МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК
И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
к.т.н., доцент
М. С. Воробьева
24.06. 2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(магистерская диссертация)

РАЗРАБОТКА РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ ДЛЯ ВЫБОРА ЭЛЕКТРОННОГО РЕСУРСА ПО КРАТКОМУ ОПИСАНИЮ

02.04.03. Математическое обеспечение и администрирование информационных систем
Магистерская программа «Разработка, администрирование и защита вычислительных систем»

Выполнила работу
Студентка 2 курса
очной формы обучения
Никитина
Дарья
Викторовна

Руководитель работы
д.п.н., профессор
Захарова
Ирина
Гелиевна

Рецензент
Старший преподаватель
кафедры
информационной
безопасности
Паюсова
Татьяна
Игоревна

Тюмень 2019

Содержание

Введение

Во время серфинга в Интернете поиск информации стал необходимым действием в жизни человека - пользователи ищут музыку, картины, книги, которые удовлетворяют определенным критериям, но кроме результатов, удовлетворяющих заданным условиям, выдаются также и лишние результаты, которые порой доминируют. Конечно, получаемые результаты зависят также и от корректности составленного запроса, но, бесспорно, важную роль играет тип запрошенной информации.

Была проблема при поиске определенной книги - сложность появилась на этапе формирования корректного запроса, так как не было фамилии, имени, отчества автора, а также наименования книги, а большинство сайтов предоставляют поиск именно по этим параметрам, не беря в расчет описание содержания книги, поэтому даже с помощью ключевых слов книга не была найдена.

В связи с этим появляется необходимость в развитии механизмов поиска некоторых данных определенной предметной области.

Цель выпускной квалификационной работы – разработка программного продукта для поиска книг по неструктурированному запросу пользователя.

Задачи:

1) проанализировать подходы к построению рекомендательных систем;
2) извлечь данные для создания системы;
3) изучить алгоритмы для обработки текстов;
4) разработать и реализовать алгоритмы выработки рекомендаций;
5) спроектировать и разработать рекомендательную систему для выбора книг.

Глава 1. Алгоритмы и технологии разработки рекомендательных систем

1.1 Рекомендательные системы

Рекомендательные системы – это такие системы, в которых результаты прогнозируются на основе данных о пользователе, ранее выбранных товарах, поставленных оценках или других собранных характеристиках.

В истории вычислительной техники достаточно рано были рассмотрены возможности компьютеров к прогнозированию – сама идея была озвучена Negroponte N. в 1975 году [14] и Kay A. [15], но первые успешные практические реализации появились в таких компаниях как Firefly и Amazon.com. Кроме того, корни рекомендательных систем можно проследить в когнитивной науке [13], поиске информации [18], в науке управления [19]. Рекомендательные системы стали самостоятельной исследовательской областью в середине 1990-х годов, когда исследователи начали концентрироваться на проблемах рекомендательных систем, которые явно полагаются на структуру рейтингов. В своей наиболее распространенной формулировке проблема рекомендательных систем сводится к проблеме оценки предметов, которые пользователь не видел. Интуитивно понятно, что эта оценка обычно основывается на рейтингах, присвоенных этим пользователем другим элементам, и на некоторой другой информации. Как только будет известно, как сделать такую оценку для еще не оцененных товаров, то появится возможность порекомендовать пользователю наиболее близкие элементы по оценке.

В своей статье авторы Ekstrand M. D., Riedl J. T., Konstan J. A. [12] обсуждают широкий спектр для разработки рекомендательных систем. Первым примером применения для подбора рекомендаций был упомянут компьютерный библиотекарь или система Grundy, в которой рассмотрены примитивные способы группировки пользователей по «стереотипам», которые строились после сбора данных о предпочтениях в книгах пользователей, данная система была разработана

в 1979 году и являлась достаточно примитивной, но это был шаг к развитию – об этом пишет E. Rich [13]. Разработкой, которой не требовалось соединение с сетью Интернет, являлся файловый менеджер, который отправлял файлы в корзину в том случае, если пользователь неправильно набирал папку назначения [16].

Adomavicius G., Tuzhilin A. [17] описывают методы составления рекомендаций, которые обычно разделяются по следующим трем основным категориям: подходы, основанные на содержании, совместные (коллаборативная фильтрация) и гибридные системы (подробнее в п. 1.2). В документе также описываются различные ограничения данных методов рекомендаций и их возможные расширения, которые могут улучшить возможности рекомендаций и сделать рекомендательные системы применимыми к еще более широкому спектру приложений. Авторы статьи пишут о том, что системы рекомендаций стали важной областью исследований с момента появления первых работ по совместной фильтрации с середины 1990-х годов, и считают, что интерес к этой области все еще остается высоким, поскольку он представляет собой проблемную область исследований, и из-за обилия практических приложений, которые помогают пользователям справляться с информационной перегрузкой и предоставляют персональные рекомендации, а также интересующий их контент и услуги. Однако, авторы обращают внимание на то, что несмотря на все эти достижения, текущее поколение рекомендательных систем по-прежнему требует дальнейшего совершенствования, чтобы сделать методы рекомендаций более эффективными и применимыми к более широкому кругу реальных приложений.

