Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Петрозаводский государственный университет»
Физико-технический институт

Кафедра физики твёрдого тела

Копытов Павел Вячеславович

РАЗРАБОТКА ВЕБ-СЕРВИСА ПЕРСОНАЛИЗИРОВАННЫХ РЕКОМЕНДАЦИЙ ДЛЯ СИСТЕМАТИЗАЦИИ И УПРАВЛЕНИЯ ЗНАНИЯМИ

выпускная квалификационная работа
обучающегося 4 курса очной формы обучения
по направлению подготовки 09.03.01 Информатика и вычислительная техника
профиль «Автоматизированное управление бизнес-процессами и финансами-ПБ»

Допущен к защите:
Зав. кафедрой _____,
_____
_____
«___»_____20___г.

Научный руководитель:
кандидат физ.-мат. наук, доцент
_____ В.Б.Пикулев
«___» _____20___г.

Петрозаводск 2021

РЕФЕРАТ

Отчёт содержит 51 с., 2 ч., 15 рис., 7 табл., 21 источник, 2 прил.

ВЕКТОРНОЕ ПРЕДСТАВЛЕНИЕ СЛОВ, WORD EMBEDDING, WORD2VEC, DOC2VEC, NATURAL LANGUAGE PROCESSING, РЕКОМЕНДАТЕЛЬНЫЕ СИСТЕМЫ

Цель работы — исследование и разработка рекомендательной системы с использованием современных методов векторизации слов для статей из различных источников. Система собирает новые статьи с доверенных сайтов, а также прочие статьи, прошедшие модерацию, обрабатывает их, а затем — рекомендует пользователю в соответствии с внутренними правилами. Для рекомендаций используется векторизация документов, при которой каждый документ представляется в виде вектора в n-мерном пространстве (общем для всех документов). Вектора документов — результат обучения нейросетевой модели (техника Doc2Vec). После обучения с векторами документов возможно производить математические операции, например, найти ближайший к текущему вектор документа. При этом, «ближайший» в векторном пространстве документ будет близок к данному семантически. Именно на этом принципе и основана рекомендательная система.

В ходе работы проведён аналитический обзор современной научно-технической литературы, затрагивающей следующие проблемы: извлечение текстовой информации с веб-страниц, обработка и очистка текстовой информации, технологии векторизации слов, рекомендательные системы (в том числе — с использованием векторизации слов).

В первом разделе рассматриваются теоретические основы работы, методы извлечения и обработки текста, а также возможности создания рекомендательной системы с использованием современных нейросетевых методов.

Во втором разделе рассматриваются существующие инструменты для работы с текстом (обработка и подготовка). Исследуется время обучения модели Doc2Vec в зависимости от ряда параметров. Проектируется и разрабатывается рекомендательная система на основе техник векторизации.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

NLP (анг. Natural Language Processing) — обработка текстов на естественном языке. Наука, которая изучает возможности компьютера анализировать и синтезировать тексты на естественном языке.

Модель (в машинном обучении) — результат, получаемый при обучении машинного алгоритма с помощью данных. После завершения обучения модель выдаёт выходные данные, когда в неё вводятся входные данные.

Датасет (в машинном обучении) — набор данных, который применяется для обучения модели машинного обучения.

Эпоха (в машинном обучении) — прохождение датасета через нейронную сеть в прямом и обратном направлении один раз.

Парсер — программа, которая преобразует входной текст (или другую информацию) в нужный пользователю формат.

Контент-вектор пользователя (контент-профиль) — n-мерный вектор (в векторном пространстве статей, обработанных Doc2Vec), определяющий предпочтения пользователя. Изменяется после перехода на ту или иную статью.

ВВЕДЕНИЕ

Информации в Интернете становится все больше. Информационные и новостные сайты в погоне за трафиком добавляют новые материалы на свои главные страницы. Старые же статьи, на которые журналисты потратили свои силы и время, постепенно сдвигаются вниз выдачи, а потом и вовсе попадают в архив. Из архива же эти данные извлечь затруднительно.

Два основных способа получить такие статьи: перекрёстные рекомендации с других материалов на сайте и выдача поисковиков. Они, однако, не лишены недостатков: перекрёстные рекомендации работают только на данном сайте, а для поиска, например, в Google, пользователю нужно знать, что искать.

Сайты с совокупным контентом, подобные видеохостингу Youtube, решают эту проблему единой системой рекомендаций. Видеороликов на Youtube очень много, а пользователь зачастую вовсе не знает, что ему нужно. Система рекомендаций позволяет находить новую информацию и тематики.

В данной работе предпринимается попытка создать рекомендательную систему для статей, полученных с разных сайтов и из разных источников. Для этого используются современные инструменты извлечения, обработки и классификации текстов, а также средства быстрой разработки веб-приложений.

Акцент данной работы — рекомендательная система именно для длинных статей (репортажей, историй, расследований, разборов). Системы сбора и управления такого рода контентом куда менее распространены, чем обычные агрегаторы новостей.

ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Рекомендательные системы

Рекомендательная система (далее – РС) — набор программных комплексов, который систематизирует, классифицирует и фильтрует контент, а затем предоставляет его пользователю, опираясь на содержание контента и предшествующее поведение пользователя.

Согласно одному из источников существует два основных вида РС по механизму рекомендаций [1].

- Коллаборативные.
- По содержанию контента.

Коллаборативные системы используют информацию о поведении пользователя, находят других, похожих на него, а затем составляют набор материалов, которые могут заинтересовать пользователя.

Этот подход, однако, имеет ряд существенных ограничений, которые не позволяют использовать его в данной работе.

- Разрежённость матрицы пользовательских рейтингов. Матрица пользовательских рейтингов представляет собой таблицу, на которую отображаются пользователи вместе с оценёнными ими объектами системы. Пример – таблица 1.1. Большинство пользователей никак не оценивает большинство объектов (они просто не пересекаются) и из-за этого в матрице появляются бреши, которые необходимо как-то закрывать и которые мешают вырабатывать корректные рекомендации для пользователей.
- Проблема «холодного старта». Коллаборативные рекомендации основаны на поведении пользователей. Система же в данной работе изначально создаётся с нуля и, следовательно, пользователей не имеет. Нет и информации, с которой можно работать.

Объекты

Пользователи		1	2	3	…	*i*	…	*m*
	1	2	5	4				
	2		1					9
	3	7	6					
	:	1		3				
	u					6		
	:		4					
	n							3
	s	2	8	?		1		?

Таблица 1.1 – Матрица пользовательских оценок [1]. На основании существующих рейтингов производится попытка предсказать оценку пользователя *S* для тех объектов, которые он ещё не оценил.

Системы, основанные на содержании контента (далее – контентные) [1, 2] используют иной подход. В данном случае элемент контента (например, статья) представляется как объект с набором определённых параметров. Пользователь также представляется как объект с параметрами. Профиль пользователя будет корректироваться в зависимости от просмотренных элементов (дополнительно можно учитывать оценку, время просмотра и т.д.). Благодаря этому система сможет рекомендовать более релевантные результаты человеку.

Некоторыми исследователями контентный подход рассматривается как одна из проблем «извлечения данных» (Information Retrieval). Действительно, ведь от качества извлечения информации зависит точность его последующей обработки.

Основные способы извлечения и предобработки данных будут рассмотрены в главе 1.3.

Методы обработки и классификации, такие как: *TF-IDF, Word2Vec* и *Doc2Vec* будут рассмотрены в главе 1.4.

В данной работе будет использоваться исключительно контентный подход. Это связано с отсутствием сколько-нибудь существенной пользовательской базы. Ещё один важный плюс контентного подхода — возможность сделать его полностью прозрачным для пользователя, убрав тематики в привычном виде (как ключевые слова) из системы. Темы и схожесть между ними будут представлены математическими методами, а это, в свою очередь, открывает интересные возможности для работы с контекстом документов.

1.2 Извлечение информации

1.2.1 Общая информация об извлечении данных из Интернета

Для понимания процессов извлечения информации из Интернета сначала обозначим некоторые определения.

Скрапинг (анг. scraping) — получение определённой информации с веб страницы. Это может быть вся веб-страница или отдельные её части (теги и т.д.) [3].

Краулинг (анг. crawling) — процесс последовательного скрапинга целых сайтов. Программа (*краулер)* перемещается по сайту и скачивает страницы или их части.

1.2.2 Этические и правовые проблемы скрапинга и Data Mining

Помимо сугубо технических аспектов скрапинга: технологий, методик обработки и подобного, нужно рассмотреть ещё и правовую и этическую стороны данного вопроса.

В. Кротов и другие в своей статье [4] выводят ряд вопросов, которые исследователь и разработчик может задать себе при работе над проектом, использующим скрапинг чужих сайтов. Вопросы эти позволяют понять — насколько проект легален и этически корректен. Попытаемся ответить на каждый из них в контексте текущей работы.

- Присутствует ли на данном сайте явный запрет на скрапинг и краулинг?
- Явно ли защищена авторскими правами информация на сайте?
- Будет ли полученная информация использоваться каким-либо незаконным или потенциально опасным образом?
- Может ли скрапинг или краулинг данного сайта нанести прямой урон сайту или хостингу, на котором сайт находится?
- Можно ли нарушить чью-либо приватность вследствие извлечения информации с сайта?
- Может ли информация с сайта раскрыть подробности о деятельности компании — владельца сайта?
- Повлияет ли извлечение информации с сайта на его ценность, как источника информации?

Легко видеть, что на большинство этих вопросов, в силу широты проекта, точно ответить практически невозможно. Рассмотрим их, однако, концептуально.

Из-за того, что многие сайты в системе могут добавляться туда самими пользователями есть совсем немного возможностей проверить — разрешено ли скрапить конкретный сайт. Это можно сделать, например, используя базы данных подобных сайтов. Как показал беглый поиск в англоязычном Интернете — таких баз пока не существует. Это может стать темой отдельного исследования.

Будем в дальнейшем предполагать, что все статьи защищены авторскими правами, которые, согласно ГК РФ [5], возникают в момент публикации статьи. Так как система сама по себе не будет публиковать статьи, а лишь их заголовки, то нарушения авторских прав здесь быть не должно.

Публичную информацию, такую, как статьи, затруднительно использовать для нанесения прямого вреда. Читатель здесь может вспомнить про модерацию. Касательно неё необходимо отметить два момента.

