Санкт-Петербургский государственный университет

Программная инженерия

Булгаков Кирилл Александрович

Подсветка важного в текстах электронной почты

Бакалаврская работа

Научный руководитель:
к.т.н., доцент Брыксин Т. А.

Рецензент:
ведущий разработчик, ООО "Яндекс Технологии" Селиванов А. С.

Санкт-Петербург
2019

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering

Kirill Bulgakov

Highlighting important text in emails

Graduation Thesis

Scientific supervisor:
Candidate of Engineering Sciences Timofey Bryksin

Reviewer:
lead developer, Yandex Technologies LLC Alexander Selivanov

Saint-Petersburg
2019

Оглавление

Введение

Началом эпохи электронный почты принято считать 1971 год, когда для операционной системы UNIX появилась первая почтовая утилита. В настоящее время сложно представить себе пользователя, читающего письмо в консольном приложении: интерфейс почтовых клиентов, объем электронной переписки сильно изменились за последние десятилетия. Значимость электронной почты растет год от года. Так, согласно отчету аналитической компании Litmus, опубликованному в 2017 году[1], количество времени, которое пользователь тратит на чтение одного электронного письма, увеличилось на 7% за последние 6 лет и составило 11.1 секунд. Также за последний год увеличился и общий трафик писем и составил 293.6 миллиардов писем в день[2]. Эти факты свидетельствуют о существенных временных затратах на использование электронной почты. В связи с этим естественным образом возникает задача упрощения взаимодействия пользователей с электронными письмами.

Современные почтовые клиенты предлагают множество интерфейсных решений для удобной работы с электронной почтой. Среди таких решений можно выделить группировку писем по папкам, различные визуальные метки для писем, объединение писем с одним отправителем и темой в списки и многие другие.

Однако этих средств недостаточно, и некоторые почтовые клиенты умеют подсвечивать в письмах различные именованные сущности, такие как даты, адреса, номера телефонов и имена, используя алгоритмы обработки текстов на естественном языке (Natural Language Processing, NLP). Но пока отсутствует функциональность по созданию выжимки из текста, в которую входит наиболее существенная информация из письма. Настоящая работа и пытается решить данную задачу.

[1] https://litmus.com/blog/email-attention-spans-increasing-infographic

[2] https://www.radicati.com/wp/wp-content/uploads/2019/04/Email-Market-2019-2023-Executive-Summary.pdf

1. Постановка задачи

Цель работы заключалась в сравнении различных алгоритмов подсветки важного в текстах электронных писем при помощи расширения для браузера Google Chrome. В связи с этим были сформулированы следующие задачи.

1. Выбрать алгоритмы обработки текстов на естественном языке для реализации и определить метрики качества подсветки важного в текстах писем.

2. Спроектировать архитектуру и реализовать решение на платформе браузерных расширений Google Chrome, включающее реализацию выбранных алгоритмов и среду для экспериментов.

3. Провести сравнительный анализ алгоритмов, подсвечивающих важные слова в текстах электронных писем.

2. Обзор

Задачу определения важного в текстах можно свести к задачам экстрактивной суммаризации и статистической оценки важности слов. Под *экстрактивной суммаризацией* принято понимать процесс выбора подмножества слов или предложений из текста, которые сохраняют его смысл. Таким образом, исходный текст становится короче и при хорошей работе алгоритма не теряет основный смысл. Полученную в результате работы алгоритма выжимку важного из исходного текста и предлагается подсвечивать в письме. Статическая оценка важности слов — мера важности слова, которая зависит от частоты его употребления в конкретном документе, а также от редкости использования в корпусе документов.

В этом разделе описаны алгоритмы, позволяющие решать данные задачи, объясняется сложность использования стандартных метрик качества, а также проводится обзор технологий и сервисов, используемых для реализации решения на платформе браузерных расширений.

2.1. Алгоритмы суммаризации и выделения важных слов в текстах на естественном языке

Программные продукты, основанные на алгоритмах обработки текстов на естественном языке, в последние годы получили широкое использование. Определение настроения текста [10], выдача по поисковых запросу соответствующих Web-страниц [1], машинный перевод [12] — все это примеры того, как NLP помогает решать задачи информационного поиска и делать программные интерфейсы более естественными для человека. Для решения поставленной задачи необходимо было выбрать алгоритмы, которые могут быть использованы для экстрактивной суммаризации и определения наиболее важных слов в тексте.

В результате обзора предметной области в данной работе были выбраны следующие алгоритмы:

- TF-IDF,

- векторные разложения,
- TextRank,
- латентно-семантический анализ (LSA).

Обзор и спецификации данных алгоритмов можно получить в следующих источниках [2, 4, 6, 11].

2.2. Метрики качества подсветки

Задачи обработки текстов на естественном языке принято делить на следующие категории: классификация текста, определение семантического смысла текста, генерация текста по тексту, разработка диалоговых систем. Подсветка важного в тексте лежит на пересечении задач определения семантического смысла текста и генерации текста по тексту. Имеющиеся исследования [3, 5, 8, 12, 13] показывают, что в данных категориях задач для оценки качества используются классические метрики качества: доля правильных ответов (precision), ROUGE, F-мера, BLEU, AUC. Все эти метрики объединяет тот факт, что для их использования необходимо иметь размеченный набор данных (dataset), который и используется для сравнения предсказаний получившейся модели с актуальными результатами. В связи с отсутствием размеченного набора данных на русском языке для оценки качества подсветки невозможно использовать подобные метрики качества. В [7] показано, что при отсутствии альтернативных вариантов, для оценки качества в задачах NLP может быть использована *оценивающая группа* — группа людей, оценки работы алгоритма которых считаются верными. В связи с этим данная метрика и была выбрана в настоящей работе для оценивания качества алгоритмов.

