Токенизатор. Отчет.

*Пискунова, Рахман, Захарова*

# Аналитическая часть

Токенизация -- один из первых и вместе с тем один из самых существенных этапов практически любого алгоритма обработки текста. В статье [Урюпина 2008] приведен пример того, как некачественное разбиение текста на предложения может повлиять на качество дальнейшего анализа:

*Но ведь Маша знает А. Б. Иванова много лет и никогда про него ничего плохого не слышала!!*

Наивный алгоритм разбиения на предложения “от точки до точки” поделил бы этот текст на четыре предложения, что сделало бы дальнейший корректный анализ автоматически невозможным. Причина этого заключается в том, что точка в русской пунктуации используется не только для обозначения конца предложения (в приведенном примере -- для сокращения инициалов).

В вышеупомянутой статье описан один из возможных действенных алгоритмов разбиения текста на предложения. Автор статьи исходит из допущения, что границы между предложениями **всегда** обозначаются терминальным знаком (точкой, вопросительным,восклицательным знаком). Таким образом, каждый такой знак является объектом, для которого с помощью бинарного классификатора строится предсказание -- разбивает ли он текст на предложения или нет. В качестве признаков для машинного обучения классификатор использует различную информацию о контексте знака препинания.

# Техническое задание

Формат данных на входе - массив с текстами.

На выходе - файл формата csv, в каждой строке - ID документа, ID токена, токен, тип токена.

# Организация программы

Класс Tokenizer с методами splitter - делит текст на предложения, tokenize - делит предложения на токены и приписывает им типы, speller - нормализует токены с ошибками и опечатками. Также функции для записи в файл данных о токенах в формате csv. Дополнительно присутствует класс Profiler, с помощью которого измеряется время работы программы.

Используемые инструменты:

* регулярные выражения
* модуль pyaspeller - для проверки и исправления правописания
* списки географических наименований (все формы для каждой лексемы), слов, пишущихся через дефис (все формы для каждой лексемы), эмодзи, текстовых смайлов

# Типы токенов

* word - словоформы, в том числе с дефисом, сокращения
* punct - знаки препинания
* num - числа
* entity - именованные сущности, которые хотелось бы считать одним тегом
* enword - некириллические словоформы
* link - ссылки и емейлы
* emoji - все эмодзи
* smile - смайлы разных видов (скобки, скобки+цифры ())00))0)), знаки препинания+скобки, текстовые смайлы (например ( ͡° ͜ʖ ͡°) ))

# Сложные случаи

## Дефисное написание

Проблема дефисного написания решается с помощью списка слов, пишущихся через дефис и списка географических названий. Таким образом “Санкт-Петербург” - один токен, “ Москва-Санкт-Петербург” - три токена (“Москва”, “-”, “Санкт-Петербург”), “кое-что” - один токен.

Отдельной проблемой стали такие лексемы в косвенных формах. Подавляющее большинство списков, например, географических названий, содержат только их начальные формы. Для того, чтобы получить все употребления такой лексемы с помощью регулярного выражения, зачастую необходима информация не о начальной форме слова (напр. *Москва*), а о его основе (*москв-*). Определение же основ по лемме само по себе является отдельной нетривиальной задачей (ср. *Сан-Марино* и *Пушкино --* для *Пушкино* возможен как вариант основы *пушкино-*, так и *пушкин-*).

Кроме того, такой подход к проблеме требовал бы обязательного использования морфологического анализатора непосредственно при разметке текста.

В связи с этим было принято решение расширить списки географических объектов/слов, пишущихся через дефис, добавив в них все косвенные формы уже включенных лексем, полученные с помощью морфологического анализатора pymorphy2.

## Другое

Проблема сокращений типа “т.д”, “т.п.”, “с.т.а.л.к.е.р.” решена с помощью регулярных выражений. Таким же образом выделялись смайлики (эмодзи и текстовые), а также ссылки.

# Тестирование

Accuracy на тестовой статье из корпуса (509 токенов):

* По выделению токенов ~ 0.97
* По определению типов токенов ~ 0.9

Подробная таблица с токенами, их типами и измерением качества прилагается (файл test\_tokens.csv).

# Часть 2: поиск сокращений по корпусу

Были выделены следующие типы аббревиатур:

* инициалы (**А.С.** Пушкин, **М. Ю.** Лермонтов) (шаблон: [^А-ЯЁ](([А-ЯЁ]\. ?){1,3})
* сокращения типа *и* ***т.д.***, ***т. к.***(шаблон: [\(\):;" ]([а-яёА-ЯЁ]\. ?[а-яё]\.)[\(\):;" ])
* сокращения типа ***шт.*** (шаблон: [\(\):;" ]([а-яё]{2,3}\.)[\(\):;" ])

стоит отметить, что для обработки сокращений этого типа был использован словарь морфологического анализатора pymorphy2. Это было необходимо, чтобы отделить сокращения от коротких слов на конце предложения (напр. *Ему было 20 лет.*)

* сокращения типа ***с.ш.а.*** (шаблон: [\(\):;" ]((\w\.){3,10})[\(\):;" ])

При выполнении данного задания использовались следующие библиотеки python: re, collections, pymorphy2

Программа получает на вход название файла с корпусом и записывает в файл abb\_list.txt все выделенные аббревиатуры и количество их вхождений в корпусе.

**Действия при выполнении задачи:**

1. эмпирический анализ данных
2. выделение типов сокращений
3. написание регулярных выражений для их поиска
4. запись результатов в некоторую структуру данных
5. запись структурированных данных в файл abb\_list.txt

# Роли в команде:

* Пискунова - подбор корпуса, дефисное написание базовых слов (не географических названий) действия 1-3 задания 2, разделение текста на предложения, спеллчекер
* Рахман - дефисное написание географических названий, действия 3-5 задания 2 (3 пункт делался совместно), разделение текста на предложения (совместно)
* Захарова - чистка корпуса, тестирование, обработка эмоджи и смайлов, ссылок, хештегов, классификация токенов, наивный алгоритм