- В случае использования системы в офлайн режиме пользователь сам решает, какие сайты добавлять в ротацию, а какие — нет. Это соответствует философии приложения, которое предоставляет лишь инструмент манипулирования информацией. Какой информацией — зависит от пользователя.
- Для онлайн сервиса модерация необходима. Опять же, если пользователь готов добавлять неблагонадёжные сайты в персональную ротацию, пусть получает и страницы с них. Однако, если это касается общедоступного набора статей, то уже требуется человек или команда, которые будут фильтровать сайты по неким формальным признакам (в противном случае рекомендательную систему может заблокировать, например, Роскомнадзор или другое надзорное ведомство).

Многие современные сайты хорошо защищены от распределённых атак типа DOS. Сервисы, подобные Cloudflare, 1cloud и Qurator, позволяют обнаружить и заблокировать вредоносный трафик. Данный проект не ставит целью нарушение работы серверов, на которых находятся статьи. Самое простое, что можно сделать для предотвращения чрезмерного исходящего трафика — поставить интервал отправки запросов GET на разумную величину (1 секунда и более). К тому же, единожды получив и обработав статью, системе более не потребуется загружать её.

Вопросы 5 и 6 естественным образом закрываются самим фактом того, что система скачивает *общедоступную статью*, которая была опубликована для широкого доступа.

И наконец 7 вопрос, ответ на который будет, как ни странно, строго положительным. Да, повлияет, но может сделать источник куда ценнее! Ведь благодаря системе рекомендаций статью найдёт больше людей, которым она будет действительно нужна и интересна.

Исходя из вышесказанного можно сделать вывод, что система (по крайней мере пока она позиционирует себя как open-source), не нарушает закон и этические нормы. Этому способствует ещё и характер рекомендаций.

Отображаемый для пользователя объект рекомендации включает в себя только:

– заголовок статьи;
– ссылку на сайт или локальное хранилище;
– тематику или набор тематик (опционально).

Система генерирует трафик на сайты (в случае веб-страниц), при работе же с локальными файлами она лишь указывает на конкретную статью или документ (даже если они отсутствуют сейчас на компьютере пользователя).

1.2 Обработка данных

После получения текстовой информации её необходимо соответствующим образом обработать. «Сырой» текст, поданный на вход классифицирующего механизма, породит множество нежеланных ошибок и неточностей.

В общем виде обработку можно разделить на несколько частей.

– Убрать из текста знаки препинания, спецсимволы и escape-последовательности. Достигается применением простого регулярного выражения.
– Убрать из текста стоп-слова. Лексемы: «*on», «at», «the», «a», «an», «and*» не несут никакой смысловой ценности, если только они не в связке с семантически «тяжёлыми» словами. Популярные библиотеки обработки текстов (например, NLTK) поддерживают удаление таких слов из текста.
– Лемматизировать текст. Лемматизация – процесс нахождения всех однокоренных слов. Все они в последствии заменяются на «основное слово» [6]. Лемматизация особенно актуальна для русского языка, где у одного слова может быть больше десятка производных.

1.3 Методы классификации текстов

Расшифруем сначала ряд аббревиатур и терминов.

NLP (*анг. Natural Language Processing*) — обработка естественного языка, одно из направлений искусственного интеллекта и математической логики, которое изучает проблемы понимания и генерации обыкновенного текста компьютером.

Корпус — набор текстов, который будет помещён в модель для дальнейшего её обучения.

Словарь — все уникальные включения слов в корпусе. Модель не сможет обработать слово, если оно предварительно не было помещено в словарь.

Рассмотрим по порядку некоторые методы классификации, начиная с тривиальных и заканчивая подходами с использованием машинного обучения и нейросетей.

1.3.1 TF-IDF

TF-IDF (анг. *Term Frequency – Inverse Document Frequency*) — статистический метод, который учитывает частоту появления слова в тексте и наборе текстов [7]. Комбинируется из двух показателей:

– TF (*term frequency*) — частота появления слова в документе. Рассчитывается по формуле:
tf (t,d) = n_t */* N_d
Где n_t количество появлений слова t документе d, а N_d — общее количество слов в документе d.
– IDF (*inverse document frequency*) — обратная частота появления в документах. Рассчитывается по формуле:
idf (t) = log (N / (df + 1))
Где *df* отражает количество появлений слова t в наборе документов N.

Рейтинг для данного слова в данном документе (из набора) будет означать то, насколько это слово существенно для текста. Если оно попадается в массиве документов часто (например, предлог или союз), то его IDF будет расти, тем самым понижая важность слова.

TF-IDF — очень популярная метрика для работы с текстами в рекомендательных системах [8]. Методика проста, реализована во многих программных пакетах (в том числе в

библиотеках Python), к тому же, объем вычислений здесь относительно невелик, особенно если сравнивать с нейросетевыми подходами.

Все это делает TF-IDF прекрасным кандидатом на роль классификатора для онлайн рекомендательной системы, для которой важна скорость и быстрый отклик.

TF-IDF хороша и как вспомогательная система в дополнение к методам «встраивания слов» (word embedding). Согласно исследованиям, применение TF-IDF с такими моделями, как Word2Vec и Doc2Vec повышает точность в среднем на два процента [9]. Про эти модели будет рассказано ниже.

1.3.3 Word embedding

1.3.3.1 Word2Vec

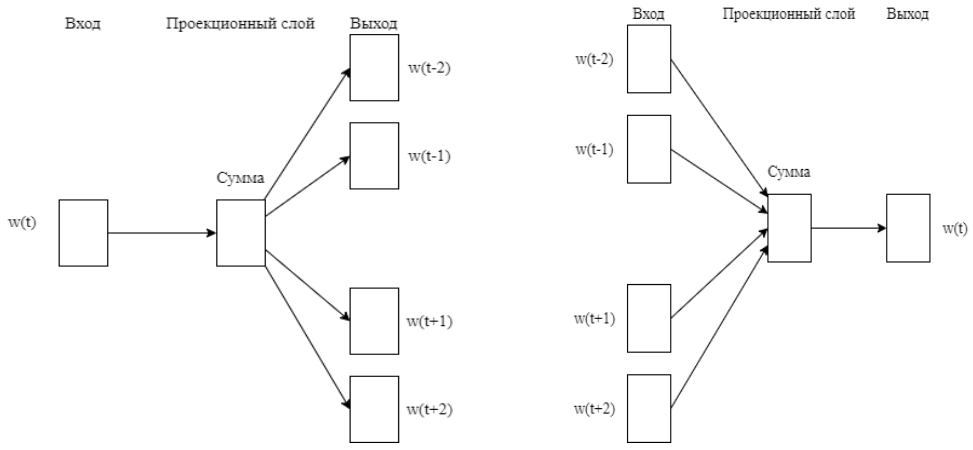
В 2013 году в работе [10] была предложена новая технология работы с текстами, использующая нейросети, под названием Word2Vec.

Сущность этого подхода состоит в построении многомерного вектора (как правило, от 50 до 1000 измерений) для каждого слова в словаре.

В Word2Vec для обучения модели применяется два подхода:

– CBOW (Continuous bag of words).
– Skip-gram.

Они отображены на рисунках 1.1, *а* и 1.1, *б* соответственно. Skip-gram пытается предсказать окружающие слова по входному слову, после чего корректирует вектор слова в нужную сторону. CBOW, наоборот, предсказывает слово по его окружению (контекстному окну) и тоже сообразно корректирует вектор.



а) б)

Рисунок 1.1 – Алгоритмы Skip-Gram и CBOW [10].

Уникальность Word2Vec (и, как впоследствии будет видно, Doc2Vec) заключается в математическом представлении семантического значения слова. Благодаря этому над словами можно производить математические операции (используя косинусную схожесть векторов).

Вот некоторые примеры (переведены с английского):

– «Человек» минус «животное» равно «этика»;
– «президент» минус «власть» равно «премьер-министр»;
– «библиотека» минус «книги» равно «зал».

Также можно рассмотреть взаимоотношения слов.

Рассмотрим отношение «Франция – Париж».

Тогда для других регионов или стран результат будет следующим:

– Италия — Рим;
– Япония — Токио;
– Флорида — Таллахаси.

1.4.3.2 Doc2Vec

Doc2Vec — надстройка на технологией Word2Vec [11], которая появилась в 2014 году. Здесь, помимо векторов слов используется ещё так называемый Paragraph ID (при старте задаётся случайно).

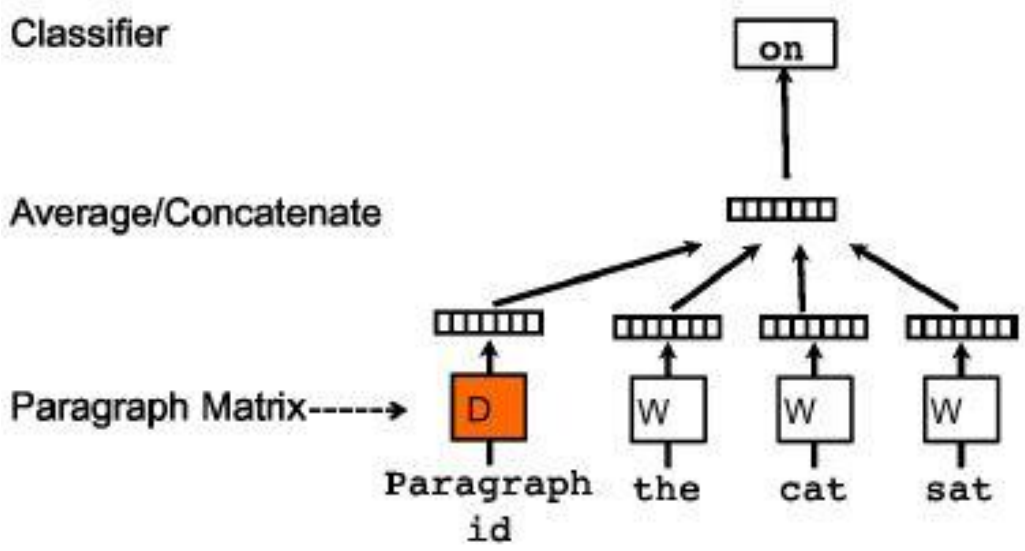


Рисунок 1.2 – Алгоритм PV-DM

Это, по сути, тоже вектор слова, только содержит он информацию о контексте (теме) всего документа. Потом этот вектор добавляется или усредняется с векторами слов внутри документа.