2.3. Браузерные расширения

Данный раздел обеспечивает краткий обзор возможностей расширений для браузера, а также общих технических деталей, необходимых

для реализации расширений. Подробное описание реализации обсуждается в главе 3.

Браузерные расширения — это небольшие клиентские программы, которые дополняют или изменяют работу браузера. Расширения разрабатываются при помощи стандартного веб-стека: HTML, JavaScript и CSS, а затем выполняются в фоновом режиме, взаимодействуя с содержимым веб-страницы и действиями пользователя. Решение использовать браузер Google Chrome в качестве целевого браузера было принято в связи с тем, что на данный момент он является наиболее используемым[3], что позволит получить наиболее широкий охват аудитории для разрабатываемого решения. Более подробное описание возможностей и архитектуры расширений обеспечивает обзор[4].

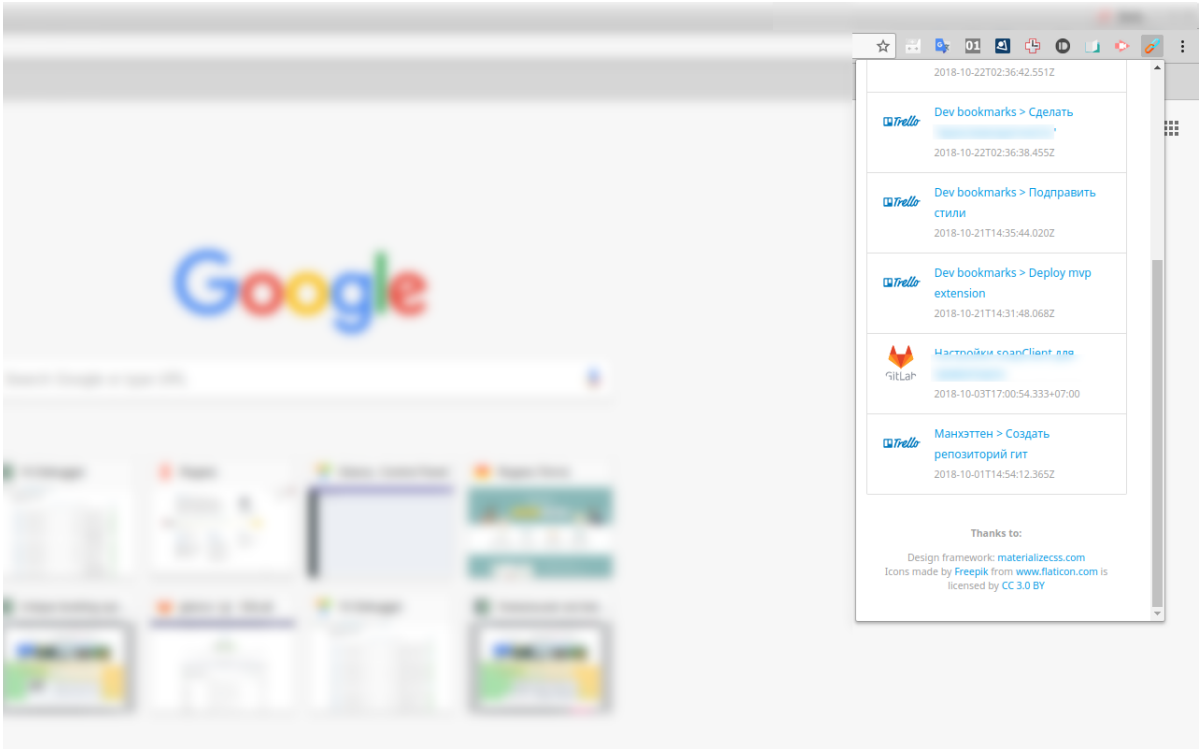

Рис. 1: Пример расширения для управления задачами

На рисунке 1 приведен пример расширения, выполняющего функцию менеджера задач. Как видно на рисунке, расширения имеют свой собственный интерфейс (или могут работать исключительно в фоновом

[3]https://www.w3counter.com/globalstats.php
[4]https://developers.chrome.com/extensions/overview

режиме) и интегрируют дополнительную функциональность в браузер.

Готовое приложение может быть выложено в Chrome Web Store[5] и установлено любым пользователем.

В связи с описанной выше необходимостью иметь клиент-серверное взаимодействие клиентский JavaScript-код должен иметь возможность выполнять кросс-доменные запросы. Браузеры по умолчанию не позволяют выполнять запросы, в которых хост запроса отличается от хоста, на котором выполняется скрипт. Это неоходимое требование с точки зрения безопасности. Справиться с ограничением возможно при помощи механизма `CORS`[6], который определяет правила для обеспечения безопасного выполнения кросс-доменных запросов. Стандартные браузеры действуют в соответствии с данным механизмом и ожидают определенные заголовки в ответ на запрос с использованием HTTP-метода `Options`, который отсылается перед основным запросом. Если в ответе получены заголовки, разрешающие кросс-доменный обмен информацией, то браузер далее выполняет и основной запрос.

2.4. Платформа Яндекс.Толока

Для привлечения оценивающей группы и проведения сравнительного анализа применения различных алгоритмов к задаче выделения главного в тексте в данной работе использовалась платформа Яндекс.Толока. Данная платформа является сервисом компании Яндекс, который используется самой компанией для совершенствования поисковых алгоритмов и машинного интеллекта. Ежедневно десятки тысяч людей выполняют задания в Яндекс.Толоке: оценивают релевантность сайтов, классифицируют изображения, отмечают объекты на фотографиях[7].

