### Московский авиационный институт

**Факультет прикладной математики и физики**

**Лабораторная работа №1**

**по курсу:**

**«Обработка естественно-языковых текстов»**

**по теме:**

**«Токенизация»**

**2 семестр**

Студент: Ахмед С. Х.

Преподаватель: Калинин А. Л.

Группа: 8О-106М

**Москва, 2019 г**

Постановка задачи

Нужно реализовать процесс разбиения текстов документов на токены, который потом будет использоваться при индексации. Для этого потребуется выработать правила, по которым текст делится на токены. Необходимо описать их в отчёте, 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 объёма входных данных. Указать скорость токенизации в расчёте на килобайт входного текста. Является ли эта скорость оптимальной? Как её можно ускорить?

Цель работы

Построить токенизатор, получить список токенов текста, разобраться в возможных проблемах, которые могут возникнуть при токенизации.

Оборудование:

Компьютер HP Omen 15 под управлением операционной системы Windows 10, Intel Core i5-7300HQ 2.50 GHz, 12 Gb RAM

Программное обеспечение

|  |  |
| --- | --- |
| Язык программирования | Python 3.6 |
| Среда программирования | Anaconda, Jupyter Notebook |
| Использованные сторонние модули | re |
| Альтернативный модель решающий поставленную задачу | Nltk(результаты его работы представлены в лабораторной работе 1 по Инфопоиску) |

Статистические данные по полученному результату

Эталоном по количеству n-gram может служить информация полученная в лабортаторной работе 1 по информационному поиску(там были рассмотрены и юниграммы и биграммы, которые в данной лабораторной работе для слов с типа: (“из-за”,Санкт-Петербург) считаются одним токеном). Судя по этим выкладкам, а также, по наблюдению выходного файла стоит сказать, что за исключением некоторых проблем связанных с парсингом дамп файлов( почему-то genism очень не любит ударения и заменяет его пробелом) и некоторых особенностей подходов, информация будет совпадать, так как для юниграмм в первой лабораторной работе я применял ту же идею, что и данной лабораторной работе, выделение целых слов(за исключением обработки некоторых уникальных паттернов данных).

Время осуществления данной операции

|  |  |
| --- | --- |
| **Время записи в один файл**(в это время входит обход файлов директории открытие файла) | 6 min 49s |
| Время записи в разные файлы(в это время входит обход файлов директории, создание файла, открытие файла) | 13min 32s |
| Итоговая скорость операции: | О(Скорость записи на диск + constant),где в константу включены операции обхода директории и открытия файла |
| Временная сложность | O(|D|/(скорость записи + constant)) |
| Сложность по памяти | O(количество токенов в файле) |

Идея алгоритма

В основе моего подхода лежит идея разделения текста на слова (разбиение по пунктуации). Я дополнил этот метод паттернами(шаблонами) того, что я буду называть словом:

1. r"(\w+)(-)(\w+)" - данное регуля

|  |  |
| --- | --- |
| r"^\s\*(?:\+?(\d{1,3}))?[-. (]\*(\d{3})[-. )]\*(\d{3})[-. ]\*(\d{4})(?: \*x(\d+))?\s\*$" | данное регулярное выражение охватывает наиболее распространённые номера телефонов.(в моем случае, как выяснилось их у меня практически нет) |
| r"(\w+)(-)(\w+)" | Слова разделенные тире. (из-за, Санкт-Петербург).(Не различает случай, когда тире является знаком препинания) |
| r'(\d{2,4})(-|/)(\d{2})(-|/)(\d{2,4})' | Даты в форматах ДДММГГ, ГГММДД, ММДДГГ,ГГДДММ. Не провереяет корректность даты |
| r"\w+" | Слово в классическом понимании(от пробела до пробела, состоящее из букв) |
| r"\d+" | Числа |

Вообще проще было бы использовать только сплит по пробелам и подчищать точки и запятые. Но на время написания программы я об этом не подумал.

Возможные проблемы:

1. Данный подход не учитывает многообразия паттернов в данных, что добавляет сложности в токенизации
2. Этот подход, что и сплит по пробелам не учитывает специфические названия (Бахчи У)
3. Чувствительность к формату дат (рассмотрены лишь некоторые аспекты), также не учитывается неккоректность данных
4. Чувствительность к формату чисел
5. Есть вероятность повторения чисел, токенов (номера, даты состоят из чисел)
6. Не учитывается другие возможные случаи токенов
7. Примитивность

В случае уникальных токенов можно ввести либо словарь, в котором будут храниться такие токены, либо использовать метод очень похожий на TF-IDF, то есть подсчитаем кол-во биграм триграмм, и найдем очень редко встречающиеся биграммы и триграммы и объединим их в один токен.

Мой подход расширяем: если мы находим довольно частый паттерн, который мы не учли, мы можем внести его регулярное выражение.

Примитивность моего подхода заключается в том, что я прохожу несколько раз по тексту чтобы найти токены удовлетворяющие требованию. По-хорошему, стоило бы запоминать позицию в котором встретился один из токенов и начинать поиск независимых шаблонов с этой позиции.

В ходе работы я столкнулся с проблемой форматирования данных, genism заменил ударения на пробелы, тем самым породив несуществующие токены. Эту проблему можно решить следующим образом.

Обычно данная проблема возникает в первом предложении до знака тире. Мы можем получить все возможные пары этих слов и собрать из них токены. Но это может повлечь к проблемам связанным с размером токена.