Методы построения рекомендательных систем начали бурно развиваться после конкурса Netflix Prize, по мнению Ю. С. Нефедовой [20]. В 2006 году был объявлен конкурс, при котором компания Netflix предоставила участникам данные о 100 миллионах рейтингах, проставленными 480 тысячами пользователями по 18 тысячам фильмам в течение 6 лет. Задача состояла в улучшении результатов рекомендаций на 10%, в сравнении с собственным движком компании для построения рекомендаций. При выполнении задачи участник получил бы миллион

долларов. На выполнение условий участниками ушло три года, но задача была выполнена.

Правильный подход для прогнозирования показывает качество используемого пользователем сервиса, так как позволит предоставить близкие запросам пользователя ресурсы.

1.2 Категории рекомендательных систем

Рекомендательные системы подразделяются по двум типам анализируемых данных:

- Характеризуемая информация (Characteristic information). Информация о каких-либо отдельных элементах (ключевые слова, категории) и о пользователях системы (предпочтения, данные профиля и т. д.).
- Действия пользователя (User-item interaction). Это хранимые результаты действий пользователя: рейтинги, количества покупок, лайки.

Существует множество различных рекомендательных систем, но среди них выделяются базовые: контент-ориентированные, коллаборативной фильтрации, гибридные.

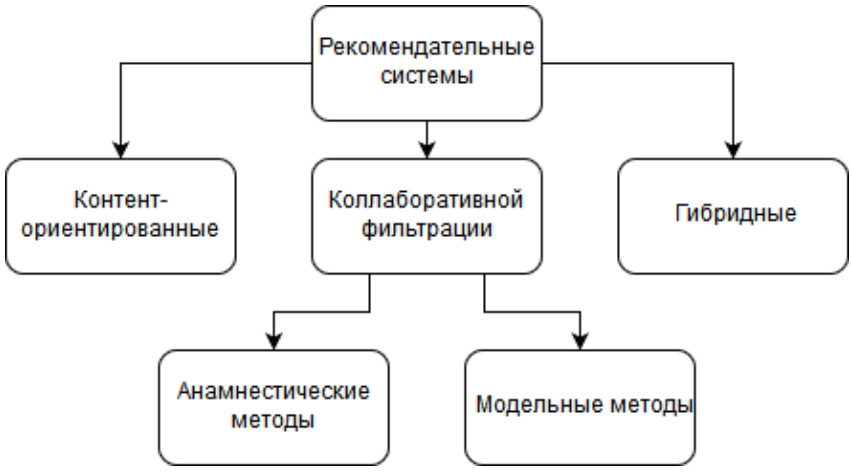

Рисунок 1. Базовые алгоритмы рекомендательных систем

1. Контент-ориентированные системы (Content-based) подразумевают алгоритмы, ориентирующиеся на характеристиках объектов, а также профилях пользователей системы. Предполагается, что если пользователь был заинтересован чем-то в прошлом, то он может интересоваться тем же в

будущем, поэтому данные об объектах, как правило, группируются по характеристикам (жанр, автор, год издания и т. д.). Профили пользователей в системах создаются с целью сбора данных о предпочтениях, также пользователям могут явно направить опрос, чтобы составить представление об их интересах [1].

2. Системы коллаборативной фильтрации (Collaborative filtering) в отличии от контент-ориентированных систем взаимодействуют накапливаемыми данными о действиях пользователей (проставленная оценка, выбранный продукт и т. д.). Такие системы можно делить на анамнестические и модельные методы [4]:
   - Анамнестические методы (Memory-based) – предполагается, что нет определенной модели данных, поэтому рекомендации составляются по ранее накопленным данным на основании определенной меры похожести.
   - Модельные методы (Model-based) – предполагается, что на первом шаге происходит формирование описательной модели предпочтений пользователей системы, объектов и взаимосвязей между ними, а на следующем шаге формируются рекомендации на основании полученной модели [3], то есть в отличии от анамнестического метода требуется предварительное обучение модели.
3. Гибридные системы (Hybrid) – это такие алгоритмы, которые объединяют в себе и контент-ориентированные, и алгоритмы коллаборативной фильтрации с целью избежать проблем, возникающих при работе только одного алгоритма и, как правило, рекомендательные системы относятся именно к гибридным [2].

Но даже имея знания о вышеописанных алгоритмах, нельзя сделать выбор в пользу определенного, так как результаты предсказать нельзя и прежде, чем выбрать один алгоритм, необходимо сравнить его с другими на тестовых данных.

Для того, чтобы разработать рекомендательную систему требуется решить, откуда будут поступать данные. В ряде случаев для интернет-магазинов данные собираются в базе данных по мере необходимости и предпочтений пользователей.

1.3 Методы извлечения данных для построения рекомендательной системы

Важным этапом работы является сбор данных, поэтому необходимо исследовать методы сбора данных с определенной веб-страницы.

На сегодняшний день существует множество вариантов извлечения данных. В ходе исследования были рассмотрены: Scrapy, BeautifulSoup.

1. BeautifulSoup – это библиотека Python для извлечения данных с HTML и XML файлов, предоставляя идиоматические способы навигации, поиска и изменения дерева синтаксического разбора веб-страниц [6], [8].
2. Scrapy – это прикладной фреймворк для сканирования веб-сайтов и извлечения структурированных данных. Он может быть использован для широкого спектра действий таких как анализ, обработка или архивирование данных.