На рисунке 1.2 показан метод PV-DM (distributed memory), при его использовании обучаются и вектора слов, и вектора документов. Второй метод, PV-DBOW (distributed bag of words), выключает вектора слов и тренирует исключительно Paragraph ID. Этот метод отображён на рисунке 1.3. В каждый момент обучения из документа берётся набор слов (равный размеру окна, заданного пользователем). Paragraph ID обучается таким образом, чтобы предсказывать слова для этого небольшого окна. Действия повторяются многократно и для разных слов из документа.

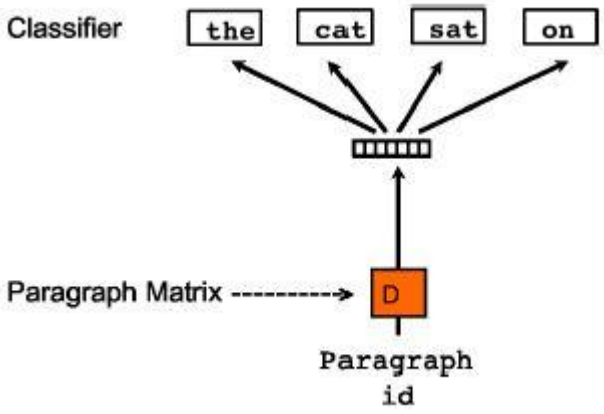


Рисунок 1.3 – Алгоритм PV-DBOW

1.5 Разработка рекомендательной системы. Постановка задачи

На основе вышеизложенных технологий утвердим задачу разработки.

- Разрабатываемая рекомендательная система должна будет использовать инструменты обработки и извлечения информации для скачивания текстов с избранных сайтов (или с отдельных страниц) и их очистки.
- После обработки статьи помещаются в базу данных, откуда поступают на вход нейросетевой модели. Нейросетевая модель обучается. После обучения модель может предоставить n-мерный вектор для каждой статьи из базы данных (это возможно и для статей, которые ещё не принимали участие в обучении; точность при этом будет невелика).
- После регистрации пользователь выбирает из набора предложенных ему статей наиболее ему интересные. Система формирует контент-профиль пользователя (аналогичный n-мерный вектор).
- Далее пользователю предлагаются статьи, чей n-мерный вектор ближе всего к контент-профилю пользователя в n-мерном векторном пространстве.
- При переходе на ту или иную статью контент-профиль пользователя корректируется в сторону вектора статьи.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Методы извлечения данных

Способы извлечения информации из Интернета можно разделить по методу:

- Создание скрапера или использование существующего;
- Извлечение информации, накопленной сторонними организациями (CommonCrawl);
- Прочие методы.

Рассмотрим плюсы и минусы этих методов.

Применение скрапера (краулера) позволяет полностью контролировать процесс получения информации. С другой стороны, разработать качественный сборщик информации трудно, а значит, требуется использовать существующий платный или бесплатный продукт, что снижает контроль над процессом. Кроме того, не существует единого формата всех сайтов. Какие-то ресурсы придерживаются стандартов доступности [12], другие же — по-прежнему используют теги «div» и «span» везде, где это возможно. Создание собственного скрапера – задача нетривиальная и подходящая для отдельного исследования по теме. Следует также помнить, что некоторые сайты намеренно искажают структуру генерируемого HTML, что ещё больше затрудняет извлечение информации.

Применение уже собранной информации имеет свои проблемы. Подготовленная база данных может иметь специфический формат, в котором будут отсутствовать нужные исследователю элементы данных. Возможно также, что информация, которая легко извлекалась с оригинальной страницы при помощи понимания контекста (например, определённый тег для заголовка статьи), будет безвозвратно утеряна при сборе (в базе данных текст станет сплошным, без сегментирования). Кроме вышеуказанного, архивы также не подходят для получения актуальной информации. CommonCrawl, рассмотренный ниже, обновляет свою базу раз в месяц.

В этой работе не ставилась задача разработать краулер. У него есть свои особенности: например, многие сайты умеют распознавать подобные программы и блокировать их IP-адреса. Это может создать дополнительные неудобства для исследователя.

Принимая во внимание указанные тезисы, приходим к выводу, что следует использовать комбинацию из двух подходов — старую информацию получать из архивов (по крайней мере можно получать URL статей, а затем уже анализировать контент своим

скрапером), а актуальную — собирать с сайтов в реальном режиме (например, раз в несколько часов).

2.1.1 Использование библиотеки Newspaper для скрапинга

Язык Python предоставляет возможность лёгкого подключения сторонних библиотек. Воспользуемся этим преимуществом.

Попробуем применить библиотеку *newspaper* [13]. Она основана на библиотеке lxml, парсере BeautifulSoup и некоторых других инструментах.

Для установки воспользуемся пакетным менеджером pip:

```
pip install newspaper3
```

Базовые возможности будут продемонстрированы в режиме интерактивной подсказки (в тексте отличается наличием «>>>» перед строками кода).

Проверим инструмент на двух статьях:

– на русском языке [14];
– англоязычной [15].

Из модуля newspaper импортируем класс Article для работы со статьями:

```
>>> from newspaper import Article
```

Для удобства поместим URL статей в переменные, url_ru и url_eng, соответственно.

Затем создадим объект русскоязычной статьи и скачаем её:

```
>>> article_ru = Article(url_ru)
>>> article_ru.download()
```

Теперь можно запустить парсер и вычленить отдельные элементы статьи.

```
>>> article_ru.parse()
```

Это позволит работать не только с HTML страницы (с этим справился бы и встроенный модуль requests), но и с текстом статьи:

```
>>> article_ru.text
```

'Об обычной проводке\n\n\n\nПолиэтиленовые клеммники\n\nНельзя зажимать алюминий. Алюминиевая жила обладает…'

Программа успешно извлекла полный тест статьи и поместила его в отдельную переменную объекта article_ru. Легко увидеть, что текст нуждается в дополнительной обработке – требуется убрать знаки препинания, а также символы конца строки. Для обработки с помощью нейросетей потребуется ещё убрать так называемые «стоп-слова» (часто повторяющиеся слова, которые не несут смысловой нагрузки). Для того чтобы убедиться в необходимости такой обработки, применим встроенный метод nlp(). Он позволяет исследователю использовать возможности библиотеки NLTK — популярного инструмента NLP.

>>> article_ru.nlp()

Получим список ключевых слов.

>>> article_ru.keywords

['провода', 'не', 'на', 'знак', 'а', 'провод', 'можно', 'от', 'восклицательный', 'с', 'в', 'и', 'скручивать', 'прекратите']

Нетрудно заметить, что в списке есть множество бесполезных с точки зрения контекста слов: частицы, союзы и предлоги не помогут узнать тему документа. Инструменты обработки текста будут рассмотрены в последующих главах этой работы.

>>> article_eng.keywords

['climate', 'countries', 'world', 'emissions', 'need', 'pandemic', 'warming', 'change', 'report', 'united', 'factor', 'degrees', 'footprint', 'warns', 'slow', 'rich', 'cut', 'worlds', 'states']

При работе с англоязычной статьёй, однако, релевантность ключевых слов много выше, чем это было с примером статьи на русском. Текст повествует об изменениях климата (глобальном потеплении), а среди ключевых слов присутствует сразу несколько непосредственно относящихся к теме.

Немаловажным для системы рекомендаций, в частности для UI сайта, будет возможность извлечь заголовок статьи.

>>> article_ru.title

'Прекратите скручивать (восклицательный знак)'

>>> article_eng.title

'The world’s rich need to cut their carbon footprint by a factor of 30 to slow climate change, U.N. warns'

Newspaper получил корректные заголовки.

При подробном рассмотрении работы Newspaper видно, что заголовки он получает весьма тривиальным способом — находит тег главного заголовка <h1> и подбирает содержащийся в нем текст.

Помимо извлечения отдельных статей Newspaper предоставляет исследователям мощный инструмент создания новостных источников. Он позволяет создать из какого-то отдельного сайта источник, выделить оттуда тематические разделы и работать с ними по отдельности.

2.1.2 Использование архива CommonCrawl

Второй метод, который можно использовать для получения набора статей – извлечение уже собранных данных из внешнего источника. В качестве примера в этой работе используется сервис CommonCrawl.org [16], который парсит сайты в Интернете и собирает её в публично доступные архивы на Amazon Cloud.

Рассмотрим преимущества и недостатки такого подхода.

Плюсы:

- На CommonCrawl сайты собираются начиная с 2011 года. Это означает, что помимо недавних веб-станиц (от одного до нескольких месяцев давности), возможно получить и очень старые материалы, которые, возможно, уже убраны с целевого сайта.
- Объем информации удобно разбит на архивы по годам. Можно скачать необходимые файлы и работать с ними офлайн. Онлайн обработка тоже доступна (возможно получить доступ к данным внутри архивов по отступу и извлечь информацию).
- Информация хранится в нескольких представлениях. WARC (Web ARCive) хранит весь HTML на указанную временную метку, WAT —метаданные, а WET — исключительно текст. Эти форматы позволяют соответствующим инструментам с лёгкостью извлекать нужные данные из больших архивов, а затем и обрабатывать

их. Возможности Python, позволяют, в свою очередь, не загружать файл в память целиком, но работать с ним построчно.

Минусы:

– Устаревшие данные. CommonCrawl собирает информацию раз в месяц, затем пакует её в архивы и размещает в публично доступном облачном хранилище. Это означает, что доступа к только что вышедшим материалам нет.
– Английский язык. Сейчас в CommonCrawl уже существует архив русскоязычного Интернета, однако в основном в облачном хранилище имеются лишь англоязычные материалы. Не выяснена и поддержка других распространённых языков.
– Необходимость скачивать большие объёмы данных одновременно. Эта проблема относиться скорее к исследовательским условиям — отсутствию достаточного объёма памяти на диске. По меркам небольшой IT компании архивы невелики (несколько терабайт данных) и могут быть легко скачаны, обработаны (токенизированы) и сжаты до удобных величин.

