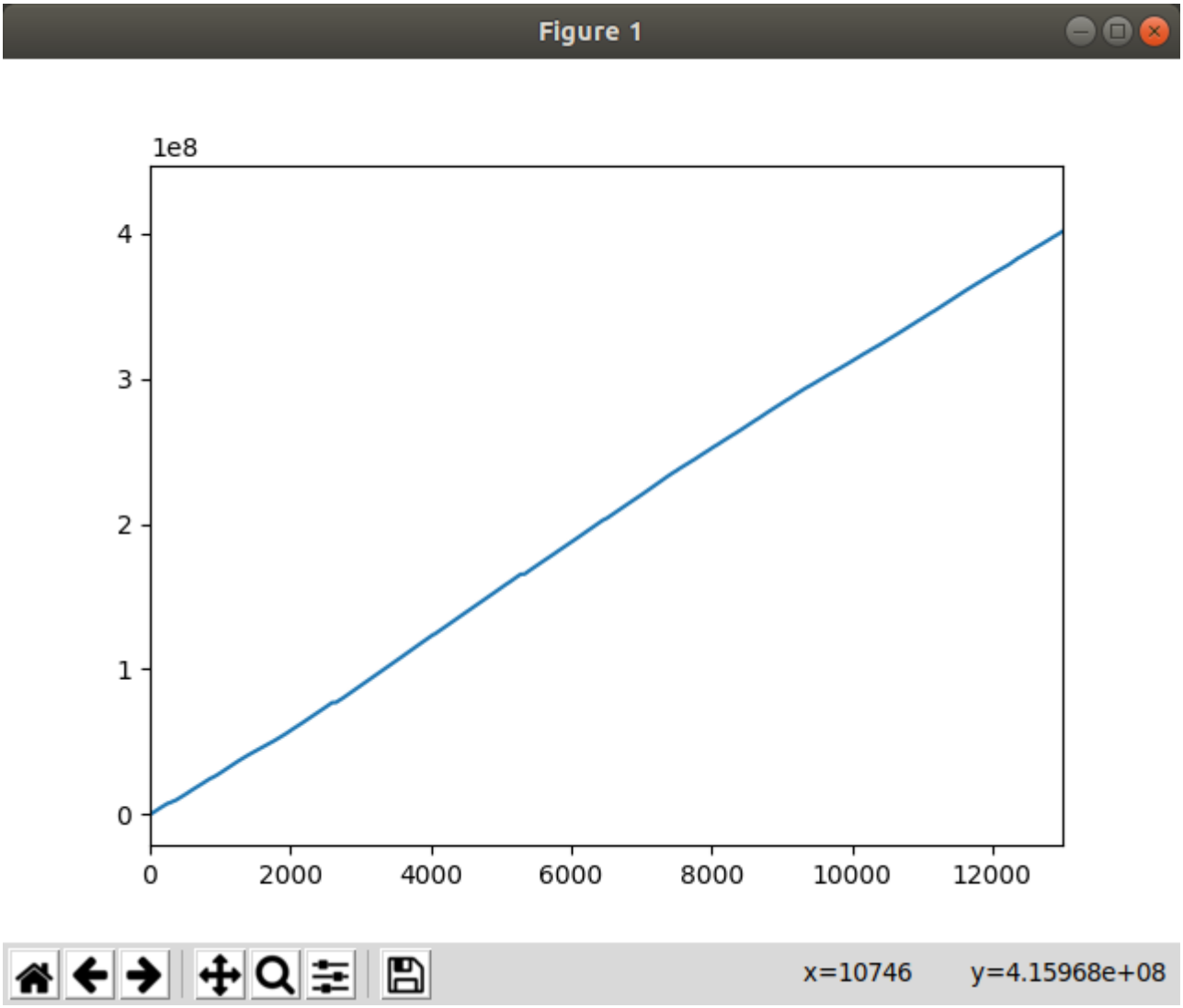
Очевидно что данным подходом мы не сможем выделить токены которые поразумевают некоторую структуру: телефоны, email, сложные слова с дефисом и много другое. В силу направленности корпуса документов. В нем втречается много примеров кода. Которые достаточно сложно разбивать на токены. Ниже приведены подобные примеры:

[8', '926', ‘313’, ‘50’, ‘69’, 'int', 'main', '(', ')', '{', '}', ';']

Однако перед нами не стоит задача точно определить токены, поэтому невилировать данную проблему можно путем разбивания поискового запроса на токены таким же способом как и сами документы.

Измерение и оценка оптимальности скорости выполнения

- loading direct index - 1661.58ms
- tokenization(simple) - 17924.86ms
- reduce stop words – 204362.08ms
- save tokens – 8662.98ms
- load tokens – 3902.29ms
- Скорость токенизации в расчете на kb/mb: 98.2846 kb/ms → 95.97 mb/s



Зависимость данных от времени (b/ms)

Из данных видно, что скорость токенизации быстрее чем скорость чтения и записи с файла, что дает возможность реализовать потоковый алгоритм токенизации На данном этапе обращение к данным шло последовательно по всем ключам, получается что в данной задаче целесообразней иметь другую структуру массив, проходя по которому обрабатывать соответственно поступающий на вход символ и решать что с ним делать исходя из состояния конечного автомата. Так же добален блок удаления стоп-слов, который пока работает дольше чем сама токенизация, т.к поиск в словаре стоп слов – долгая операция.

Вывод

В ходе выполнения лабораторной работы была реализована функция выделения токенов из входных документов, проанализировано качество и скорость предложенного решения, приведено сравнение и возможности улучшения.