   Также есть множество функций для извлечения данных, таких как:

   - Встроенная поддержка выбора и извлечения данных из источников HTML/XML с использованием селекторов CSS, выражения XPath, возможность использования регулярных выражений.
   - Интерактивная консоль (с поддержкой IPython) для проверки выражений CSS и XPath для получения данных.
   - Встроенная поддержка экспорта результатов в таких форматах как JSON, CSV, XML [5].

Для извлечения данных был выбран сайт mybook.ru, на котором представлены данные о современных и популярных книгах. Многие книги (не все) имеют подробное описание, что важно для точности прогнозирования, а также данный сайт не имеет защиты от ботов, что позволит избежать проблем при сборе данных.

Существует несколько вариантов для получения данных с защищенных веб-страниц, а именно:

1. Имитация работы браузера – достаточно трудоемкий метод, который подразумевает обратный инжиниринг (reverse engineering) для того, чтобы определить, как именно формируется запрос и в будущем использовать эти данные для сбора данных.
2. Автоматическое использование браузера. Этот вариант подразумевает запуск браузера и имитацию поведения живого пользователя. Наиболее популярным инструментом является Selenium WebDriver и с его помощью защиту сайта можно обойти. Selenium WebDriver – это программная библиотека для управления браузерами. WebDriver представляет собой драйвер для различных браузеров и клиентские библиотеки, предназначенные для управления этими драйверами [9]. Используя этот драйвер, можно эмулировать поведение реального пользователя и собрать данные.

   Главным минусом такого варианта является только то, что браузеры ресурсозатратнее в сравнении с предыдущим вариантом.
3. Использование браузера вручную – это способ, при котором требуется присутствие и взаимодействие пользователя с браузером, он должен или запускать скрипты, или выполнять определенные действия.
4. Написать браузер самостоятельно [7].

Основным минусом получения данных с сайта с защитой от ботов – это скорость получения данных. Кроме того, что данные будут долго загружаться, например, за счет открытия страницы в браузере, также потребовалось бы предусмотреть обращение к Интернет-ресурсу периодами, так как последовательная загрузка страницы могла бы вызвать нежелательное блокирование работы бота, поэтому в приоритете был выбран сайт без защиты от ботов.

1.4 Библиотеки и алгоритмы для предварительной обработки данных

После сбора данных требуется предварительная обработка – зачастую данные могут быть неточны, искажены, а также быть неполными, что будет отрицательно сказываться на точности результатов разрабатываемой рекомендательной системы. Качество рекомендательной системы напрямую зависит от точности выдаваемых пользователю результатов [10].

Предварительная обработка может содержать текстовую категоризацию, кластеризацию, стратификацию, концептуализацию, извлечение семантической сущности, семантический анализ, построение отношений между сущностями.

Существует проект с открытым исходным кодом – scikit-learn. Этот проект не прекращает развиваться и является наиболее популярным инструментом для анализа данных. В этот проект входят такие методы для предварительной обработки данных, как:

- StandardScaler, который приводит к тому, что все признаки будут иметь один масштаб за счет того, что каждый признак будет иметь среднее равное 0 и дисперсию равную 1.
- RobustScaler по аналогии с предыдущим методом приводит признаки к одинаковому масштабу, но масштаб вычисляться будет через медиану и квартили.
- MinMaxScaler сдвигает данные так, что признаки находятся в диапазоне от 0 до 1. Для двумерного набора данных это означает, что все данные помещаются в прямоугольник, образованный осью x с диапазоном значений от 0 и 1 и осью y с диапазоном значений от 0 и 1.
- Normalizer в отличии от остальных методов приводит к тому, что векторы признаков имеют евклидову длину равную 1, то есть проецируется точка данных на окружность радиусом 1. Этот метод применяют в том случае, когда важно направление вектора признаков, но не длина.

Так как в работе будет происходить анализ данных книг, то важным шагом является обработка именно текстов: убрать стоп-слова, удалить лишние морфемы, такие как окончание слова. Такую задачу позволит выполнить библиотека NLTK, которая является ведущей платформой для разработки программ на языке программирования Python для анализа естественного языка. Библиотека для работы с данными предоставляет интерфейсы корпоративных и лексических ресурсов, таких как WordNet с набором библиотек обработки текста для классификации, обработки по меткам, синтаксического и семантического анализа и многие другие возможности.

1. Под стоп-словами понимаются такие слова, которые не несут никакой смысловой нагрузки (например, а, в, но, и, или, не и так далее). То есть эти слова не несут смысловой нагрузки, но, если их не убрать, они будут влиять на разработку рекомендательной системы. Библиотека NLTK имеет встроенный список стоп-слов на 11 языках. Для использования возможностей библиотеки необходимо импортировать модуль stopwords и применить метод stopwords.words("russian").
2. Морфемы также необходимо удалить, иначе однокоренные слова будут восприниматься как разные и также будут отрицательно влиять на точность результатов. Для исключения морфем применяется стемминг (stemming), который содержится в модуле SnowballStemmer (nltk.stem) – его также требуется импортировать. Чтобы задать язык применяется – SnowballStemmer ("<Язык, на котором написан обрабатываемый текст>"), а для применения стемминга для одного слова использовать stem (Например, из слова «леса» после стемминга будет слово «лес») [21].