2.2 Подготовка текста

После извлечения текста необходимо его обработать. Рассмотрим возможности Python и его библиотек.

2.2.1 Токенизация. Очистка текста от знаков препинания и спецсимволов

Текст следует разбить на лексемы. Воспользуемся библиотекой NLTK [17].

```
tokens = word_tokenize("I'm a southern salesman.")
```

```
['I', "'m", 'a', 'southern', 'salesman', '.']
```

При испытаниях функции word_tokenize обнаружилось, что она работает очень медленно. В частности, датасет «All the news» обрабатывался бы несколько недель. Поэтому воспользуемся встроенными средствами Python.

```
text = re.sub(r"[^a-zA-Z0-9]+", ' ', row["content"])
for word in text.split():
    if len(word) < 2:
        continue
    tokens.append(word)
```

Обратите внимание, что в данном случае сначала происходит очистка от знаков препинания и спецсимволов (с помощью регулярного выражения), а уже затем — токенизация. Нельзя сказать, что это универсальный подход, ведь в английском языке, например, есть сокращённые формы (don’t, aren’t). При выбранном методе они разбиваются на два слова, а буква «t» отбрасывается последующим ограничением на длину. Подобный подход, однако, можно обосновать, если вспомнить об использовании Doc2Vec в данной рекомендательной системе. Модель безразлична к форме слова, она смотрит лишь на окружение лексемы. Следовательно, нет разницы, будет в словаре «don» или «don’t». С другой стороны, грубый фильтр на длину слова полностью выбрасывает из текстов лексему «I», очень существенную для контекста. В данной работы было отдано предпочтение скорости обработки, но этот вопрос, несомненно, достоин дальнейшей дискуссии.

2.2.2 Очистка текста от стоп-слов и лемматизация текста

Очистка от стоп-слов (не несущих контекста лексем) также производится библиотекой NLTK.

```
from nltk.corpus import stopwords
nltk.download('stopwords')
text = "Nick likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]
['Nick', 'likes', 'play', 'football', ',', 'however', 'fond', 'tennis', '.']
```

Такого рода операции на больших датасетах, как уже упоминалось, занимают очень много времени.

Лемматизация, в свою очередь, преобразует производные слова (например, с окончанием), в оригинальные. Это позволяет избежать избыточного разнообразия слов и повысить качество обучения модели.

```
>>> from nltk.stem import WordNetLemmatizer
>>> lemmatizer = WordNetLemmatizer()
>>> lemmatizer.lemmatize("newspapers")
'newspaper'
lemmatizer.lemmatize("constructions")
'construction'
```

2.3 Обучение нейросетевой модели

2.3.1 Применение алгоритма Doc2Vec

Так как рекомендательная система будет работать с документами, следует использовать соответствующую ей технологию преобразования текста во внутреннее представление (которое будет применяться в самой системе рекомендаций). Для этого воспользуемся технологией Doc2Vec, которая была описана в I части работы.

Для обучения модели были задействованы два датасета – «text8» [18] (для тестирования модели) и «All the news» [19] (для разработки прототипа рекомендательной системы).

Преимущество применения Doc2Vec в рекомендательной системе заключается в подходе к обучению. Это обучение без учителя — модели подаётся на вход некий объем текстов (корпус), она тренируется указанное количество эпох, после чего её можно использовать для прослеживания связей между статьями.

Следовательно, модели не требуется никаких тегированных и тестовых данных — она обучается сама, а все последующее взаимодействие с моделью происходит по принципу общения с «черным ящиком».

Так, контентный профиль пользователя тоже будет, по сути своей, n-мерным вектором, где n — соответствует мерности векторов документов модели Doc2Vec. При переходе на статью (или при оценке) на сервер посылается POST запрос, в котором указан ID статьи. Система после этого вычисляет смещение, которое будет прибавлено к контент-профилю пользователя.

2.3.2 Гиперпараметры модели

При обучении модели задаются следующие гиперпараметры.

- Алгоритм. Выбор между «softmax» и «negative sampling»
- Мерность векторов (от 50 до 1000).
- Контекстное окно. Определяет количество слов, которые будут учитываться при предсказании текущего слова (для Skip-Gram) и предсказываемых слов (для CBOW). Рекомендуется от 5 до 10.
- Количество эпох обучения.

Они напрямую влияют на эффективность модели.

Эти параметры также влияют и на скорость обучения. Кроме того, для векторизации с помощью Word2Vec и Doc2Vec очень важно количество документов. Малое количество текста даёт неудовлетворительные результаты даже с хорошо подобранными параметрами.

2.3.3 Зависимость времени обучения от параметров и размера выборки

Были проведены испытания модели (датасет – «text8»). Результаты представлены далее в рисунках 2.1-2.4 и таблицах 2.2-2.5.

Таблица 2.1 – Параметры компьютера

CPU	Intel Celeron 2957U 1.4 GHz (2 CPU)
RAM	8GB
GPU	Integrated (Intel HD)
Storage type	HDD

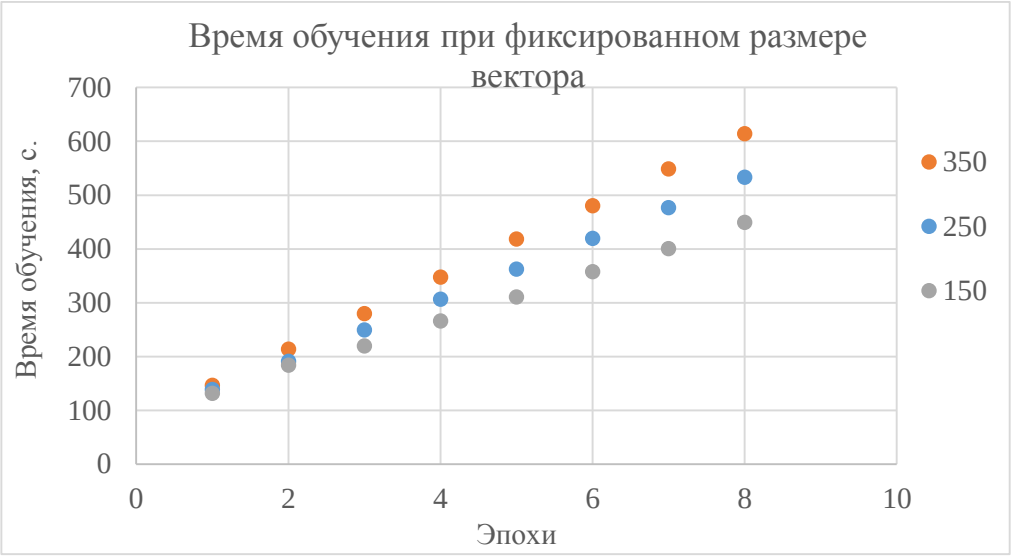


Рисунок 2.1 – График зависимости времени обучения от количества эпох

Таблица 2.2 – Время обучения модели при фиксированной размерности вектора (PV-DM).

Эпохи	Размер вектора		
	150	250	350
	Время, с.		
1	131.16	138.64	145.94
2	183.78	190.88	213.25
3	219.5	248.96	279.52
4	265.63	306.16	347.28
5	310.51	362.04	418.44
6	357.69	419.33	480.34
7	400.54	476.79	548.23

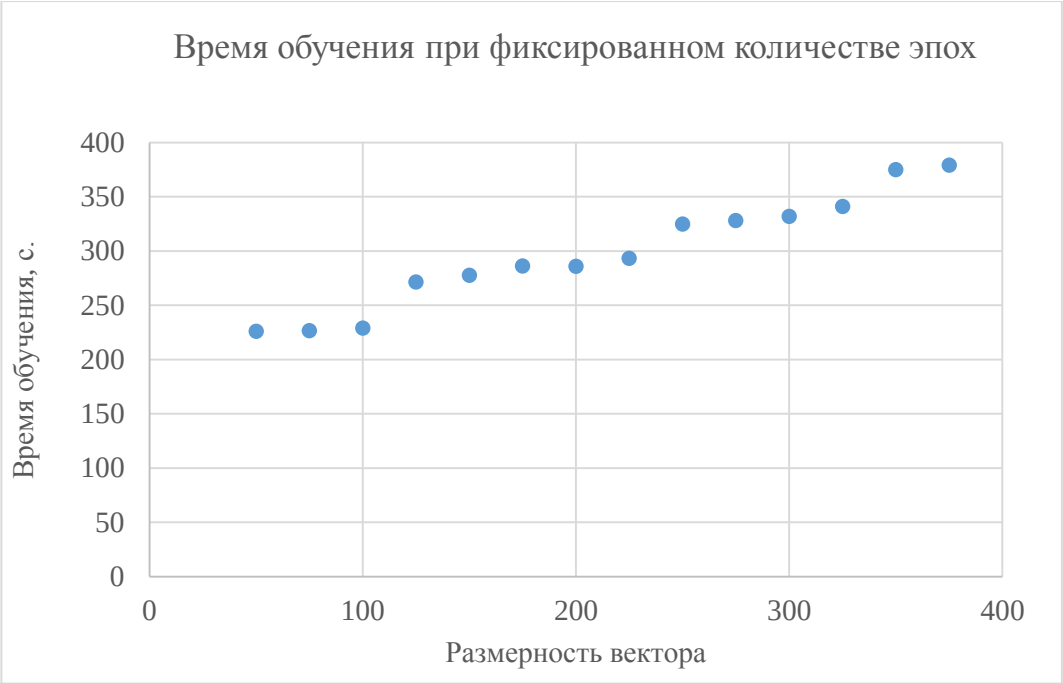


Рисунок 2.2 – График зависимости времени обучения от размерности вектора

Таблица 2.3 – Зависимость времени обучения от размерности вектора (PV-DM).