На рисунке 2 представлен интерфейс задания, который чаще всего представляет собой несколько медиафайлов и форму отправки ответов. Медиафайлы и их внешний вид легко меняются в настройках задания,

[5]https://chrome.google.com/webstore/category/extensions

[6]https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[7]https://habr.com/ru/company/yandex/blog/358462/

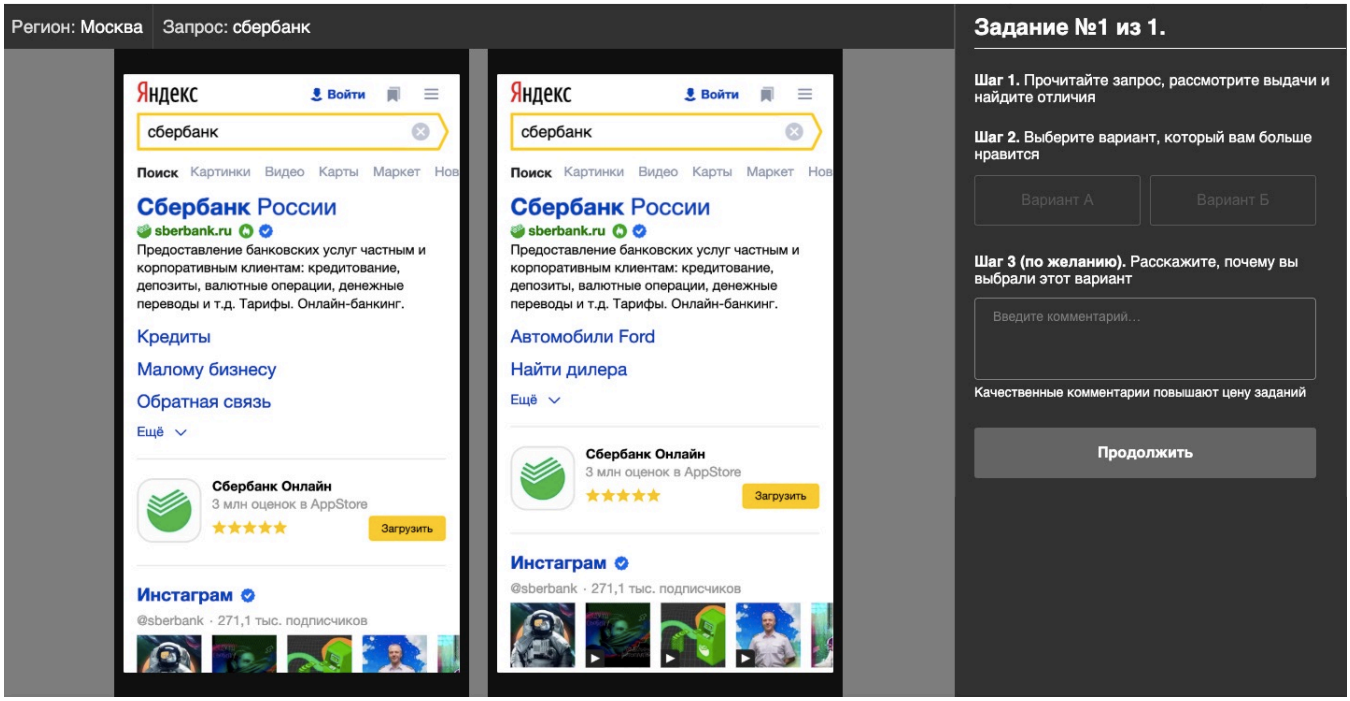

Рис. 2: Пример задания со сравнением двух вариантов поисковой выдачи

что делает данный сервис подходящим почти к любому типу задач сравнения и классификации.

С 2016 года сервис доступен всем желающим в рамках открытого бета-тестирования. Основная задача сервиса заключается в том, чтобы соединять между собой заказчиков и исполнителей. Например, если заказчику необходимо разметить некоторый набор данных, он может создать задание с подробной инструкцией, а затем отправить эти задачи пользователям на выполнение. Для заказчика также существует возможность фильтрации списка исполнителей по различным параметрам, таким как пол, возраст, геопозиция и другие. Эти и другие возможности позволяют исследователям получать наиболее релевантные ответы.

Яндекс.Толока поддерживает REST API[8], который позволяет автоматически создавать и редактировать задания, а также получать и обрабатывать результаты. Поставленная задача предполагает работу с большим количеством данных, которое исключает возможность ручной

[8]https://tech.yandex.ru/toloka/doc/concepts/about-docpage/

обработки результатов, поэтому наличие программного интерфейса является очень полезным.

2.5. Платформа AWS Lambda

Клиент-серверная архитектура требует развертывания приложений на сервере. Данный раздел обеспечивает обзор платформы AWS Lambda, которая была использована в настоящей работе для этих целей. Подробное описание процесса развертывания приводится в главе 3.4.

Популярность платформ бессерверных вычислений стремительно выросла в последние годы [9]. Данный тип вычислений, несмотря на название, не предполагает полное отсутствие серверов в архитектуре приложений. Разработчику предлагается возможность загружать свой программный код на платформу, которая будет запускать его в собственном облаке как функцию, предоставляя к ней доступ через API для работы с облачной инфраструктурой. Таким образом, разработчик вместо развертывания и обслуживания собственного сервера использует предоставленный API. Такая архитектура подходит для приложений, в которых клиентам не требуется постоянное соединение с сервером, а достаточно нечастых запросов для обмена информацией. Используемая в настоящей работе архитектура является примером такого клиент-серверного взаимодействия.

AWS Lambda запускает программный код в собственной вычислительной среде, самостоятельно занимаясь администрированием, масштабированием и мониторингом кода. От пользователя необходимо только загрузить исходный код и все нужные зависимости, а также установить требования к ресурсам.

