Обрезковой Дарьи
Валерьевны
НИУ ВШЭ ФКН
группа БПМИ164

Отчет по домашнему заданию №1 по автоматической обработке текста

Графематический и морфологический анализ текста, статистика

2019 г.

Содержание

Введение

Задача. Реализовать на одном из языков программирования собственный графематический анализатор для русскоязычных текстов (токенизация и/или сегментация на предложения) и протестировать его на реальных текстах.

Для разработки графематического анализатора необходимо было реализовать программу, которая принимала бы текст и некоторые параметры вида разбиения и разбивала бы его на токены; и интерфейс, где пользователь мог бы завести текст для обработки и установить необходимые параметры.

Для работы я решила использовать язык программирование JavaScript, языки разметки и стилей HTML и CSS и фреймворк Vue.js. Для создания сервера использовались Node.js и бесплатный хостинг Heroku.

Разработка графического интерфейса

Если коротко описывать интерфейс - он должен был содержать простой и понятный дизайн, необходимы были поля настроек, поле ввода текста и поле вывода массива с полученными после выполнения программы токенами.

В качестве ввода параметров были использованы кнопки - типы токенизации и сегментации, а также чекбоксы дополнительных параметров для работы программы.

Кратко о возможностях программы

Данная программа представляет собой сегментатор и токенизатор, которые рассматривают наиболее популярные разделения и протестированы на различных сложных видах текстов.

В программе доступна сегментация - разделение на предложения.

Токенизация в виде разделения на слова; на строки; разделение по пробелам, знакам препинания и переносу строк; посимвольно.

После выбора разбиения появится поле для ввода текста. Как только вы введете текст, нажмите кнопку “Разделить”. Ниже появится новое поле с результатом работы программы, представленном в виде массива строк - токенов (токенизация) или предложений (сегментация).

Разработка сегментатора

Задача сегментатора - разбивка текста на предложения, с учетом наличия неделимых токенов, в которых содержатся точки, пробелы и т.д.

Для каждого языка можно независимо задать свой набор разделителей предложений. Для русского языка таковыми символами обычно являются точка, знаки вопроса и восклицания. Будем считать, что нам вводят верные относительно правил русского языка предложения. Рассмотрим сложные случаи при разбиении текста на предложения:

1. Предложение может заканчиваться не одним знаком препинания (Напр. «…», «?!», «????» и т.д.)
2. В предложение могут содержаться сокращения слов и словосочетаний («напр.», «и т.д.» и т.д. :))
3. В том числе. В предложениях могут содержаться сокращения имен («И.И.Иванов»)
4. В предложение могут содержаться абревиатуры типа «Р.С.Ф.С.Р.»
5. Предложение может быть односложным («Приехали!»)
6. Предложение содержит диалоги и прямую речь. («Приехали!» - сказал водитель.)
7. Предложение содержит цитату, не являющуюся прямой речью ("Справедливо сказал Гоголь, что «в Пушкине, как будто в лексиконе, заключилось все богатство, гибкость и сила нашего языка»» (Белинский).)
8. Содержит дробные числа с точкой и факториалы («3.14», «3!»)

Вариации прямой речи в русском языке:

Прямая речь перед словами автора	Прямая речь после слов автора	Слова. автора внутри прямой речи	Прямая речь внутри слов автора
«П!» - а. «П?» - а. «П», - а.	А: «П!» А: «П?» А: «П…» А: «П».	«П, — а. — П». «П, — а, — п». «П, — а и а: — П».	А: «П», — а. А: «П?» — а. А: «П!» — а. А: «П…» — а.

Обработка таких исключений - токенов с точкой внутри - опирается на возможность сегментатора распознавать в потоке символов специальные цепочки с разделителями внутри.

В ходе работы я постаралась обработать наибольшее количество сложных случаев.

1. Первый случай со множественными знаками препинания в конце предложения можно решить с помощью регулярного выражения. Так как предложения не могут начинаться со знака препинания, будем причислять все знаки препинания, стоящие строго друг за другом к последнему предложению.

2. Так как существует множество сокращений (как официальных, так и чисто пользовательских), то попробуем сформировать небольшой словарь, в который войдут самые популярные сокращения. Есть целые сборники сокращений и аббревиатур, которые можно в перспективе включить в программу, но есть проблема, которая не дает это сделать сейчас: необходим прямой доступ к данным (не через интерфейс сайта) или такие данные, которые мы могли бы использовать сразу (не бумажная книга, так как ее придется переводить в электронный формат). Воспользуемся выборкой с сайта (https://orfogrammka.ru/OGL03/70091445.html), чтобы мы могли покрыть некоторые случаи.
Будем просто смотреть, делить ли по данной точке на 2 предложения эту часть текста или нет.