Размер вектора	Эпохи	Время, с.
50		226.17
75		226.9
100		229.06
125		271.63
150		277.93
175		286.31
200		285.91
225		293.32
250		325.13
275		328.22
300		331.96
325		341.19
350		375.16
375	4	379.34

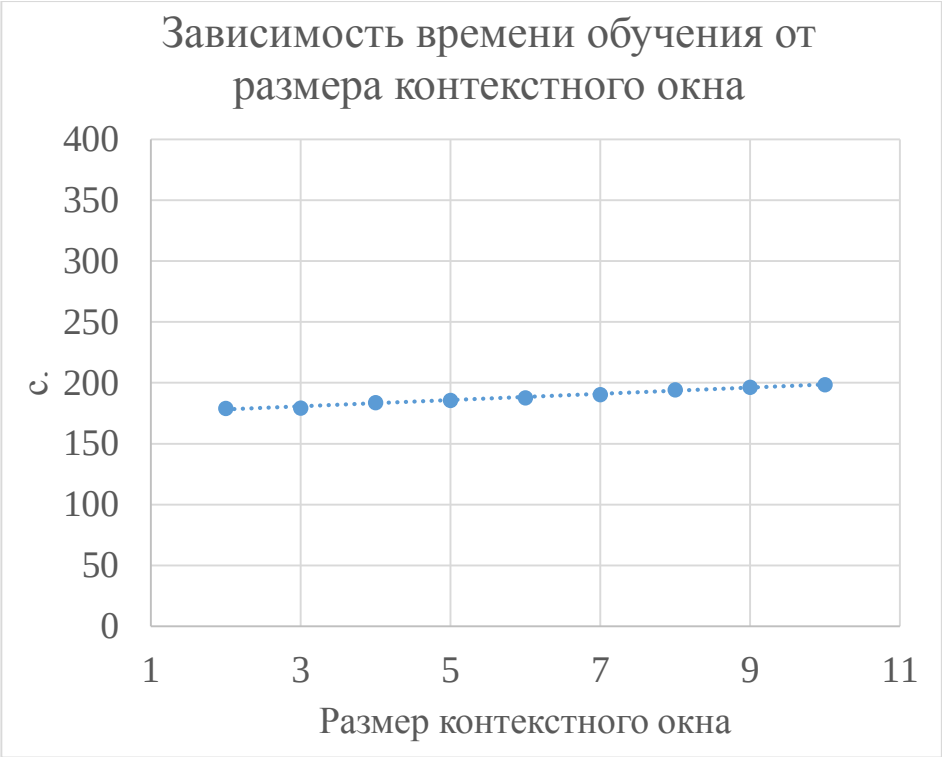


Рисунок 2.3 – График зависимости времени обучения от размера контекстного окна (логарифмический график).

Таблица 2.4 – Зависимость времени обучения от размера контекстного окна (PV-DM).

Размерность вектора	Эпохи	Время, с.	Размер контекстного окна
100	3	179.08	2
		179.46	3
		183.79	4
		185.61	5
		187.75	6
		190.37	7
		194.41	8
		196.53	9
		198.63	10

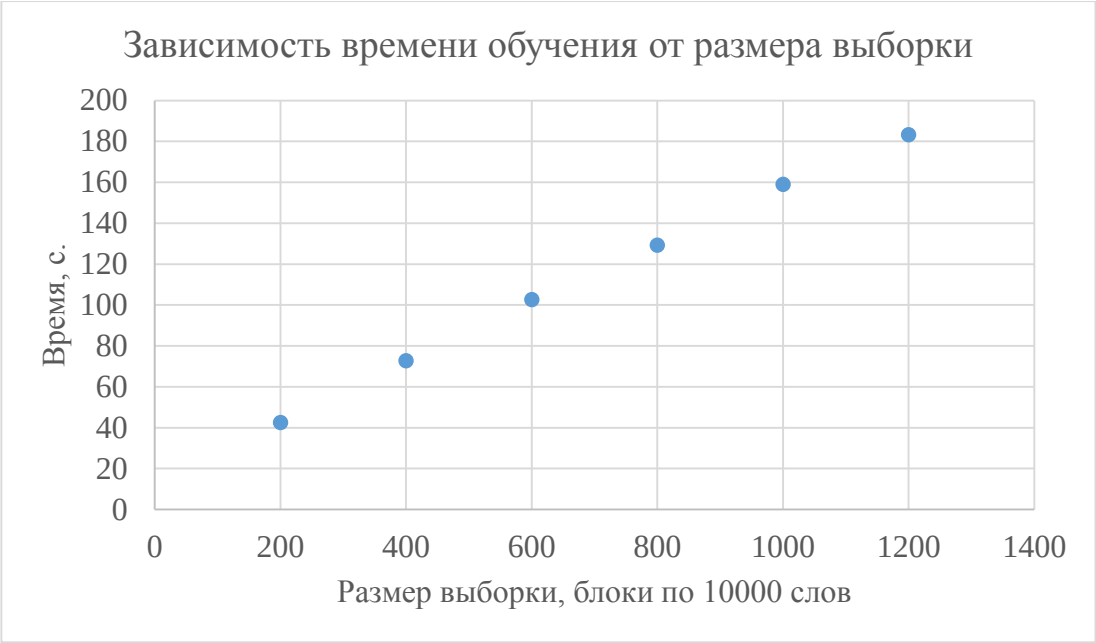


Рисунок 2.4 – График зависимости времени обучения от размера выборки

Таблица 2.5 – Зависимость времени обучения от размера выборки (PV-DM).

Размерность вектора	Эпохи	Время, с.	Размер выборки
100	3	42.56	200
		72.86	400
		102.73	600
		129.38	800
		159.07	1000
		183.33	1200

Код испытаний находится в Приложении А.

По испытаниям видно, что сильнее всего на время обучения влияют размер корпуса, а также размерность вектора. К плюсам алгоритма стоит отнести линейную зависимость скорости обучения от размера выборки. Это существенный критерий, ведь на обучение модели и так уходит много времени и ресурсов.

2.4 Особенности реализации рекомендательной системы

2.4.1 Концепция рекомендательной системы

В существующих работах, например, в [9], успешно применятся классификация с использованием предопределённых классов. Также используются тестовые данные для проверки. В данной рекомендательной системе, однако, использование обычных методов классификации затруднено.

Проблема заключается в разнородности информации. Классов (и их подклассов) в этой системе может быть огромное множество, которое, к тому же, будет разрастаться с поступлением новых статей. Следовательно, требуется другое решение.

Применим информацию об алгоритме Doc2Vec. Вместо разбиения векторного пространства на классы (что требует предопределённого тегированного набора данных), воспользуемся самим векторным пространством и его свойствами для имплицитной классификации.

В частности, в библиотеке Gensim [20], которая применялась при подготовке работы, присутствует следующая функция.

model.docvecs.most_similar([doc_vector])

Где переменная doc_vector — Paragraph ID текущего документа.

Функция возвратит список наиболее близких (в n-мерном пространстве) векторов документов.

Как было указано ранее, обученные Paragraph ID хранят семантический «слепок» статьи в математическом виде. Следовательно, выбирая наиболее близкие статьи в векторном пространстве, система получит документы, близкие по смыслу к входному документу.

Профиль пользователя точно также можно представить в виде n-мерного вектора. Инициализировать его можно несколькими способами:

– случайно;
– по результатам опросника или выбора интересных статей;
– после импорта закладок пользователя (и нахождения векторов существующих статей).

Первый способ в системе будет применяться для незарегистрированных пользователей, а второй — для зарегистрированных (опросник проходится после прохождения первого шага регистрации).

Полный код программы располагается в [21].

2.4.2 Диаграмма вариантов использования и глоссарий системы

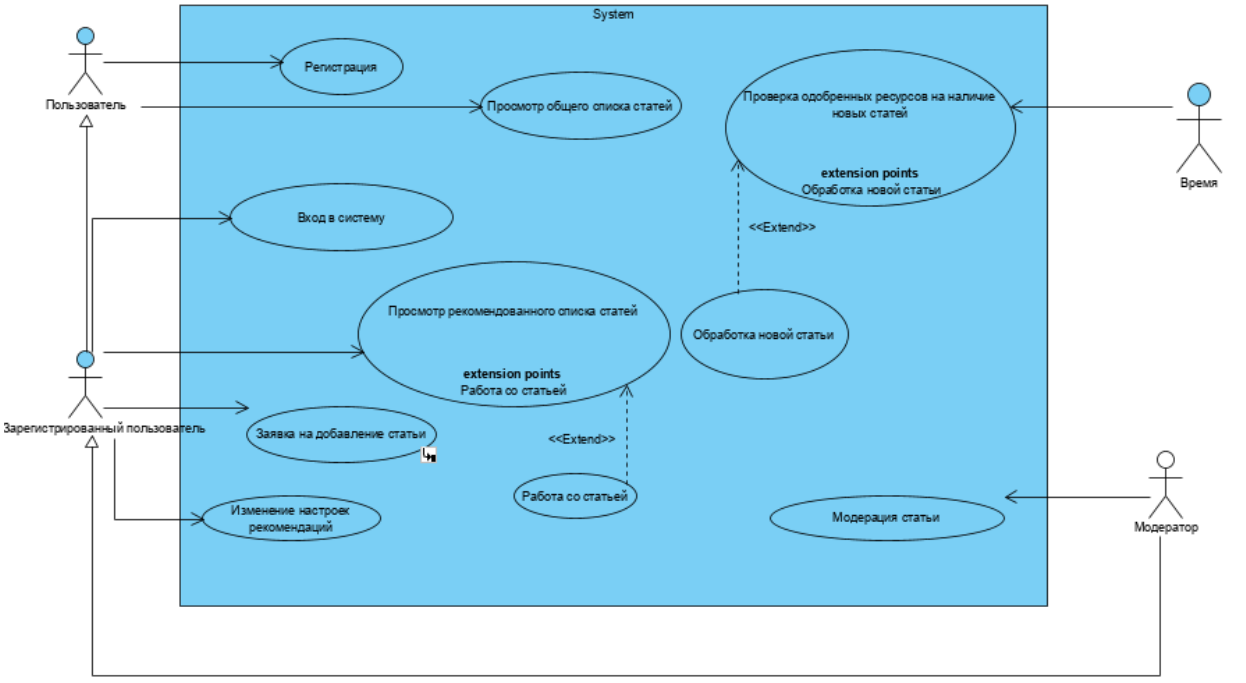


Рисунок 2.5 – Диаграмма вариантов использования

Таблица 2.6 – Глоссарий системы

Рекомендательная система (RecommendationSystem)	Генерирует список персонализированных статей для пользователя, основываясь на ранее просмотренных статьях.
Пользователь (User)	Пользователь системы. Может просматривать базовый список статей. Может зарегистрироваться, вследствие чего изменяет роль на «Зарегистрированного пользователя».
Зарегистрированный пользователь (RegisteredUser)	Пользователь системы. После входа в систему может просматривать персонализированный список статей и проводить дополнительные операции со статьями. Доступно изменение настроек.
Список статей (ArticleList)	Набор статей, подобранный индивидуально (для зарегистрированного пользователя) или базовый (для незарегистрированного пользователя).
Модератор (Moderator)	Пользователь системы. Одобряет внесение новых статьёй в базу рекомендательной системы.
Обработчик статей (ArticleProcessor)	Загружает статью с доверенного сайта. Обрабатывает (убирает пунктуацию, лишние слова и т.д.) и преобразует содержание статьи в утверждённый формат. Записывает статью в базу данных.
Ресурс (Resource)	Сайт с текстовыми материалами
Одобренный ресурс (ApprovedResource)	Сайт, на котором любой текстовый материал считается одобренным.