AWS Lambda использует модель без сохранения состояния, что делает Lambda функции похожими по смыслу на чистые функции. На самом деле Lambda использует кеширование для оптимизации скорости выполнения, но этот факт не позволяет выполнять установку зависимостей исходного кода при каждом новом вызове. Поэтому загружаться в AWS Lambda должен архив, в котором содержатся все необходимые

для выполнения программные файлы.

После загрузки программного кода на сервис создается функция Lambda, которая пребывает в состоянии постоянной готовности к запуску. Такой запуск может произойти в ответ на определенные события, в том числе HTTP запросы.

Lambda поддерживает интеграцию с сервисом Amazon API Gateway[9]. Данный сервис предназначен для создания, публикации, обслуживания, мониторинга и обеспечения безопасности API в любых масштабах. Вызовы к созданному API могут обрабатываться различными серверными сервисами Amazon, в том числе AWS Lambda. Таким образом решается задача по интеграции реализаций алгоритмов в расширение: обмен информацией происходит при помощи HTTPS запросов на специальный адрес, создаваемый API Gateway.

2.6. Система виртуализации Docker

Система виртуализации Docker использована в настоящей работе для эмуляции окружения, в котором реализованные алгоритмы будут выполняться на бессерверной платформе. Данный раздел аргументирует необходимость этой системы в разработанном решении, а также приводит обзор основных принципов ее работы.

Каждая Lambda функция выполняется в облачном окружении, которое чаще всего отличается от локального окружения разработчика. В связи с этим становится актуальным вопрос обеспечения идентичной среды исполнения приложений на локальной машине и в облаке. Для решениях таких задач могут использоваться в том числе различные средства виртуализации на уровне операционной системы, когда в рамках ОС носителя могут одновременно существовать несколько гостевых ОС. Среди разработанных и используемых на данный момент инструментов был выбран Docker, так как он является единственным кроссплатформенным решением (поддерживает Linux, FreeBSD, Windows и MacOS).

[9]https://aws.amazon.com/ru/api-gateway/

Docker оперирует такими терминами, как образ и контейнер. *Образ* по своей сути является неизменным шаблоном операционной системы и среды выполнения. На основе образов Docker создает и запускает *контейнеры*, в которые пользователь добавляет приложения по собственному усмотрению, используя файл сборки, называемый Dockerfile. В листинге 1 приведен пример использования образа `microsoft/aspnetcore` для запуска `dotnet MyApplication.dll` из директории пользователя `/app`.

```
1 FROM microsoft/aspnetcore
2 WORKDIR /app
3 COPY bin/Debug/publish .
4 ENTRYPOINT["dotnet", "MyApplication.dll"]
```

Листинг 1: Пример кода Dockerfile

В системе Docker также существуют регистры, которые по сути являются хранилищами для образов (как, например GitHub является хранилищем для исходного кода). Одним из популярнейших является регистр DockerHub, который в том числе хранит доступный публично образ[10], идентичный окружению AWS Lambda. Таким образом становится возможным локальная разработка и тестирование приложений в окружении, в котором затем будет исполняться развертывающаяся функция на сервисе Lambda.

[10]https://hub.docker.com/r/lambci/lambda

3. Архитектура и реализация решения

В данной главе представлена архитектура и реализация решения подсветки важного в текстах электронных писем на платформе браузерных расширений.

3.1. Архитектура

Спроектированное решение представляет собой клиент-серверную систему, где клиентом является расширение для браузера Google Chrome, а на сервере реализуются NLP-алгоритмы. Клиент-серверная архитектура позволяет для реализации алгоритмов использовать технологии, недоступные на платформе браузера, и преодолевать некоторые ограничения, такие как размер реализованного расширения. Архитектура представлена на рисунке 3.

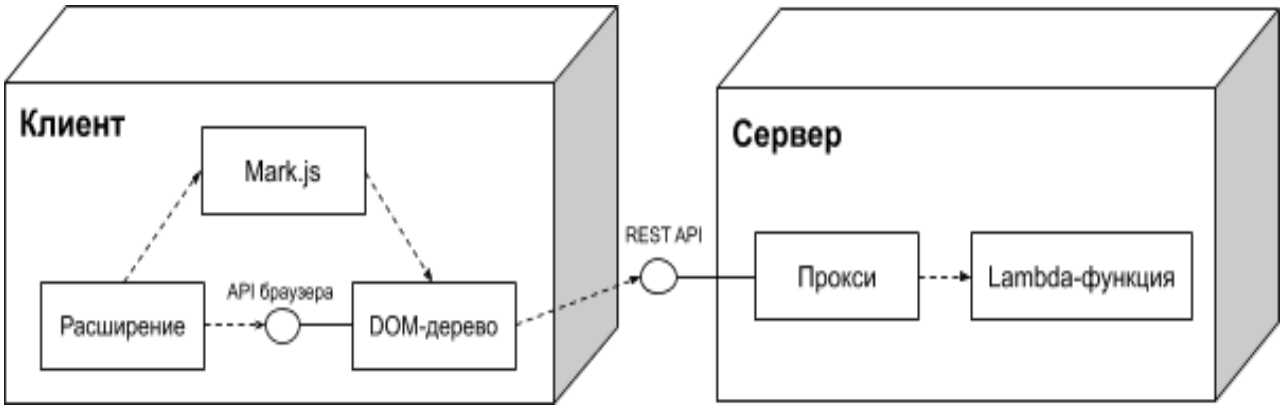

Рис. 3: Клиент-серверная архитектура решения

Итак, на серверной стороне имеются следующие компоненты:

- “Прокси” — отвечает за возврат HTTP-адреса одной из множества реализованных Lambda-функций и предоставляет интерфейс REST API расширению, который использует его для отправки HTTP-запросов, содержащих текст письма;
- “Lambda-функция” — реализованный NLP-алгоритм, определяющий какие слова необходимо подсветить.

