ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ
им. А.Н. ТИХОНОВА

Вдовкин Василий Алексеевич, группа БИВ-144

МЕТОДЫ СОКРАЩЕНИЯ ТЕКСТА И ИЗВЛЕЧЕНИЯ КЛЮЧЕВОЙ ИНФОРМАЦИИ ИЗ РУССКОЯЗЫЧНЫХ НОВОСТНЫХ СТАТЕЙ

Выпускная квалификационная работа
по направлению 09.03.01 Информатика и вычислительная техника
студентов образовательной программы бакалавриата
«Информатика и вычислительная техника»

Студент _____ В.А. Вдовкин

Рецензент

_____

Руководитель
старший преподаватель
Ф.В. Строк

_____

Москва 2018 г.

Аннотация

Данная работа описывает процесс реализации сервиса, способного значительно улучшить и упростить пользовательское взаимодействие с новостным контентом, используя обработку естественного языка (Natural Language Processing, NLP). Сервис анализирует поток русскоязычных новостных статей в реальном времени, группирует их по конкретным событиям и выделяет ключевую информацию о событии. Для построения такого сервиса мы изучаем и используем различные методы и модели, распространенные в NLP для решения следующих задач: нормализация, векторизация, кластеризация и суммаризация текста. Кластеризация используется для выделения из потока данных множества статей, относящемся к одному событию. Мы собираем и обрабатываем большое количество данных с web-сайтов медиа и обучаем векторизатор TF-IDF, позволяющий использовать K-means для кластеризации. Суммаризация извлекает самые информативные предложения из кластера-события и формирует параграф из 5 следующих по смыслу предложений. Для суммаризации используются две модели: SimBasic и DivRank. Выбранные решения сравниваются и оцениваются.

Abstract

This work describes implementation of the service that can significantly improve user experience in news content consumption by using Natural Language Processing (NLP for short). This service analyses stream of news articles from russian media web-sites in real-time, groups news by events and extracts the most valuable information about the events. To implement this service we first research and then use models and methods from NLP to find suitable solution for the following problems: normalization, vectorization, clusterization and summarization of text. Clusterization allows us to automatically group news by events. In order for this to work, we collect the data from web-sites of media and train TF-IDF model allows to use K-means algorithm for news clusterization. Summarization depends on SimBasic and DivRank models and extracts the most informative sentences from the event-cluster. We compare and evaluate the proposed solutions.

Содержание

1 Введение

Когда в мире происходит какое-либо событие, различные средства массовой информации пишут статьи с информацией об этом событии в виде новостей. Пользователю часто бывает сложно ориентироваться в большом потоке данных от разных источников. Автоматическая систематизация и обработка таких данных с целью предоставить наиболее полную и информативную картину может сэкономить человеку много времени.

Модели и алгоритмы обработки естественного языка (Natural Language Processing, NLP) решают задачи автоматического анализа текста и позволяет: при использовании нескольких источников объединять новостные статьи в группы (кластеры) по релевантности к конкретному событию (кластеризация), извлекать из кластера наиболее информативные данные о событии, например, в виде нескольких предложений.

Основной целью выпускного проекта является разработка веб-сервиса, который автоматически группирует российские новостные статьи по событиям и извлекает из них ключевую информацию в режиме реального времени.

Для достижения данной цели необходимо решить следующие задачи:

1. Реализация системы сбора данных: парсинг сайтов СМИ для получения новостей вместе с их метаданными.
2. Изучение алгоритмов и моделей анализа текста: нормализация, векторизация и кластеризация, суммаризация.
3. Техническая реализация модуля анализа текста.
4. Проектирование и разработка инфраструктуры сервиса, интеграция с ранее реализованными модулями сбора и анализа данных.
5. Оценка качества сервиса, выводы о проделанной работе.

Достижение цели проекта и успешное выполнение задач определяется получением следующих результатов:

1. Комбинация моделей NLP, обученных на русском языке, корпусе и алгоритмах, подходящих для кластеризации и сокращения новостных статей.
2. Веб-сервис, который демонстрирует работу предлагаемых решений для эмпирической оценки качества методов обобщения и кластеризации.