2.4.3 Стек технологий

При создании прототипа рекомендательной системы использовались следующие технологии.

Фронтенд: Bootstrap, HTML, CSS, JavaScript.

Бэкенд: Flask Framework, Python 3.

База данных: MongoDB (No-SQL).

2.4.4 Структура базы данных

Так как при разработке использовалась No-SQL база данных, схема БД отображена в объектной нотации.

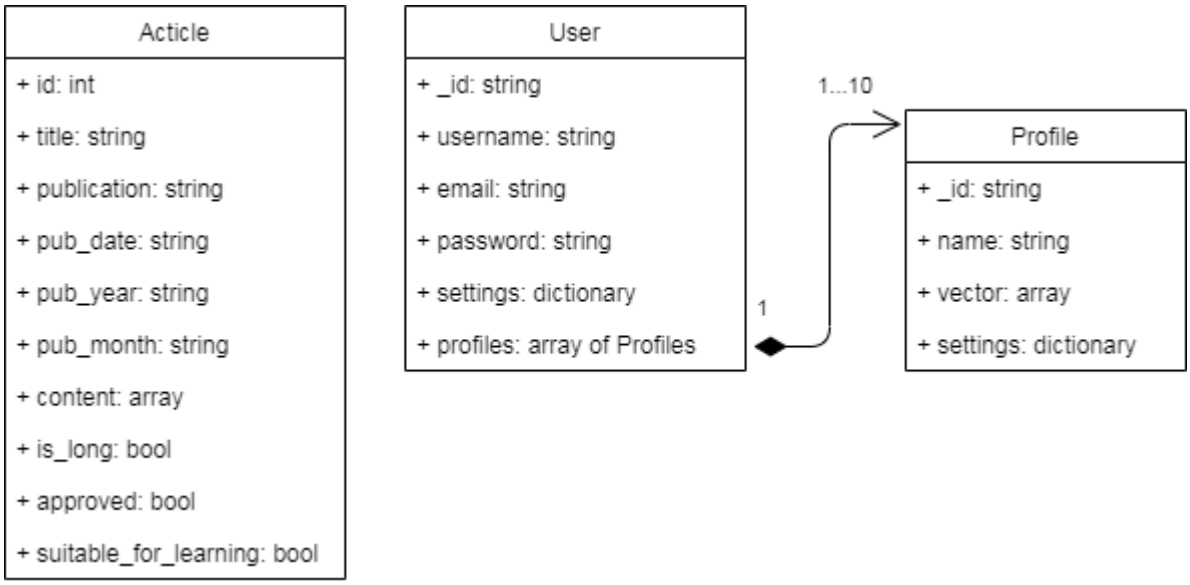


Рисунок 2.6 – Коллекции (аналог таблиц в терминологии MongoDB) базы данных.

В данный момент структура элемента коллекции Article соответствует структуре элементов датасета «All the news», которой использовался при разработке прототипа (за исключением служебных полей).

2.4.5 Реализация вариантов использования

2.4.5.1 Вариант использования «Регистрация»

При регистрации пользователь вводит свой логин, электронную почту и пароль (с подтверждением) в нужные поля ввода. Система тут же проверяет, зарегистрирован ли пользователь с таким логином или электронной почтой и, если это так, выдаёт сообщение об ошибке.

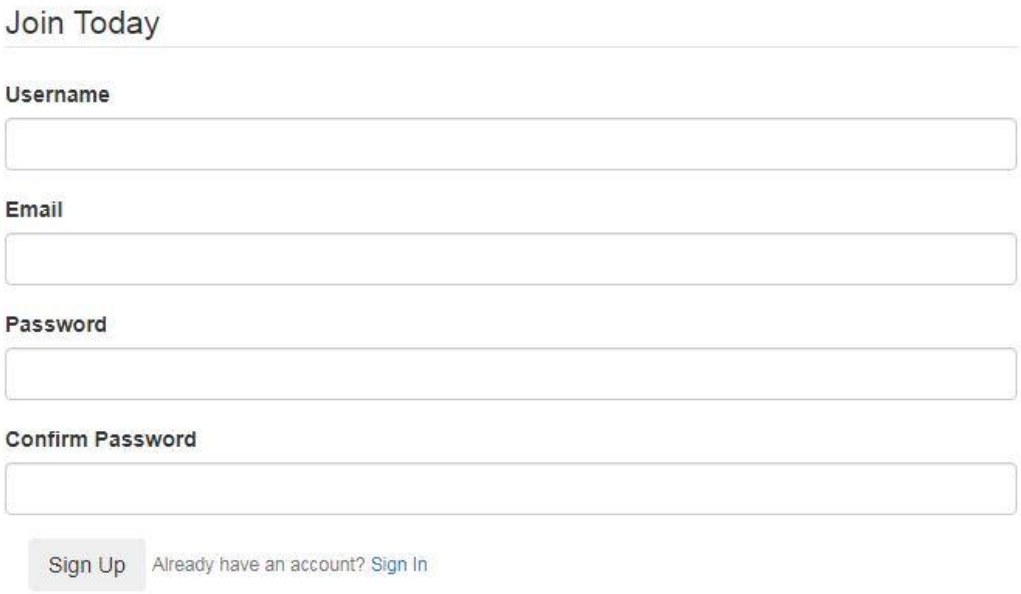


Рисунок 2.7 – Экран регистрации

2.4.5.2 Вариант использования «Вход в систему»

После регистрации пользователь может войти в систему.

В при первом входе система перенаправляет пользователя на страницу с опросником, на котором ему предлагается ряд статей. Пользователь отмечает наиболее релевантные статьи, а система корректирует контент-профиль пользователя (изначально он состоит из нулей). Следует отметить, что в этом варианте использования, как и в «Просмотре рекомендованных статей» к контент-профилю добавляется не полное значение вектора статьи, а некая вычисленная часть. Наиболее простой вариант — разделить все элементы вектора на некую константу C. В данной работе C = 50.

Рисунок 2.8 – Вход в систему

После прохождения опросника пользователь перенаправляется на страницу с персонализированными рекомендациями.

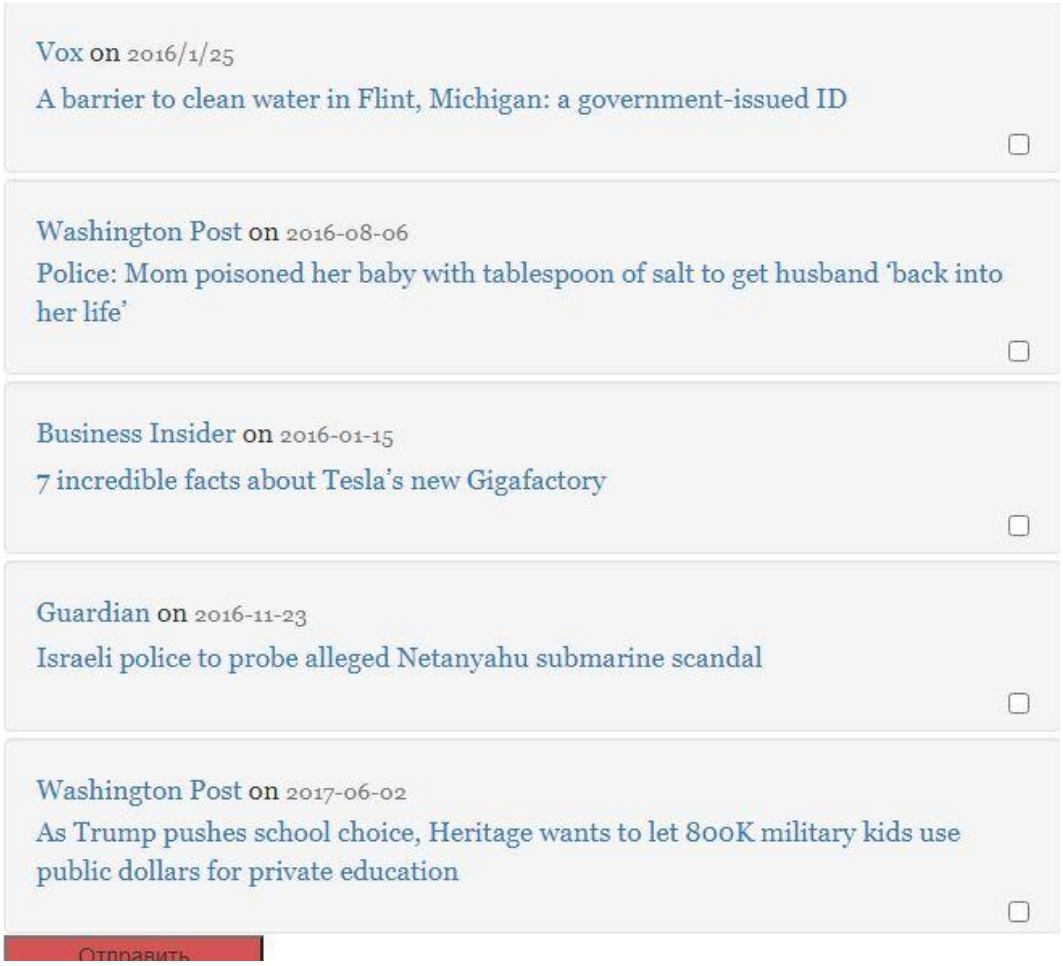


Рисунок 2.9 – Прохождение опросника

Код варианта использования можно найти в Приложении Б.