Следует отметить, что методы машинного обучения продолжают развиваться, и, соответственно, правильный выбор алгоритмов позволит достичь большей эффективности в прогнозировании. Поэтому целесообразно рассмотреть для каких задач можно применять методы машинного обучения.

1.5 Задачи машинного обучения

Машинное обучение относится к научной области, которая находится на пересечении статистики, искусственного интеллекта и компьютерных наук. Машинное обучение стремится к автоматизации процессов принятия решения за счет объединения известных примеров. Классификация задач машинного обучения представлена на рисунке 2.

Рисунок 2. Классификация задач машинного обучения

1. Обучение с учителем или контролируемое обучение (supervisor learning). Как следует из названия, существует оператор, который предоставляет алгоритму пары объект-ответ, а алгоритм выдает решение на основе каких-либо характеристик. Обучение с учителем подразделяется на такие задачи как классификация (classification) и регрессия (regression).
    - Классификация – прогнозирует метку класса (class label), которая является одним из ранее определённых вариантов списка допустимых значений. Классификация делится на бинарную (binary), что подразумевает работу в двух классах, и мультиклассовую (multiclass) – в этом случае работа происходит с большим числом классов.
    - Регрессия применяется, когда необходим прогноз непрерывного числа или числа с плавающей точкой (например, прогнозирование годового заработка в зависимости от таких характеристик как возраст, страна, опыт, образование).

Отличительной чертой регрессии от классификации является непрерывность (преемственность) – если полученные результаты непрерывно связаны, то решаемая задача является задачей регрессии.

2. Обучение без учителя или неконтролируемое обучение (unsupervisor algorithms) – обучение, когда работа происходит без обучающего оператора только на основе ранее полученных объектов. Обучение без учителя подразделяется на следующие виды: преобразование данных, кластеризация [11].
    - Неконтролируемые преобразования (unsupervised transformations) – алгоритмы, которые образуют новое представление данных для удобного способа прогнозирования.
    - Кластеризация (clustering algorithms) – алгоритм разделения данных на обособленные группы схожих по структуре элементов.

Для предсказания на больших объемах данных работать удобнее именно с типом «обучение без учителя», так как для записей от нескольких сот тысяч для оператора представляет трудность обучать модель, поэтому необходимо рассмотреть алгоритмы кластеризации.

Алгоритм K-Means

Одним из известных алгоритмов кластеризации является алгоритм k-средних или k-means, достоинством которого является простота использования и быстрая работа.

Данный алгоритм образует k кластеров, которые расположены на максимальных расстояниях друг от друга, то есть кластеры максимально отличны по некоторым характеристикам. Данное число может основываться на предыдущих исследованиях, теоретических данных или каких-то фактах.

Для первоначального распределения объектов по кластерам выбирается количество кластеров (k). Далее происходит вычисление «центров» кластеров (каждому кластеру соответствует один центр), такой выбор может происходить как

выбор первых или случайных k-наблюдений. Процесс вычисления центра происходит до тех пор, пока кластерные центры не стабилизируются (все наблюдения принадлежат кластеру, которому принадлежали до текущей итерации) или число итераций не станет равным максимальному числу итераций. Таким образом, каждый объект соответствует только одному кластеру [28].

К недостаткам данного алгоритма относится то, что он, во-первых, чувствителен к выбросам, которые могут искажать среднее (например, пустые данные), медленная работа на больших базах данных, а также требуется явно указать количество кластеров.

Алгоритм CLOPE

Алгоритм CLOPE (Clustering with sLOPE) предназначен для кластеризации категорийных данных, а именно множество транзакций, где под одной транзакцией подразумевается некоторый набор объектов. Как и в остальных алгоритмах кластеризации, CLOPE имеет критерий, который путем итеративного сканирования базы данных разделяет данные на кластеры.

Критерий может быть локальным или глобальным. Локальный критерий определяет сходство между парными объектами. Глобальный критерий эффективен для кластеризации больших категорийных наборов данных [29].

Достоинствами данного метода является высокая масштабируемость и скорость обработки, а также качество кластеризации, которая достигается путем применения глобального критерия оптимизации. При обработке алгоритм хранит в RAM определенную информацию по всем кластерам в незначительных объемах, поэтому задействует минимальное число сканирований набора данных. В отличии от предыдущего алгоритма, количество кластеров подбирается автоматически, но регулируется параметром, а именно коэффициентом отталкивания.

Самоорганизующиеся карты Кохонена

Самоорганизующиеся карты (SOM, Self Organizing Maps), были разработаны Т. Кохоненом и объединяют такие парадигмы анализа данных как кластеризацию и проецирование, а именно визуализацию многомерных данных на плоскости.