2 Данные

Многие модели, используемые в решениях NLP задач, требуют большого количества предварительно обработанных данных для обучения, в данном случае такими данными является корпус русскоязычных новостей. В исследовательских работах авторы часто используют готовые данные —общедоступные размеченные датасеты, но если учитывать цель проекта, то без реализации своей системы, позволяющей получать новости с нескольких источников за определённый период времени, не обойтись.

3 Анализ текста

3.1 Нормализация

В данной работе рассматриваются и используются решения следующих NLP задач: нормализация, векторизация, кластеризация, суммаризация.

Текст на естественном языке содержит много избыточных элементов, без которых его смысл не изменится, но которые влияет на точность моделей, так как могут быть непостоянными и действовать как шум. Поэтому при решении любой задачи в NLP текст нормализуют, то есть приводят в общую, более удобную форму. Нормализация включает в себя несколько шагов.

Чтобы с информацией можно было работать, её дискретизуют. Похожий процесс в NLP называется токенизация, заключающийся в делении текста на части — токены, чаще всего токен является одним словом. К сожалению, нельзя просто делить текст по пробелам, так как существуют множество исключений, например, Великие Луки — это один токен, хотя и состоит из двух слов. Если рассматривать это как два токена, то смысл текста будет искажён, что может сказаться на результате и на качестве решения задачи. Для токенизации удобно использовать регулярные выражения.

Следующий шаг нормализации — удаление стоп-слов. Стоп-слова примерно одинаково распределены по всему корпусу языка. В русском языке многие служебные части (союзы, междометия, предлоги и т.д.) являются стоп-словами.

Для однозначной идентификации слова его приводят к начальной форме. Данный процесс называется лемматизация, а начальная форма — лемма. Для лемматизации недостаточно использовать только словарь, потому что существу-

```
from nltk.corpus import stopwords
from pymystem3 import Mystem; mystem = Mystem()
import string
from stop_words import get_stop_words

STOP_WORDS = (set(stopwords.words('russian')) | set(stopwords.words('english'))
set(get_stop_words('ru')) | set(get_stop_words('en')))

def get_word_normal_form(word):
return ''.join(mystem.lemmatize(word)).strip().replace('ё', 'е').strip('-')

def lemmatize_words(text):
text = text.lower()
text = ''.join([i for i in text if ( i not in string.punctuation )])
res = []
for word in text.split():
norm_form = get_word_normal_form(word)
if len(norm_form) > 2 and norm_form not in STOP_WORDS:
res.append(norm_form)
return ' '.join(res)
```

Рис. 1. Простая функция нормализации.

ет огромное количество неологизмов, подчиняющимся тем же морфологическим правилам при образовании форм. Хорошие лемматайзеры проводят полный морфологический парсинг, при котором слова делятся на морфемы: стемы (самые осмысленные части) и афиксы (придают дополнительное значение слову) [?]. Более простая версия морфологического анализа — стемминг, использующий определённые правила для извлечения основы слова.

Простая функция нормализации показана на рис. 1. В функции последовательно применяются следующие операции:

1. приведение текста в нижний регистр;
2. удаление символов пунктуации;
3. удаление стоп-слов.
4. лемматизация каждого слова с помощью Python библиотеки MyStem.

3.2 Векторизация

TF-IDF [?] — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. TF-IDF — это произведение двух статистик: TF (term frequency) и IDF (inverse document frequency). На сегодняшний день, TF-IDF один из самых популярных

Оригинал	Нормализация
Однако когда их проверили на восприятие концепций и идей, оказалось, что те, кто писал от руки, понимают пройденный материал лучше однокашников.	однако когда они проверять на восприятие концепция иде оказываться что тот кто писа от рука понимать проходить материал хорошо однокашни
Лингвистическую относительность упоминали в своих сочинениях немецкие философы еще в конце XVIII – начале XIX века, но известность гипотеза получила именно благодаря Уорфу.	лингвистический относительность уп свой сочинение немецкий фило еще конец xviii – начало xix в но известность гипотеза получ именно благодаря уорфу

Рис. 2. Примеры нормализованного текста

способов взвешивания слов, входящих в корпус документов. Например, 83% рекомендательных систем цифровых библиотек используют TF-IDF [**?**].