3. 4. и 5. Пусть если до точки стоит заглавная буква, то точка является либо частью аббревиатуры, либо инициалом.
Однако, хуже с сокращениями имени типа «Дж. Мартин». В таких случаях, если мы сформулируем правило, что «предыдущий или пред-предыдущий символ - заглавная буква», то у нас возникнет конфликт с односложными предложениями, в которых одно слово длины 2. Например: «Еж!», «Ай!». Так как в русском языке инициал чаще всего в виде одной буквы, то воспользуемся первой версией правила, а в перспективе заведем словарь с сокращениями имен длинной в 2 и более букв.

6. и 7. Предложения с прямой речью будем считать одним предложение, так как хоть они и часто разделены знаками конца предложения (« ! ? .»), но являются одним целым. Нам важно, что если предложение содержит кавычки, после которых идет заглавная буква, то дальше до закрытия кавычек или до конца речи автора у нас будет 1 предложение.

У нас есть еще и цитаты, которые мы будем считать тоже как часть предложения.

Возможен еще вариант деления на предложения внутри цитат, однако тогда существуют ситуации, когда некоторые предложения будут содержать открывающие и закрывающие кавычки, а другие предложения из цитаты их содержать не будут. Из-за этого предложения с кавычками самостоятельно не могут быть использованы, так как содержат пунктуационные ошибки.

Опустим случаи, когда внутри одних кавычек у нас содержаться еще кавычки. Позде такое событие можно обработать, введя новые переменные, которые будут отвечать, какие кавычки сейчас открыты.

Также не будем рассматривать случаи с диалогом. Он схож с прямой речью, однако в нем могут содержаться больше одного предложения как в прямой речи, так и в словах автора. Оставим этот случай на усовершенствование.

8. Если в тексте содержатся цифры с точкой (например, “3,14”) , то будем игнорировать точку, если до и после нее стоят цифры. Если у нас факториал (например, “3!”), то будем оставлять знак восклицания, если до него стоит цифра. Здесь мы опять же видим конфликт - знак восклицания может означать как факториал, как и просто число в конце восклицательного предложения. Будем считать, что в последнем случае в нашем тексте будет не число, а числительное, написанное прописью. Иначе значения можно определить только понимая его значение, чего технически в данной программе пока сделать не можем (если не захотим еще прикрутить сюда машинное обучение, но это уже другая история).

Тесты

1.

Ввод:

А тут много вопросов?!! Не кричите! Что? Не кричите, говорю!!!! А, простите... Не хотел.

Вывод:

["А тут много вопросов?!!", "Не кричите!", "Что?", "Не кричите, говорю!!!!", "А, простите...", "Не хотел."]

2.

Ввод:

А там были и др., и пр., и т. п., и т. д., т. е., и т.п., и т.д., т.е., во как! ПОНЯТНО?

Вывод:

["А там были и др., и пр., и т. п., и т. д., т. е., и т.п., и т.д., т.е., во как!", "ПОНЯТНО?"]

3. 4. 5.

Ввод:

А.С. Пушкин великий поэт! Ёж. При чем тут ёж? И. Иванов жил во времена Р.С.Ф.С.Р.

Вывод:

["А.С. Пушкин великий поэт!", "Ёж.", "При чем тут ёж?", "И. Иванов жил во времена Р.С.Ф.С.Р."]

6. 7.

Проверим, реагирует ли программа на все виды кавычек. Чтобы не было конфликтов и экранирования (\”), мы пожертвуем двойными кавычками - при выводе они будут одинарными. В целом, их можно сохранить, но при выводе они, как уже я упомянула, будут экранироваться, что выглядит не очень красиво и понятно.

Ввод:
Маленькая девочка бежала и повторяла: "Не видали маму?"
Маленькая девочка бежала и повторяла: ‘Не видали маму!’
Маленькая девочка бежала и повторяла: 'Не видали маму...'
Маленькая девочка бежала и повторяла: «Не видали маму.»

Вывод:
["Маленькая девочка бежала и повторяла: 'Не видали маму?'",
"Маленькая девочка бежала и повторяла: ‘Не видали маму!’",
"Маленькая девочка бежала и повторяла: 'Не видали маму...'",
"Маленькая девочка бежала и повторяла: «Не видали маму.»"]

Ввод:
«А что Казбич?» – спросил я нетерпеливо у штабс-капитана.
«А что Казбич», – спросил я нетерпеливо у штабс-капитана.
Вывод:
["«А что Казбич?» – спросил я нетерпеливо у штабс-капитана.",
"«А что Казбич», – спросил я нетерпеливо у штабс-капитана."]

Ввод:
"Выслушайте меня, – сказала Надя, – когда-нибудь до конца".
"Выслушайте меня, – сказала Надя. – Когда-нибудь до конца".
"Выслушайте меня, – сказала Надя и продолжила: – Когда-нибудь до конца".
Вывод:
["'Выслушайте меня, – сказала Надя, – когда-нибудь до конца'.",
"'Выслушайте меня, – сказала Надя. – Когда-нибудь до конца'.",
"'Выслушайте меня, – сказала Надя и продолжила: – Когда-нибудь до конца'."]