Данный алгоритм имеет два слоя: входной и выходной («экран»). SOM является алгоритмом без учителя и в ходе своего обучения производит анализ характера расположения точек входного слоя в m-мерном пространстве, на выходе организует топологический порядок и определенную степень регулярности исходных данных (метрическую близость векторов). Подгонка SOM заключается в итеративной настройке вектора весовых коэффициентов w_j каждого нейрона, j = 1, 2, …, p, для чего используется модифицированный алгоритм соревновательного обучения Хебба, который учитывает не только вклад нейрона-победителя, но и ближайших его соседей, расположенных в R-окрестности:

1. На стадии инициализации всем весовым коэффициентам присваиваются небольшие случайные значения w^0ij, i = 1, 2, ... m.
2. На выходы сети подаются последовательно в случайном порядке образы y объектов входного слоя и для каждого из них выбирается «нейрон-победитель» (BMU, Best Matching Unit) с минимальным расстоянием $\sum_{i=1}^{m}\left(y_i - w_{ij}^t\right)^2$
3. Определяется подмножество «ближайшего окружения» BMU, радиус которого R уменьшается с каждой итерацией t.
4. Пересчитываются веса w^t_j выделенных узлов с учетом их расстояний до нейрона-победителя и близости к вектору y.

Шаги 2-4 повторяются, пока выходные значения сети не стабилизируются с заданной точностью [30].

1.6Сферы применения рекомендательных систем

Рекомендательная система при правильном подходе позволит получать требуемые пользователю результаты не только при поиске книг, но и в других областях, например, в университете при выборе студентом близкого по его навыкам курса для обучения среди тысячи других вариантов.

На сайтах, посвященных кинематографу, играм, покупкам можно составить рекомендательную систему основываясь на предпочтениях, возрасте, времени года, местоположению пользователей и многих других признаках, которые могут на первый взгляд не казаться значимыми. Также можно разделять (сегментировать) клиентов на группы по схожим предпочтениям.

Рекомендательные системы на сегодняшний день используются во многих веб-сайтах: Интернет-магазины (Amazon), которые составляют список товаров после выбора товаров на основе выборов, сделанных другими пользователями, при составлении списков рекомендуемых фильмов (Netflix), сервис Yandex.Радио, а также рекомендательные системы используют Facebook, Twitter, Google, MySpace, Last.fm, Del.icio.us, Pandora, Goodreads и именно поэтому важно развивать данное направление, так как сфера применения становится шире с каждым днем.

Глава 2. Формирование базы данных и предварительная обработка

2.1Алгоритм формирования базы данных

Для того чтобы получить данные с выбранного Интернет-ресурса (сайта www.mybook.ru), требовалось изучить структуру сайта и разработать механизм получения информации о книгах.

После исследования разделов сайта было выяснено, что, для того чтобы получить информацию обо всех книгах, требуется получить общий список жанров. Основной список жанров располагается по адресу: https://mybook.ru/catalog/. Каждый из жанров имеет множество поджанров и, чтобы отслеживать работу приложения и при необходимости знать, с какого шага следует продолжить загрузку информации (например, если веб-сервис прервет свою работу в связи с проведением профилактических работ и тому подобными причинами), необходимо перед извлечением данных о книгах получить общий список всех жанров.

Когда список был получен, выяснилось, что не все наименования жанров являются уникальными, а именно жанров «Современная зарубежная литература» было два, но они имели разные адреса.

Чтобы получить все книги одного жанра требуется пройти по всем страницам каталога, также требуется учесть, что в случае, если книг меньше восемнадцати, то на странице отсутствует атрибут для постраничного перехода.

При получении информации об одной книге собиралась информация о наименовании, авторе и описании, также сервис сайта mybook.ru по некоторым книгам указывал темы и необходимо учесть то, что одна книга может относиться к нескольким жанрам, а также к нескольким темам, в связи с этим диаграмма базы данных представлена на рисунке 3.

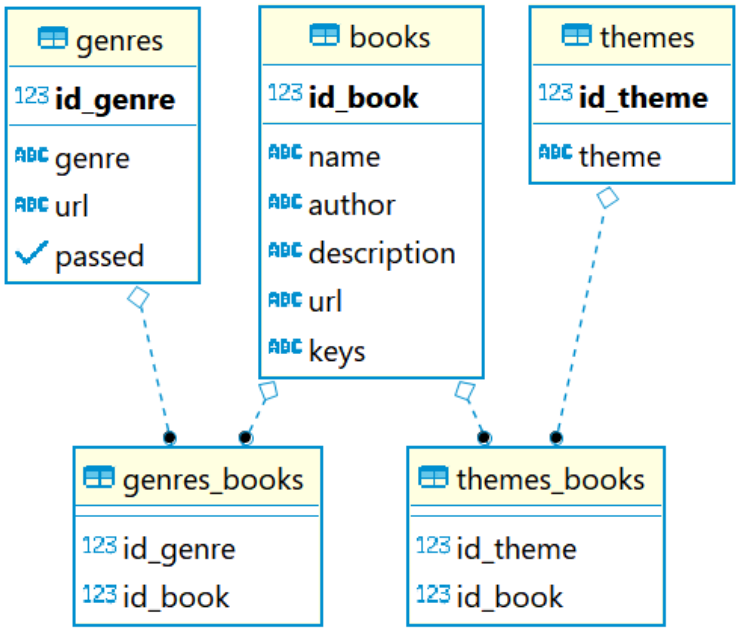

Рисунок 3. Схема базы данных

Таблица «genres» содержит поля: id_genre, genre, url, passed, где id_genre – идентификатор жанра, genre – наименование жанра, url – адрес страницы, на которой расположены книги жанра, passed – булевская переменная, значение которой показывает были ли все книги жанра извлечены.