На клиентской стороне имеются следующие компоненты:

- “Расширение” — отвечает за разбор HTML-страницы и обмен информацией с сервером. Расширение использует интерфейс, предоставляемый API браузера для доступа к содержимому HTML-страницы и механизму оповещений;
- DOM-дерево — HTML-страница, представленная в виде дерева HTML-узлов;
- Mark.js — JavaScript-библиотека, инкапсулирующая функциональность для визуального отображения подсвеченного текста.

Серверная часть разработана на языке Python и развернута благодаря AWS Lambda. Клиентская часть разработана на языке JavaScript, являющийся стандартом для веб-разработки.

Опишем базовый сценарий работы системы. Основные этапы работы реализованного скрипта можно описать следующим образом:

1. Обход DOM-дерева и сохранение текстовых HTML-узлов.
2. Обмен информацией с сервером.
3. Подсветка нужных слов в тексте.

Первая задача решается путем обхода DOM-дерева с использованием алгоритма поиска в глубину. Корень дерева выбирается в зависимости от используемого почтового Web-сервиса: при помощи паттерна проектирования `Strategy` определяется алгоритм поиска открытого письма в текущей вкладке. В процессе обхода всех HTML-узлов исходного графа в специальном массиве сохраняется ссылка на узел вместе с его текстовым содержимым.

Собранный массив текстов с индексами узлов, из которых он был извлечен, отправляется методом `POST` на HTTP-адрес, который заранее запрашивается от прокси сервера. Для HTTP-запросов используется JavaScript-библиотека Axios[11], предоставляющая более удобный ин-

[11]https://github.com/axios/axios

терфейс для работы с XHR[12], чем стандартный API в браузере Google Chrome. Для более отзывчивого интерфейса в момент отправки также включается таймер, который при слишком длительном ожидании ответа от сервера (на данный момент в качестве такого значения используется 1000 мс) оповещает пользователя, что процесс занимает больше времени, чем обычно. Для оповещений используется Chrome Notifications API[13]. Кроме того, данные оповещения также используются для отображения ошибок расширения.

Когда ответ от сервера получен, расширение начинает процесс подсветки. Для этого используется ранее сохраненный ранее массив со ссылками на текстовые узлы. Для подсветки используется библиотека Mark.js[14], в которой содержатся CSS-стили для обеспечения эффекта подсвечивания. Суть подсветки заключается в удалении старых узлов из DOM-дерева HTML-страницы и вставки новых с CSS-классом, который имеет визуальный эффект подсветки за счет использованной библиотеки. Перестраивание DOM-дерева необходимо, так как в исходном дереве текстовые узлы чаще содежат сразу несколько слов, поэтому для подсветки одного из них приходится в исходный узел добавлять новый со специальным CSS-классом вместо подсвеченного слова. Благодаря такому решению, в том числе, становится возможным отличать одно и то же слово, находящееся в разных позициях (так как оно находится в разных узлах), если NLP-алгоритм использует такой параметр, как позиция слова в предложении.

3.2. Реализация алгоритмов обработки текстов

Все алгоритмы были реализованы при помощи языка Python и таких библиотек как gensim[15], numpy[16], pandas[17], sumy[18] и nltk[19]. Алгорит-

[12]https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest
[13]https://developer.chrome.com/apps/notifications
[14]https://markjs.io/
[15]https://pypi.org/project/gensim/
[16]https://www.numpy.org/
[17]https://pandas.pydata.org/
[18]https://github.com/miso-belica/sumy
[19]https://www.nltk.org/

мы развернуты как серверные приложения в виде Lambda-функций с помощью платформы AWS Lambda.

Все приложения для платформы AWS Lambda должны иметь функцию, определяющую точку входа. Псевдокод такой функции представлен в листинге 2.

```
function handler(event, ... ) {
  parseArguments(event); // Обработка входящих параметров
  response = applyAlgorithm(event.body); // Применение реализованного алгоритма
  return {
    body: toSpans(response) // Ответ со списком слов для подсветки
  }
}
```

Листинг 2: Псевдокод вызываемой функции

Обработка входящих параметров заключается в проверке HTTP-метода запроса. Для метода `Options` ответом функции служит набор заголовков, необходимый для `CORS`. В случае `POST` запроса происходит дальнейшее выполнение кода Lambda-функции.

Для анализа текста в дальнейшем вызывается функция, представляющая интерфейс реализованного алгоритма. Сигнатура такой функции является общей для всех алгоритмов, единственным ее аргументом является список текстовых HTML-узлов, представляющий анализируемый текст. Все реализованные алгоритмы абстрагированы от понятия узлов и работают непосредственно с текстовой информацией: содержимое всех HTML-узлов объединяется в текст, а затем делится на слова. По этой причине на данном этапе “теряется” информация, о том, где находится то или иное слово в исходном DOM-дереве.

Результатом применения алгоритма является список слов с пометками о том, нужно ли подсвечивать каждое конкретное слово. Приложение перед возвратом ответа осуществляет процедуру восстановления узлов по словам, используя функцию `toSpans` (см. листинг 2). Данная операция выполняет поиск узла, содержащего слово, и проставляет метки, сигнализирующей о том, нужно ли подсветить данный узел. Это необходимо, так как расширение для браузера реализует подсветку именно по приходящим в ответе HTML-узлам, как это было описано

выше.

В рамках данной работы было реализовано 7 алгоритмов и один подход со случайной подсветкой:

- TF-IDF с национальным корпусом русского языка,
- TF-IDF с корпусом русской википедии,
- TF-IDF в комбинации с нормами векторных разложений,
- TextRank, выделяющий только предложения или только слова,
- алгоритм, использующий нормы векторных разложений и
- алгоритм, использующий LSA.