Существует множество способов подсчёта TF-IDF, в данной работе использовался следующий:

$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k},$$

где n_t есть число вхождений слова t в документ, а $\sum_k n_k$ — общее число слов в данном документе.

$$\mathrm{idf}(t,D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|},$$

где $|D|$ — число документов в корпусе, $|\{d_i \in D \mid t \in d_i\}|$ — число документов из коллекции D, в которых встречается t (когда $n_t \neq 0$).

Таким образом, мера TF-IDF является произведением двух сомножителей:

$$\text{TF-IDF}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D).$$

Признаковым описанием одного объекта $d \in D$ будет вектор

$$\big(\text{TF-IDF}(t,d,D)\big)_{t \in V},$$

где V — словарь всех слов, встречающихся в коллекции D.

3.3 Кластеризация

Представленные в виде векторов статьи можно кластеризовать алгоритмом k-means. Если подобрать оптимальные параметры количества кластеров и максимального расстояния от центра кластера, после которого статья удаляется из кластера, то статьи в одном кластере будут об одном событии [?].

3.4 Суммаризация

3.4.1 SumBasic

3.4.2 DivRank

4 Реализация сервиса

4.1 Архитектура и транспорт данных

4.2 Особенности реализации компонентов анализа

4.2.1 Система сбора данных

ПЕРЕПИСАТЬ, ВСЁ УЖЕ НЕ ТАК Система сбора данных представляет собой коллекцию парсеров сайтов российских СМИ, основанные на одном подходе и имеющие одинаковый интерфейс. Пример использования представлен на рис. 3.

В общей архитектуре сервиса система является обособленным фоновым процессом, собирающим раз в некоторое время новые статьи, поэтому нет необходимости постоянно поддерживать строгие ограничения к скорости парсинга. Но эту же систему можно использовать для относительно быстрого сбора собственного датасета. Кроме того, при инициализации сервиса понадобится получить новости за последние несколько часов, чтобы сформировать актуальные кластера. Парсинг множества статей можно ускорить в несколько раз, обрабатывая их параллельно.

Чтобы понять, как реализовать универсальную систему, с возможностью быстрого добавления поддержки нового ресурса, достаточно взглянуть на сайты русскоязычных медиа-ресурсов. Многие сильно отличаются друг от друга внешне, но у всех присутствует следующая логика: существуют страницы со списком новостей в хронологическом порядке, которые либо агрегированы по дням (Lenta.ru, Gazeta.ru, vedomosti.ru), либо используют параметр offset, указывающий с какой

```
from parsers import Gazeta, Tass, Lenta, Vedomosti, Novaya
import datetime

parsers = [
Gazeta(procs=4), # Number of processes used
Tass(),
Lenta(),
Vedomosti(),
Novaya(procs=4)
]
until_time = datetime.datetime.now() - datetime.timedelta(hours=4)
for parser in parsers:
print(parser.id)
for n in parser.get_news(until_time=until_time):
print(n['title'])
```

Рис. 3. Пример использования для получения всех новостей за последние 4 часа.

статьи начинать страницу (novayagazeta.ru, tass.ru, meduza.io). Первый вариант пагинации удобнее, потому что позволяет просто получить новости за любой заданный интервал времени. Во втором случае можно использовать бинарный поиск по страницам, но это не реализовано, так как новости всегда нужны с текущего момента. Большинство СМИ отдают данные в HTML формате и только лишь малая часть использует API в JSON формате.