Таблица «genres_books» содержит поля: id_genre, id_book, которые являются вторичными ключами.

Таблица «books» содержит поля: id_book, name, author, description, url, keys, где id_book – идентификатор книги, name – наименование книги, author – автор, description – описание книги, url – адрес страницы на сайте mybook.ru, на которой представлена перечисленная информация, keys – ключевые слова, которые формируются из описания книги.

Таблица «themes» содержит поля: id_theme, theme, где id_theme – идентификатор темы, theme – наименование темы.

Таблица «themes_books» содержит поля: id_theme, id_book, которые являются вторичными ключами.

Таблицы «themes_books» и «genres_books» позволяют реализовать связь многие ко многим.

Алгоритм формирования базы данных (блок схема представлена на рисунке 4):

1) Извлечь и записать в базу данных общий список всех жанров.
2) Обращаясь к базе из таблицы жанров получать список жанров, по которым данные о книгах еще не были получены.
3) Выбрать из списка жанр.
4) Пройти на страницу жанра и получить список книг с одной страницы.
5) Проверить была ли книга ранее добавлена, если нет, то добавить информацию о книге в базу.
6) Загрузить все книги выбранного жанра и отметить жанр как пройденный.
7) Вернуться к 3 пункту.

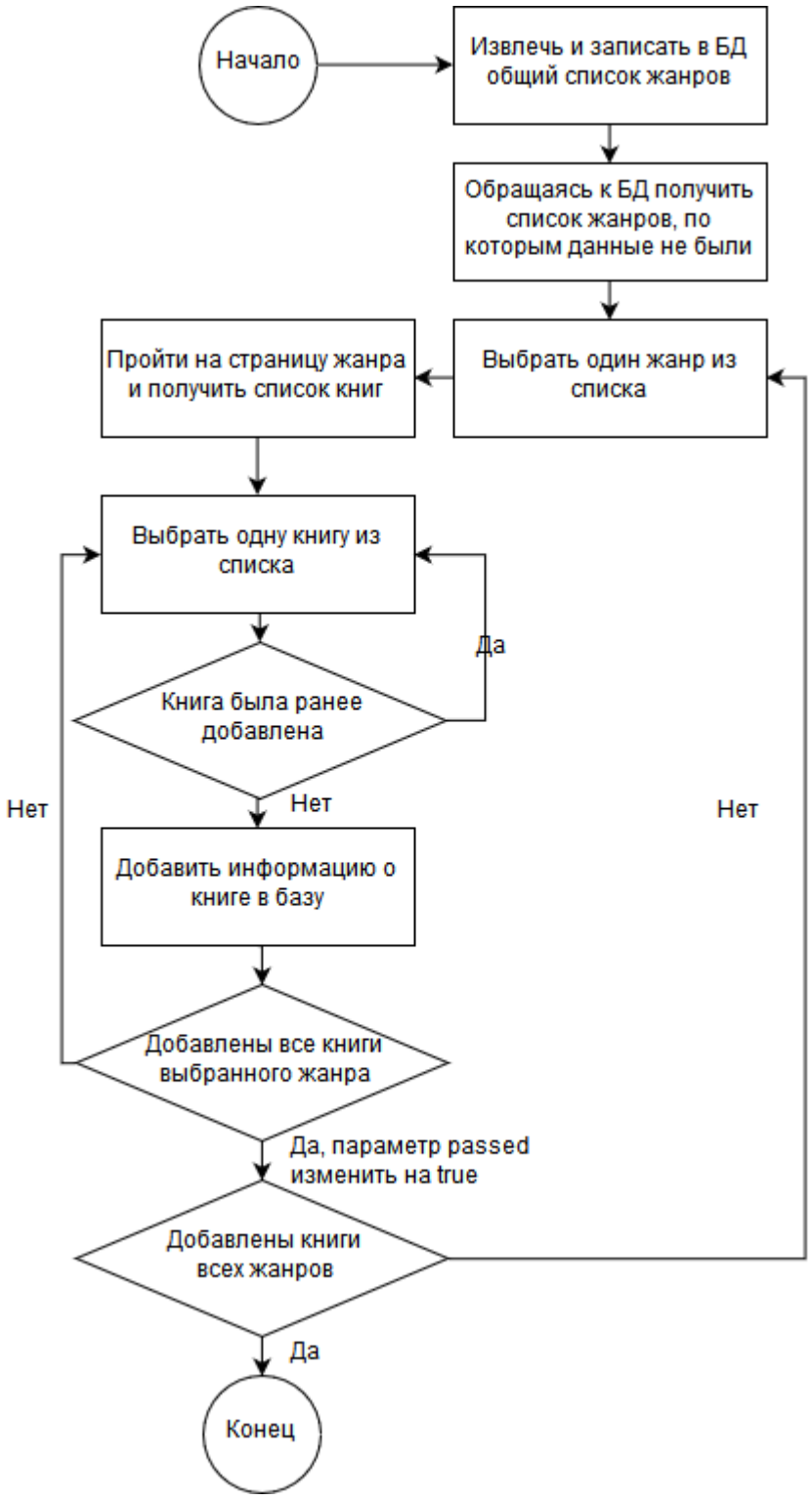

Рисунок 4. Блок-схема

2.2Выбор технологий

Приложение для скачивания информации с Интернет-ресурса написано на языке Python. Язык Python был выбран для разработки программного продукта, так как имеет множество открытых библиотек и модулей для обеспечения работы с