Эти алгоритмы являются вариациями алгоритмов, представленных в разделе 2.1 и описанных в [2, 4, 6, 11].

3.3. Клиент-серверное взаимодействие

После публикации расширения в магазине Chrome Web Store менять его исходный код становится затруднительно, так как каждое изменение должно снова пройти процесс модерации, обычно занимающее несколько дней. Это препятствует быстрому переключению алгоритмов, которое расширение использует для подсветки. По этой причине оказался неприемлимым вариант сохранения в коде HTTP-адреса для запросов, и был реализован прокси-сервер в виде еще одной независимой Lambda-функции. Задача этой функции заключается в возврате HTTP-адреса одного из реализованных алгоритмов. На рисунке 4 показано клиент-серверное взаимодействие между расширением и Lambda-функциями.

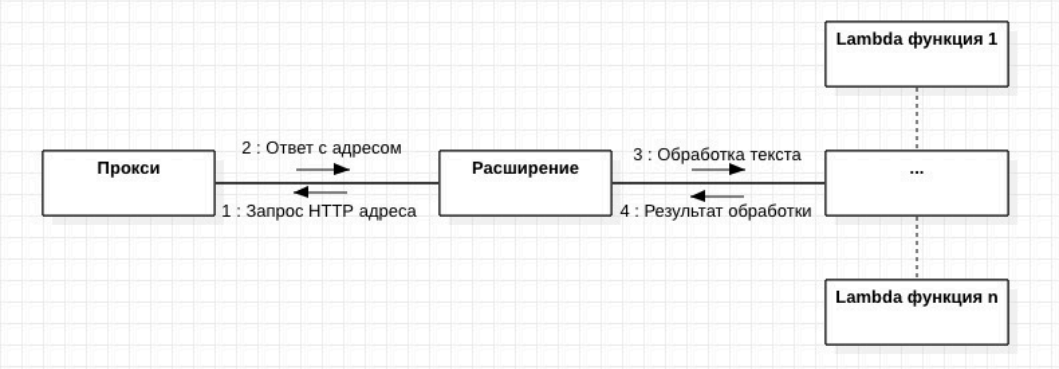

Рис. 4: Клиент-серверное взаимодействие

3.4. Автоматическое развертывание алгоритмов

Все реализованные алгоритмы были развернуты как независимые Lambda-функции в количестве 8 штук. Решение разделять различные NLP-алгоритмы на независимые функции было принято в связи с тем, что платформа AWS Lambda имеет ограничение на размер загружаемого в хранилище архива с исходным кодом в 250 Мб, а размер 8 реализованных алгоритмов превышает данное ограничение в несколько раз. Ручное управление загрузкой функций является рутинным процессом, который необходимо было автоматизировать. Для полной автоматизации развертывания необходимо выполнить следующие шаги.

1. Компиляция зависимостей под целевую ОС.
2. Архивирование исходного кода и всех зависимостей.
3. Обращение к API AWS Lambda для загрузки заархивированого приложения.

Первый пункт оказался необходим, так как некоторые реализации алгоритмов имеют компилируемые зависимости (например, pandas), и это накладывает требование по предварительной компиляции под целевую операционную систему. Для решения данной задачи была использована система Docker и docker-образ Lambda, описанные в Обзоре.

Описанные выше шаги были автоматизированы до возможности вы-

зова одной командой при помощи инструмента Gulp[20], который позволяет императивно описывать задачи, группировать их и выполнять в нужном порядке.

[20]https://gulpjs.com/

4. Сравнительный анализ алгоритмов

В настоящей задаче был важен грамотный выбор схемы проведения сравнительного анализа, так как платформа Яндекс.Толока является платным сервисом, и, соответственно, необходимо в том числе учитывать ограниченность бюджета. В данном разделе описывается модель, используемая для сравнений, а также реализованные дополнительные инструменты для автоматизации анализа и обработки полученных результатов. Кроме этого приводятся результаты экспериментов и доказывается работоспособность используемой модели.

4.1. Модель проведения экспериментов

Для проведения сравнительного анализа реализованных алгоритмов была выбрана модель турнира с олимпийской системой, то есть проигравший в каждом раунде выбывал из соревнования. Данная система помогает снизить количество сравнений: проводится $O(\log_2 n)$ вместо $O(n^2)$ сравнений для варианта, где каждый алгоритм сравнивается со всеми остальными.

Каждый раунд представляет собой сравнение результатов двух алгоритмов, а пользователю предлагается возможность выбора того из них, который лучше осуществляет подсветку важного в тексте. Система оценок с фиксированной шкалой не использовалась по той причине, что разработка грамотной инструкции для асессоров является трудной задачей, которая выходит за рамки данной работы. Пользователи Яндекс.Толоки не являются профессиональными асессорами, что привело бы к субъективным оценкам и проблемам с их нормализацией. Кроме того, в результате консультации с экспертами из области разметки текстов вариант “против всех” также не был использован, так как бинарная оценка является эффективнее (затрачивается меньше времени) при сохранении результата.

Некачественные ответы свойственны, практически, любому опросу. Для улучшения качества ответов использовались следующие фильтры:

- констистентные ответы;
- honeypots;
- время выполнения заданий.

Под консистентными ответами понимались ответы, в которых пользователь одинаково ответил на предложенные варианты подсветки, когда они были предложены более одного раза.

Honeypot — это вариант, при котором заранее известен правильный ответ (в качестве заранее проигравшего варианта использовался алгоритм случайной с низким шансом подсветки каждого слова).

Одно задание состояло из пяти пар скриншотов. Медианное время выполнения одного такого задания было около 47 секунд. Последний фильтр отсеивал пользователей, выполнявших задание быстрее.