Так как система обособлена, то писать её можно на любом языке, а взаимодействовать с сервисом через внешнее хранилище, но для удобства выбран `Python 3`, с использованием дополнительных библиотек: `BeautifulSoup` для парсинга HTML и `requests` для выполнения HTTP запросов. Так как Питон имеет ограничение на потоки из-за GIL, то чтобы обеспечить параллелизм, позволяющий с увеличением количества процессорных ядер ускорять обработку множества статей, используется модуль `multiprocessing`.

Главной частью системы является класс `BaseParser`, инкапсулирующий сетевые запросы, работу по синхронизации процессов и передаче данных между ними. В целом, логика межпроцессного взаимодействия системы достаточно тривиальная и находится в методе `get_news` (рис. 4): процесс-предок запускает процессы-потомки и начинает заполнять очередь задач ссылками на статьи, в этот момент дочерние процессы уже разбирают ссылки из очереди и обрабатывают их. В качестве очереди задач выступает класс `multiprocessing.Queue`, который комбинирует межпроцессное взаимодействие через pipe и разделяемые блокиров-

```
def get_news(self, start_time=None, until_time=None,
news_count=None, topic_filter=None):
. . .

Q_urls = Queue(0) # News urls and for deeper parsing
Q_out = Queue(0) # Results of parsing
sync_flag = Value('i', 1) # Flag to stop processes

workers = []
# Getting news by url in proceses
for _ in range(procs):
workers.append(Process(target=self._process_news,
args=(Q_urls, Q_out, sync_flag, topic_filter)))
workers[-1].start()
# Parsing pages with urls ("Лента новостей") and putting them to Q_urls
self.parse_pages(Q_urls, sync_flag, start_time,
until_time, news_count, topic_filter)
# Clearing output queue while processes still working
# Probably significantly slowing down other workers, need to fix it
out = []
self._listen_queue(workers, Q_out, out)
. . .
```

Рис. 4. Часть кода класса `BaseParser`.

ки. Благодаря модулю pickle очередь может передавать сложные объекты и часто применяется в подобных случаях.

После того, как основной процесс закончил парсинг страниц и отправил все задачи в очередь, он меняет значение переменной `sync_flag`, которая находится в общем для процессов сегменте памяти (Shared memory), если значение меняется, процессы-потомки понимают, что после опустошения очереди можно больше её не «слушать».

Результаты выполненных задач не принято передавать родительскому процессу, но в данном случае это было сделано для удобства интерфейса с помощью второй очереди. После смены значения общего флага предок начинает «слушать» очередь и ждать результатов. Такой подход крайне нежелателен, так как результат в разы больше параметров самой задачи (из-за текста статьи), и при большом количестве обработанных новостей все процессы-потомки останутся висеть в простое, пока предок будет извлекать из очереди результаты. На небольших объёмах это незаметно, но запускать такое на 64 ядерном процессоре в 64 процесса с целью выкачать новости за несколько лет не стоит.

Описанная проблема будет решена заменой Pipe-очереди на базу данных в оперативной памяти (in-memory database), например, на Redis. Можно решить её и с использованием Shared memory, но это гораздо сложнее, так как объекты Питона придётся сериализовывать для хранения и десериализовывать для использования вручную.

Чтобы создать новый парсер необходимо наследоваться от `BaseParser`, вызвать родительский конструктор с параметрами: название парсера, URL-префикс любой новости ресурса, URL-префикс ленты новостей, дефолтное количество процессов. Определить следующие методы:

- `_get_news_list(self, content)` — возвращает список списков с необработанными параметрами новости, используя контент (HTML или JSON) ленты новостей.
- `_get_news_params_in_page(self, news)` — возвращает `tuple` с параметрами статьи, используя элемент списка из предыдущего метода. URL и дата (в `datetime` объекте) должны быть первыми в списке параметров (`dict` не используется в данном случае в попытке выиграть время и место на pickle-запаковке объекта).
- `_parse_news(self, news_params)` — возвращает `dict` с новостью, например, `{'title': title, 'url': url, 'text': text, 'topic': topic, 'date': date}`.
- `_page_url(self)` — возвращает URL текущей страницы.
- `_next_page_url(self)` — «переворачивает» страницу и возвращает `_page_url`.