данными. Для написания программного кода приложения была выбрана среда разработки PyCharm [24]. PyCharm — это интегрированная среда разработки для языка программирования Python, она предоставляет средства для анализа кода, графический отладчик. С помощью библиотеки Python BeautifulSoup извлекалась информация с HTML файла (п. 1.3).

В качестве базы данных была выбрана PostgreSQL [25] — свободная объектно-реляционная система управления базами данных. Для реализации локальной базы данных выбрана система управления базами данных DBeaver [26] — платформенно-независимый клиент баз данных, написан на Java.

Для обеспечения взаимодействия между базой данных и объектами программирования была выбрана Peewee ORM (Object-relational mapping) [27].

2.3Извлечение данных

Для связи базы данных и приложения используются модели Peewee. Каждая модель соответствует таблице в базе данных, представленной на рисунке 3:

- модель `Books` – таблица books;
- модель `Genres` – таблица genres;
- модель `Genres_Books` - таблица genres_books;
- модель `Themes` – таблица themes;
- модель `Themes_Books` – таблица themes_books.

Настройки для подключения указаны в объекте `dbhandle`:

```
#Указание настроек подключения к базе
dbhandle = PostgresqlDatabase(
    'Books',
    user='postgres',
    password='',
    host='localhost')
```

`BaseModel` - базовый класс, который указывается для каждой модели для связи с базой данных:

```
#Базовый класс с данными о подключении к БД, эти данные должны быть указаны в каждой из моделей
class BaseModel(Model):
    class Meta:
        database = dbhandle
```

Модели описаны в коде:

```
#Класс для таблицы "books"
class Books(BaseModel):
    id_book = PrimaryKeyField(null=False)
    name = TextField()
    author = TextField()
    description = TextField()
    url = TextField()
    class Meta:
        db_table = "books"
        order_by = 'id_book'

#Класс для таблицы "genres"
class Genres(BaseModel):
    id_genre = PrimaryKeyField(null=False)
    genre = TextField()
    url = TextField()
    passed = BooleanField(default=False)
    class Meta:
        db_table = "genres"
        order_by = 'id_genre'

#Класс для таблицы "genres_books"
class Genres_Books(BaseModel):
    genre = ForeignKeyField(Genres,
                            db_column='id_genre',
                            on_delete='cascade',
                            on_update='cascade')
    book = ForeignKeyField(Books,
                           db_column='id_book',
                           on_delete='cascade',
                           on_update='cascade')
    class Meta:
        primary_key = False
        db_table = "genres_books"
        order_by = ('id_genre_book')

#Класс для таблицы "themes"
class Themes(BaseModel):
    id_theme = PrimaryKeyField(null=False)
    theme = TextField()
    class Meta:
        db_table = "themes"
        order_by = ('id_theme')

#Класс для таблицы "themes_books"
class Themes_Books(BaseModel):
    theme = ForeignKeyField(Themes,
                            db_column='id_theme',
                            on_delete='cascade',
                            on_update='cascade')
    book = ForeignKeyField(Books,
                           db_column='id_book',
                           on_delete='cascade',
                           on_update='cascade')
```

```
    class Meta:
        primary_key = False
        db_table = "themes_books"
        order_by = ('id_theme_book')
```

Последовательность действий, представленная в блок-схеме, описана ниже. Получение списка основных жанров реализуется функцией node_genres:

```
def node_genres(url):
    lst_genres = []
    lst_genres_name = []
    r = requests.get(url)
    data = r.text
    soup = BeautifulSoup(data, "lxml")
    genres = BeautifulSoup((soup.find('div', {"class":["billetContainer
CatalogAsideMenu__nicheListContainer"]})).prettify(),
'html.parser').find_all('a')

    point = 0
    for link in genres:
        str_=(urljoin(url, link.get('href')))
        if([str_] not in lst_genres):
            lst_genres.append(str_)
            lst_genres_name.append(link.find('span',
{"class":["link"]}).get_text().strip())
        point = point + 1

    d = {"lst_genres":lst_genres, "lst_genres_name": lst_genres_name}
    return pd.DataFrame(d)
```

После получения основного списка жанров происходит получение списка поджанров, далее работа происходит именно с этим списком:

```
#Получение поджанров
def all_genres(url):
    lst_genres = []
    lst_genres_name = []
    r  = requests.get(url)
    data = r.text
    soup = BeautifulSoup(data, "lxml")
    genres = BeautifulSoup((soup.find('div', {"class":["billetContainer
CatalogAsideMenu__nicheListContainer
NicheDetailView__asideMenu"]})).prettify(), 'html.parser').find_all('a')
    point = 0
    for link in genres:
        str_=(urljoin(url, link.get('href')))
        if([str_] not in lst_genres):

            lst_genres.append(str_)
            temp = link.find('span', {"class":["link"]})
            lst_genres_name.append(temp.find('span').get_text().strip())
        point = point + 1

    d = {"lst_genres":lst_genres, "lst_genres_name": lst_genres_name}
    return pd.DataFrame(d)
```