4.2. Инструменты для автоматизации проведения сравнений

Сравнительный анализ требовал большого количества однотипных действий. Для ускорения процесса для всей разработанной инфраструктуры была создана специальная обертка, позволяющая агрегировать весь процесс анализа в единую систему с CLI-интерфейсом. Таким образом, были автоматизированы шаги, представленные ниже.

1. Создание скриншотов с результатами подсветки алгоритмами.
2. Формирование задания для Яндекс.Толока.
3. Выгрузка и обработка результатов сравнения.

Первый пункт был реализован с использованием инструмента Selenium[21], который запускает браузер и предоставляет API для взаимодействия с ним. Таким образом, с помощью JavaScript был реализован скрипт, запускающий заранее собранные HTML-письма в браузере Google Chrome

[21] https://www.seleniumhq.org/

при помощи Selenium, а затем этот скрипт осуществлял подсвечивание при помощи реализованного расширения. Результат работы попадал на скриншот и сохранялся в директории для дальнейшего использования.

Задания в Яндекс.Толоку загружаются при помощи файла .tsv (tab-separated values) со специально размеченными колонками, которые содержат входные данные задания, правильный ответ, подсказки и некоторые другие поля. Для формирования таких файлов была написана утилита, но автоматизировать данный пункт полностью не удалось. Задания в Яндекс.Толока используют медиа файлы, загруженные на Яндекс.Диск[22], который не поддерживает API для загрузки файлов на диск, поэтому в процессе сравнительного анализа частично необходима ручная работа.

Как было показано в разделе 2.4, Яндекс.Толока поддерживает API для получения ответов пользователей. Данный пункт так же был автоматизирован при помощи утилиты на языке Python и библиотеки pandas. Утилита осуществляет проверку консистентности ответов пользователей, проверку на honeypots и время выполнения, а также формирует статистику ответов в виде диаграмм. С помощью данной утилиты стало возможным в ходе проведения анализа сравнить более 7500 пар скриншотов.

4.3. Результаты сравнительного анализа

В сравнительном анализе участвовали алгоритмы, выбранные и реализованные выше (в скобках написано название алгоритма, используемое далее в турнирной сетке).

- TF-IDF в комбинации с нормами векторных разложений (TF-IDF#1).
- TF-IDF на национальном корпусе русского языка (TF-IDF#2).
- TF-IDF на корпусе Википедии (TF-IDF#3).

[22]https://disk.yandex.com/

- Алгоритм со случайным выделением (Random).
- Нормы векторных представлений (Embeddings).
- TextRank для предложений (TextRank#1).
- TextRank для ключевых слов (TextRank#2).
- Латентно-семантический анализ (LSA).

Так как анализ проводился по модели турнира, то для удобства чтения результатов проведенные сравнения представлены в виде турнирной таблицы:

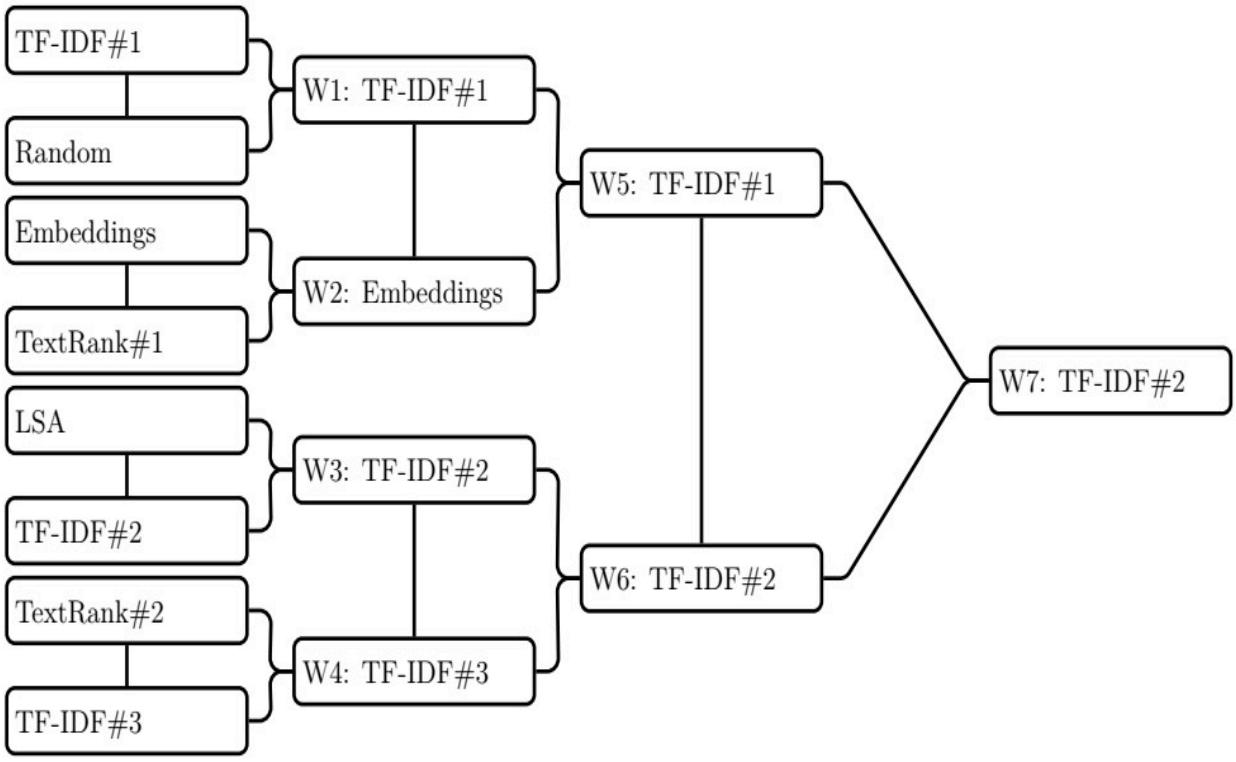

Рис. 5: Итоговая турнирная таблица

Как видно из турнирной сетки, оценивающая группа выбрала реализацию алгоритма TF-IDF на национальном корпусе русского языка с подсветкой всех цифр как наиболее качественную.