Ввод:
Маленькая девочка бежала и повторяла: "Не видали маму?", - с трудом выговаривая она.
Вывод:
["Маленькая девочка бежала и повторяла: 'Не видали маму?', - с трудом выговаривая она."]

Ввод:
Введите 3.14, должно сработать!
Нужно посчитать 1000!, но точно.
Сколько будет 10!?
Вывод:
["Введите 3.14, должно сработать!",
"Нужно посчитать 1000!, но точно.",
"Сколько будет 10!?"]

Разработка токенизатора

Токенизатор — инструмент для автоматического или полуавтоматического разделения текста на токены, т.е. на слова и другие цепочки символов, которые мы хотим считать минимальными линейными единицами текста.

Получается, что как и для сегментатора, мы должны задать некоторые разделители для текста. Рассмотрим несколько видов токенизации, которые я реализовала в программе, и их особенности:

1. Разделение посимвольно.

   Пожалуй, самый простой способ разделения, так как наша задача разделить текст относительно разделителя - “пустой символ”.

   Разделение посимвольно доступно в разделе дополнительная токенизации.
2. Разделение по знакам препинания, переносу строк и пробелам.
   Так как мы все равно разделяем по знакам препинания, уберем все пробелы после знаков препинания. Теперь воспользуемся регулярным выражением со следующими знаками препинания: [! ; : , . ? - –], а также пробельными символами и переносом строки.
   Разделение по знакам препинания, переносу строк и пробелам доступно в разделе дополнительная токенизации.
3. Разделение только по пробелам. Опять же используем простое разделение.
   Разделение только по пробелам доступно в разделе дополнительная токенизации.
4. Разделение на строки. Разделение производится, отслеживанием нажатия клавиши Enter или ввода “\n”.
5. На слова. Деление на слова - самая сложная из рассматриваемых токенизаций, так как в ней необходимо учитывать много сложных случаев. Самое простое разбиение на слова - это уже рассмотренный случай разбиение по пробелам, знакам препинания и переносу строк. Но нужно учитывать, что так же у нас есть сокращения, слова, пишущиеся через дефис, дробные числа и др.

Тесты на строки:

Ввод:

Введите 3.14, должно сработать!
Нужно посчитать 1000!, но точно.
Сколько будет 10!?

Вывод:

["Введите 3.14, должно сработать!", "Нужно посчитать 1000!, но точно.", "Сколько будет 10!?"
]

Тесты на пробелы:

Ввод:
Введите 3.14, должно сработать!
Нужно посчитать 1000!, но точно.
Сколько будет 10!?
Вывод:
["Введите", "3.14,", "должно", "сработать!Нужно", "посчитать", "1000!,", "но", "точно.Сколько", "будет", "10!?"]

Тесты на знаки препинания:

Ввод:
Введите 3.14, должно сработать!
Нужно посчитать 1000!, но точно.
Сколько будет 10!?
Вывод:
["Введите", "3", "14", "должно", "сработать", "Нужно", "посчитать", "1000", "но", "точно", "Сколько", "будет", "10"]

Тесты на символы:

Ввод:
Введите 3.14, должно сработать!
Нужно посчитать 1000!, но точно.
Сколько будет 10!?
Вывод:
["В", "в", "е", "д", "и", "т", "е", " ", "3", ".", "1", "4", ",", "д", "о", "л", "ж", "н", "о", " ", "с", "р", "а", "б", "о", "т", "а", "т", "ь", "!", "Н", "у", "ж", "н", "о", " ", "п", "о", "с", "ч", "и", "т", "а", "т", "ь", " ", "1", "0", "0", "0", "!", ",", "н", "о", " ", "т", "о", "ч", "н", "о", ".", "С", "к", "о", "л", "ь", "к", "о", " ", "б", "у", "д", "е", "т", " ", "1", "0", "!", "?"]

Тесты на слова:

Ввод:
Введите 3.14, должно сработать!
Нужно посчитать 1000!, но точно.
Сколько будет 10!?
Вывод:
["Введите", "3.14", "должно", "сработать", "Нужно", "посчитать", "1000", "но", "точно", "Сколько", "будет", "10"]

Вывод

В ходе работы были проанализированы виды различные структуры текстов, на основе которых сформированы сложные случаи, которые позже вошли в тесты программы. Можно сделать вывод, что токенизаторы и сегментаторы очень сильно привязаны к каждому языку, так как они используют синтаксис и структуру языка. Хороший токенизатор должен содержать в себе словарь, в котором будут содержаться сложно обрабатываемые случаи (например, сокращения слов в русском языке). Не смотря на то, что данная работа не может обработать идеально все случаи, она обрабатывает самые популярные из них, давая представлении о структуре подобных программ и тех сложностей, с которыми можно столкнуться.

Список используемых источников

http://www.solarix.ru/for_developers/docs/tokenizer.shtml
https://www.hse.ru/data/2013/06/03/1285521221/report_v2.pdf
http://new.gramota.ru/biblio/readingroom/rules/164-znapr
https://orfogrammka.ru/OGL03/70091445.html