2.4.5.3 Вариант использования «Просмотр рекомендованных статей»

При переходе зарегистрированного пользователя на главную страницу сайта (также действителен роут «/recommendations») пользователю выдаются рекомендованные системой статьи.

Система извлекает контент-профиль пользователя, затем, с помощью упомянутой ранее функции most_similar, находит нужные статьи.

Следует упомянуть, что при разработке прототипа объёма датасета оказалось недостаточно для полного охвата. Дело в том, что по концепту системы в рекомендации должны попадать только статьи определённой длины (например, больше 9000 символов без пробелов). Кроме того, очень существенным является и наличие ссылки на оригинальную

статью. Как оказалось, далеко не все материалы в датасете содержат в поле «URL» ссылку. Поэтому для тестирования системы было решено временно убрать ограничения.

При переходе по ссылке на сервер отправляется POST запрос с ID статьи. Далее система корректирует контент-профиль пользователя — поэлементно прибавляет к нему элементы вектора статьи, уменьшенные в C раз.

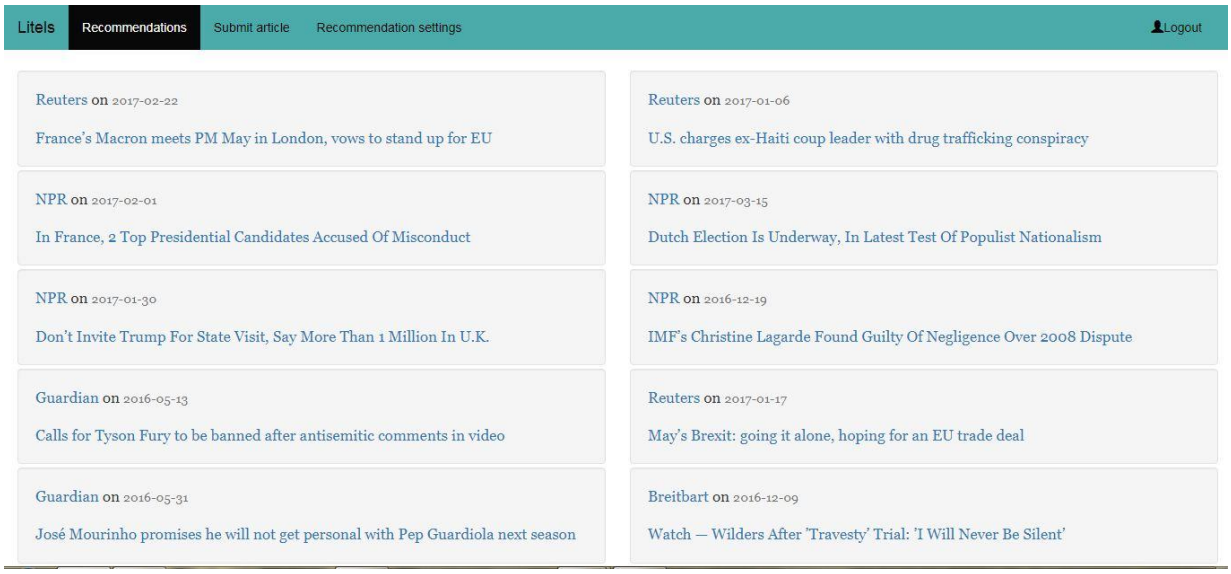


Рисунок 2.10 – Просмотр рекомендованных статей

Код варианта использования можно найти в Приложении Б.

2.4.5.4 Вариант использования «Заявка на добавление статьи»

В этом варианте использования пользователю предлагается заполнить несколько полей:

– URL статьи;
– наименование статьи;
– дата публикации статьи;
– текст статьи.

После заполнения и отправки статья проходит автоматическую проверку. Системе необходимо убедиться, что такого документа ещё нет в базе данных. Проверка производится по URL, однако в последующих версиях программы возможно реализовать более продвинутый вариант анализа по тексту (токенам) или даже по вектору статьи (полученная от пользователя статья подготавливается и очищается, после чего, с помощью функции

most_similar ищутся похожие статьи; слишком высокая степень соответствия будет означать наличие копии в БД).

Рисунок 2.11 – Отправка новой статьи

Если статья прошла автоматическую проверку, то она добавляется в отдельную коллекцию базы данных – ArticleQueue. Оттуда она в нужное время будет подгружена модератору на проверку.

2.4.5.5 Вариант использования «Модерация статьи»

Из очереди непроверенных статей модератору загружается некоторое количество элементов. Модератор проверяет статьи, одобряет или не одобряет их. При одобрении статья просто присоединяется к коллекции Articles. Отказ же можно обрабатывать, введя два дополнительных булевых поля в коллекцию Articles: «approved» и «suitable_for_learning». Первое поле означает то, одобрена статья или нет (можно ли рекомендовать её пользователю), второе — может ли система использовать статью для обучения модели (некоторые «забракованные» статьи могут быть написаны хорошим языком).

Рисунок 2.12 – Проверка статьи.

2.4.5.6 Вариант использования «Проверка одобренных ресурсов на наличие новых статей»

Периодическая задача на сервере время от времени (в данной работе установлено срабатывание раз в сутки) проверяет наличие новых статей на доверенных сайтах. Далее новые документы скачиваются и обрабатываются. После этого модель переобучается (переобучение затрагивает также статьи, одобренные модераторами).

2.5 Меры безопасности системы

Так как система подразумевает авторизованный вход пользователей, реализован ряд мер безопасности.

- При регистрации пароль хэшируется (с добавлением соли) и в открытом виде в базе данных не хранится (используется библиотека BCrypt).
- На всех формах отправки размещается токен для предотвращения атак типа CSRF.
- Некоторые URL доступны только после авторизации (проверка «сессионного куки»).

В будущем могут быть также внедрены дополнительные элементы защиты.

- JWT (Jason Web Token) вместо «сессионного куки».
- Применение HTTPS.

ЗАКЛЮЧЕНИЕ

В ходе данной работы была выяснена возможность создания рекомендательной системы на основе современных подходов NLP (векторизации слов). Были разобраны методы извлечения, преобразования и подготовки информации с помощью специализированных библиотек на Python (Newspaper — для извлечения информации, NLTK — для обработки информации, Gensim — для имплицитной классификации информации).

Проанализирована техника NLP Doc2Vec, которая позволяет после обучения на корпусе статей получить вектор в n-мерном пространстве для каждого документа, а после — производить математические операции с этими векторами (находить ближайшие по смыслу статьи и т.д.). Модели Doc2Vec были созданы для двух наборов данных. Длительность процесса обучения модели была оценена для различных гиперпараметров (размер выборки, размер контекстного окна и т.д.).

Был разработан и протестирован прототип рекомендательной системы (серверная часть – Flask; клиентская часть – HTML, CSS, JavaScript, Bootstrap; база данных – MongoDB). Рекомендательная система предоставляет пользователю наиболее релевантные статьи, основываясь на предыдущих действиях пользователя. При регистрации пользователю требуется пройти опросник, в котором он выбирает понравившиеся статьи из ряда предложенных системой. Система, в свою очередь, формирует начальный рекомендательный профиль пользователя. После начала полноценной работы, система корректирует вектор на основании действий пользователя (переход на статьи).

Основные результаты работы были доложены на 73-ей Всероссийской (с международным участием) научной конференции обучающихся и молодых учёных в 2021 году (доклад «Применение современных техник векторного представления слов в рекомендательных системах»).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 P. Melville. Recommender Systems / P. Melville and V. Sindhwani // www.ime.usp.br. — 2010. — Режим доступа: https://www.ime.usp.br/~jstern/miscellanea/seminario/Melville1.pdf. — (Дата обращения 12.04.21)

2 C. Zisopoulos. Content Based Recommendation Systems / C. Zisopoulos, S.Karagiannidis, S.Antaris // www.researchgate.net. — 2008. — Режим доступа: https://www.researchgate.net/publication/236895069_Content-Based_Recommendation_Systems. — (Дата обращения 12.04.21)

3 B. Zhao. Web Scraping / B. Zhao // Encyclopedia of Big Data. — 2017. — Режим доступа: https://www.researchgate.net/publication/317177787_Web_Scraping/ — (Дата обращения 12.04.21)

4 V. Krotov. Legality and Ethics of Web Scraping / V.Krotov, L.Silva // www.researchgate.net. — 2018. — Режим доступа: https://www.researchgate.net/publication/324907302_Legality_and_Ethics_of_Web_Scraping.— (Дата обращения 12.04.21)

5 Гражданский кодекс Российской Федерации (часть четвертая) от 18.12.2006 N 230-ФЗ (ред. от 30.04.2021) // Собрание законодательства РФ. - 25.12.2006. - № 52. - ст. 5496.

6 A. Nürnberger. A Brief Survey of Text Mining / A. Nürnberger, G. Paass. — 2005. — Режим доступа: https://www.researchgate.net/publication/215514577. — (Дата обращения 12.04.21)

7 M. Artama, Classification of official letters using TF-IDF method / M.Artama, I.N. Sukajaya, G.Indrawan// Journal of Physics Conference Series 1516 012001. — 2020. — Режим доступа: https://www.researchgate.net/publication/341936711_Classification_of_official_letters_using_TF-IDF_method. — (Дата обращения 12.04.21)

8 J. Beel. Research-paper recommender systems: a literature survey / P. Melville and V. Sindhwani // International Journal on Digital Libraries. — 2015. — Режим доступа: https://www.researchgate.net/publication/280576923_Research-paper_recommender_systems_A_literature_survey. — (Дата обращения 12.04.21)

9 A. Elsafty. Document-based Recommender System forJob Postings using Dense Representations / A. Elsafty, M. Riedl, C. Biemann // Language Technology, Universitat Hamburg, Germany. — 2014. — Режим доступа: https://www.inf.uni-

hamburg.de/en/inst/ab/lt/publications/2018-elsaftyetal-naacl-industry.pdf. — (Дата обращения 12.04.21)

10 T. Mikolov. Efficient Estimation of Word Representations on Vector Space / T.Mikolov, K. Chen, G. Corrado, J. Dean // arXiv.org. — 2013. — Режим доступа: https://arxiv.org/abs/1301.3781. — (Дата обращения 12.04.21)