Для дополнительной проверки полученных результатов оценивающей группе было предложено отдельное задание, в котором им предлагалось оценить скриншот с подсветкой по следующей бинарной метрике: является ли подсветка полезной. Идея заключалась в том, чтобы проверить существует ли корреляция между результатами такого опроса и результатами турнира.

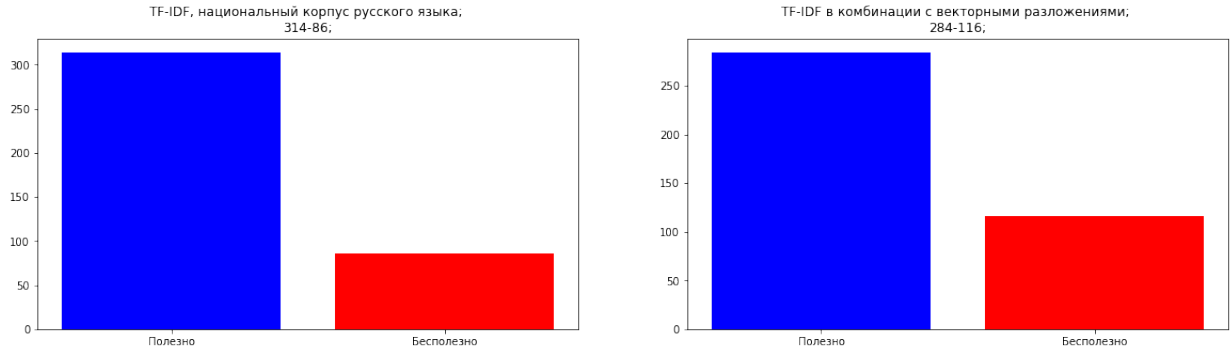

Рис. 6: Оценка полезности подсветки двух алгоритмов

Результаты, представленные на рисунке 6 коррелируют с результатами турнира. Это закрепляет результат сравнительного анализа, а также подтверждает работоспособность схемы проведения анализа по модели турнира.

Заключение

В ходе данной работы были получены следующие результаты.

- Выбраны следующие алгоритмы NLP для подсветки важного в тексте: TF-IDF в трех вариациях, TextRank в двух вариациях, алгоритм с использованием норм векторов, алгоритм с использованием LSA. Выбрана метрика качества “оценивающая группа”.

- Спроектирована и реализована клиент-серверная архитектура на базе браузера Google Chrome (использованы технологии Яндекс.Толока, AWS Lambda, Docker и язык JavaScript), а также реализованы выбранные алгоритмы (Python).

- Реализованы дополнительные утилиты для автоматизации процесса проведения экспериментов (Python, JavaScript). Для реализованных алгоритмов спроектирован и выполнен на базе технологии Яндекс.Толока эксперимент с целью выявления лучшего алгоритма. Лучшем алгоритмом оказался TF-IDF на национальном корпусе русского языка.

Из полученных результатов можно сделать вывод о том, что подсветка 30% самых важных слов текста с использованием TF-IDF в качестве меры важности может быть успешно применена для полезной подсветки слов в тексте. Для улучшения работы алгоритма может быть проведена работа над подсветкой именованных сущностей, таких как даты, адреса, имена и другие. В совокупности с подсветкой, осуществляемой настоящим алгоритмом, это способно существенно облегчить взаимодействие пользователя с электронной почтой.

Список литературы

[1] A Click Sequence Model for Web Search / Alexey Borisov, Martijn Wardenaar, Ilya Markov, Maarten de Rijke. — The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval pp. 45-54, 2018.

[2] Efficient Estimation of Word Representations in Vector Space / Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean // arXiv preprint arXiv:1301.3781. — 2013.

[3] An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation // arXiv preprint arXiv:1607.05368.

[4] Gong Yihong, Liu Xin. Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis. — Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, New Orleans, Louisiana, United States 2001, pp. 19-25, 2001.

[5] LCSTS: A Large Scale Chinese Short Text Summarization Dataset // arXiv preprint arXiv:1506.05865. — 2016.

[6] Mihalcea Rada, Tarau Paul. Textrank: Bringing order into texts. — Proceedings of EMNLP 2004. pp. 404–411. Association for Computational Linguistics, Barcelona, Spain (July 2004), 2004.

[7] Resnik Philip, Lin Jimmy. Evaluation of NLP Systems. The Handbook of Computational Linguistics and Natural Language Processing, pp. 271–295. — 2010.

[8] Sequence-to-Sequence Learning for Task-oriented Dialogue with Dialogue State Representation // Proceedings of the 27th International Conference on Computational Linguistics, pages 3781–3792. — 2018.

[9] Serverless Computing: Current Trends and Open Problems /

Ioana Baldini, Paul Castro, Kerry Chang et al. // arXiv preprint arXiv:1706.03178. — 2017.

[10] Sun Chi, Qiu Luyao Huang Xipeng. Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence // arXiv preprint arXiv:1903.09588. — 2019.

[11] TF-IDF. What does tf-idf mean? — Access mode: `http://www.tfidf.com/`.

[12] Tensor2Tensor for Neural Machine Translation // arXiv preprint arXiv:1803.07416. — 2016.

[13] VQA: Visual Question Answering / Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu et al. // The IEEE International Conference on Computer Vision (ICCV). — 2015. — December.