11 T. Mikolov. Distributed Representations of Sentences and Documents / T.Mikolov, Q. Le // arXiv.org. — 2014. — Режим доступа: https://arxiv.org/abs/1405.4053. — (Дата обращения 12.04.21)

12 The Average Web Page (Data from Analyzing 8 Million Websites) [Электронный ресурс] // www.advancedwebranking.com. — 2021. — Режим доступа: https://www.advancedwebranking.com/html/. — (Дата обращения 30.05.21)

13 Newspaper3k: Article scraping & curation [Электронный ресурс]. — Режим доступа: https://newspaper.readthedocs.io/en/latest/. — (Дата обращения 30.05.21)

14 Прекратите скручивать (восклицательный знак) [Электронный ресурс]. // habr.com. — 2012. — Режим доступа: https://habr.com/ru/post/157947/. — (Дата обращения 30.05.21)

15 The world's rich need to cut their carbon footprint by a factor of 30 to slow climate change, U.N. warns [Электронный ресурс] / www.washingtonpost.com. — 2012. — Режим доступа: https://www.washingtonpost.com/climate-environment/2020/12/09/carbon-footprints-climate-change-rich-one-percent/. — (Дата обращения 30.05.21)

16 Common Crawl [Электронный ресурс] / commoncrawl.org. — Режим доступа: https://commoncrawl.org/. — (Дата обращения 30.05.21)

17 NLTK Library [Электронный ресурс] // nltk.org. — Режим доступа: https://www.nltk.org/— (Дата обращения 30.05.21)

18 text8 - word embedding [Электронный ресурс] // kaggle.com. — Режим доступа: https://www.kaggle.com/gupta24789/text8-word-embedding. — (Дата обращения 30.05.21)

19 All the news [Электронный ресурс] // kaggle.com. — 2017. — Режим доступа: https://www.kaggle.com/snapcrack/all-the-news — (Дата обращения 30.05.21)

20 Gensim: Topic modeling for humans [Электронный ресурс] // radimrehurek.com. — Режим доступа: https://radimrehurek.com/gensim/ — (Дата обращения 20.01.21)

21 Litels: Recommendation system for long articles [Электронный ресурс] — Режим доступа: https://github.com/Anzurna/litels. — (Дата обращения 10.06.21)

Приложение А

Листинг 1. Скрипт, который применялся для тестирования времени обучения модели.

```
import gensim
import gensim.downloader as api
import time
from gensim.test.utils import get_tmpfile
import nltk

#Создание тегированных документов для обучения
def create_tagged_document(list_of_list_of_words):
    for i, list_of_words in enumerate(list_of_list_of_words):
        yield gensim.models.doc2vec.TaggedDocument(list_of_words, [i])

#Функция, в которой происходит обучение и измерение скорости обучения
def elapse_gensim_time(t_data, vec_size=50, min=2, eps = 5, window=5):
    start = time.time()
    model = gensim.models.doc2vec.Doc2Vec(vector_size=vec_size, min_count=min, epochs=eps)
    model.build_vocab(t_data)
    model.train(t_data, total_examples=model.corpus_count, epochs=model.epochs)
    #Обучение модели
    end = time.time()
    return {"vec_size": vec_size, "epochs": eps, "time": end-start}

def main():
    dataset = api.load("text8") #Загрузка датасета

    data = [d for d in dataset]
    results = []
    train_data = list(create_tagged_document(data))
    #Зависимость времени обучения от размера контекстного окна
    for win_size in range(2, 11):
```

```
            results.append(elapse_gensim_time(train_data, vec_size=50, eps=5, window=
win_size))

            print(results)
            results = []
            #Зависимость времени обучения от количества эпох
            for ep in range(1, 9):

              results.append(elapse_gensim_time(train_data, vec_size=50, eps=ep, window=
win_size))

            print(results)
            results = []
            #Зависимость времени обучения от размерности вектора
            for vec_size in range(50, 400, 25):
              results.append(elapse_gensim_time(train_data, vec_size=vec_size, eps=ep,
          window= win_size))

            print(results)
```

Приложение Б

Листинг 2. Код серверной части приложения. Используется фреймворк Flask и база данных MongoDB.

```
#routes.py
from recommender import app, request, jsonify, db, bcrypt, model
from flask import render_template, url_for, flash, redirect
from recommender.forms import RegistrationForm, LoginForm, SimpleForm, SubmitArticleForm
from recommender.functions import (create_random_user_vector, get_articles,
    is_field_value_in_db, calc_user_vector_delta, get_article_by_id)
from recommender.models import User
from flask_login import login_user, current_user, logout_user
import numpy

setup_session_max_step = 5
setup_session_step = 0

user_vector = [0] * 300

#Осовной роут рекомендация для зарегистрированного пользователя
@app.route("/")
@app.route("/recommendations")
def main():
    if current_user.is_authenticated:
        doc_vector = numpy.asarray(current_user.get_doc_vector())
    else:
        doc_vector = create_random_user_vector()
    return render_template('main.html', articles=get_articles(doc_vector))

#Роут, по которому отправляется POST запрос при переходе на ту или иную статью (содержит ID статьи). Применяется для учёта переходов пользователя
@app.route("/redirect", methods=['GET', 'POST'])
def redirect_and_register_click():
    if request.method == "POST":
```

```
        if current_user.is_authenticated:
            print(request.json)
            article = get_article_by_id(request.json["id"])
            article_doc_vector = model.infer_vector(article["content"])
            shrunked_vector = [obj/50 for obj in article_doc_vector]
            user_vector = current_user.get_doc_vector()
            user_vector = [a_ + b_ for a_, b_ in zip(user_vector, shrunked_vector)]
            current_user.set_doc_vector(user_vector)


    resp = jsonify(success=True)
    return resp

#Регистрация в системе
@app.route("/register", methods=["GET", "POST"])
def register():
    if current_user.is_authenticated:
        return redirect(url_for("main"))
    form = RegistrationForm()
    if form.validate_on_submit():
        username = form.username.data
        email = form.email.data
        password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        find_user =  User.get_by_email(email)
        if find_user is None:
            User.register(username, email, password)
            flash(f'Account created for {form.username.data}!', 'success')
            return redirect(url_for('main'))
    return render_template("register.html", title="Register", form=form)
#Вход в систему (для зарегистрированного пользователя)
@app.route("/login", methods=["GET", "POST"])
def login():
    if current_user.is_authenticated:
        return redirect(url_for("main"))
    form = LoginForm()
```

```
    if form.validate_on_submit():
        email = form.email.data
        password = form.password.data
        find_user = db.users.find_one({"email": email})
        print("ID of find user", find_user["_id"])
        if User.login_valid(email, password):
            loguser = User(find_user["username"], find_user["email"],
                find_user["password"], _id=find_user["_id"], profiles=find_user["profiles"])
            print("testing", loguser._id)
            login_user(loguser, remember=form.remember.data)
            flash(f"{form.email.data} logged in succesfully", "success")
            return redirect(url_for('setup'))
        else:
            flash(f"You have not been loogged in, whoops!", "danger")
    return render_template("login.html", title="Login", form=form)

#Прохожнение опросника
@app.route("/setup", methods=["GET", "POST"])
def setup():
    global setup_session_step, setup_session_max_step, user_vector
    if not current_user.is_authenticated:
        return redirect(url_for("main"))
    if setup_session_step < setup_session_max_step:
        form = SimpleForm()
        random_doc_vector = create_random_user_vector()
        articles = get_articles(random_doc_vector)
        if form.validate_on_submit() and form.example.data:
            chosen_articles = form.example.data
            delta_vector = calc_user_vector_delta(articles, chosen_articles)
            user_vector = [a_ + b_ for a_, b_ in zip(user_vector, delta_vector)]
            print(user_vector)
            setup_session_step += 1
            if setup_session_step == setup_session_max_step:
                current_user.add_profile("Start", user_vector)
                return redirect(url_for("main"))
```

```
            else:
                return redirect(url_for("setup"))
        else:
            pass
            # return redirect(url_for("setup"))
    return render_template("setup.html", title="Setup", form=form, articles=articles)

@app.route("/submit_article", methods=["GET", "POST"])
def submit_article():
    form = SubmitArticleForm()
    if form.validate_on_submit():
        article = {"id": db.articles.count_documents({}), "title": form.title.data,
            "url": form.url.data , "content": form.content.data}
        db.moderation_queue.insert_one(article)

    return render_template("submit_article.html", title="Submit article", form=form)

@app.route("/logout")
def logout():
    logout_user()
    return redirect(url_for("main"))



#functions.py
from recommender import random, numpy, db, model

#Создание случайного пользовательского вектора (для незарегистрированных # посетителей
def create_random_user_vector():
    user_vector = []
    for i in range(0, 300):
        user_vector.append(random.uniform(-0.5, 0.5))

    user_vector = numpy.asarray(user_vector)
```

```
    return user_vector


similar_articles = []
#Проверка валидности статьи перед выачей пользователю
def is_article_meets_requirements(article):
    if article["url"] != "": #article["is_long"] and
        return True
    else:
        return False
#Получение набора статей
def get_articles(doc_vector, doc_range=10, min_amount=5):
    similar_articles = []
    # doc_vector = model.infer_vector(test_article["content"])

    most_similar_docs = model.docvecs.most_similar([doc_vector], topn = doc_range)
    while len(similar_articles) < min_amount:
        most_similar_docs = model.docvecs.most_similar([doc_vector], topn = doc_range)
        for i in range(len(most_similar_docs)):
            s_a = db.articles.find()[int(most_similar_docs[i][0])]
            similar_articles.append({"url": s_a["url"],
                "title": s_a["title"],
                "id": s_a["id"],
                "date": s_a["pub_date"],
                "publisher": s_a["publication"]
                })

        doc_range += 10
    return similar_articles
#Расчет изменения контент-вектора в зависимости от выбранных в опроснике статей
def calc_user_vector_delta(articles, chosen_articles):
    vector = [0] * 300
    for article_number in chosen_articles:
        article_id = articles[int(article_number)]["id"]
        article = get_article_by_id(article_id)
        article_doc_vector = model.infer_vector(article["content"])
```

```
        shrunked_vector = [obj/50 for obj in article_doc_vector]
        vector = [a_ + b_ for a_, b_ in zip(vector, shrunked_vector)]

    return vector
```