Добавление жанра реализуется с помощью ORM (обращение к таблице genres с помощью модели `Genres`):

```
#Добавление жанра в БД
def add_genre(genre_, url_):
    row = Genres(
            genre=genre_,
            url=url_)
    row.save()
```

Запись о новом жанре добавляется в базу данных с полем passed со значением false по умолчанию. После того как общий список всех жанров был полностью получен, извлекается из списка один элемент и работа по извлечению списка книг происходит для одного жанра.

Затем обрабатывается HTML файл, то есть извлекается список книг текущей страницы:

```
# Получение основного списка книг одной страницы
def base_all_books(soup):
    print('base_all_books_enter')
    temp_urls = []
    content = soup.find('div', {"class":
["BookList__bookList"]}).find('ul')
    list_Book = content.find_all('li', {"class": ["BookList__item
BookList__treeColsLayout jest-book-card-item"]})

    for element in list_Book:
        temp_urls.append(urljoin(url, element.find('a').get('href')))
    print('base_all_books_out')
    return temp_urls
```

После того как список книг для текущей страницы полностью получен, требуется перейти к следующей:

```
#переход на следующую страницу
def next_page(soup, url):
    print('next_page_enter')
    content = soup.find('div', {"class":
["ContextPagination__contextPagination GenreDetailView__pagination"]})
    url_next = content.find('a', {"class": ["NavigationButton__next jest-
context-pagination"]}).get('href')
    url = urljoin(url, url_next)
    print('next_page_out')
    return url
```

Получение данных об авторе, наименовании книги, кратком описании и url-адресе на сайте для одной книги описан в коде:

```
# Получение данных одной книги
def base_data(url):
```

```
    print('base_data_enter')
    try:
        book = Books.get(Books.url == url)
        print('Книга уже есть в базе')
        return
    except Books.DoesNotExist:
        print('Книга еще не заведена в базе')
    url_catalog = 'https://mybook.ru/catalog/'

    name = None
    author = None
    description = None

    r = requests.get(url)
    data = r.text
    soup = BeautifulSoup(data, "lxml")

    try:
        name = soup.find('h1', {"class":
["BookPageHeaderContent__coverTitle"]}).get_text().strip()
    except AttributeError:
        name = None

    try:
        # выделяем блок
        author = (soup.find('div', {"class": ["BookAuthor__coverAuthor"]}))
        # берем список имен
        bookAuthor__authorList = author.find('div', {"class":
["BookAuthor__authorList"]}).find_all('span')
        author = ''
        for one_author in bookAuthor__authorList:
            temp_author = one_author.find('a').get_text().strip()
            author = temp_author + ', ' + author
    except AttributeError:
        author = None

    block = soup.find('div', {"class": ["TextTruncate__text"]})
    if (block != None):
        description = block.text.strip()

    #добавление книги
    id_book = add_book(name, author, description, url).id_book
    block = soup.find('div', {"class":
["BookDetailAnnotation__bookGenresBlock"]})
    next_or_not = block.find_all('h3')

    point = 0
    if (next_or_not != None):
        block = block.find_all('ul', {"class":
["BookGenresThemes__itemList"]})
        for element in next_or_not:
            if (element.text == "Жанры"):
                genres = (block[point]).find_all('li', {"class":
["BookGenresThemes__listItem"]})
                if (genres != None):
                    for link in genres:
                        temp_link = link.find('a').get('href')
                        temp_link = urljoin(url_catalog, temp_link)
```

```
                        id_genre = get_id_genre(link, temp_link)
                        add_genre_book(id_genre, id_book)


            if (element.text == "Темы"):
                tegs = (block[point]).find_all('li', {"class":
["BookGenresThemes__listItem"]})
                if (tegs != None):
                    for link in tegs:
                        add_theme_book(link.text, id_book)
            point = point + 1
    print('base_data_out')
```

В результате было получено 173 тысячей записей о книгах, по 260 жанрам и 9 тысячам темам, среди указанного числа книг 2,5 тысяч не имеют автора и 6 тысяч не имеют описания.

Так как количество записей о книгах составило 173 тысячи, то для поиска книг удобнее работать с типом «обучение без учителя», так как обучение модели на данных указанного объема для оператора представляет трудность.

2.4Предварительная обработка текста

Предварительная обработка текста представляет собой приведение к нормализованному виду.

Нормализация текста

Как была указано в пункте 1.4, нормализация происходит при удалении морфем и стоп-слов (с помощью библиотеки NLTK). Стемминг русского языка сложнее, чем у других языков за счет того, что в русском языке сложная морфологическая изменяемость слов, что влечет ошибки при выделении определенной морфемы слова. Работа по удалению морфем происходит с помощью стеммера MyStem [22], разработанного для русского языка специалистом «Яндекс» Сегаловичом И. [23].

Также необходимо учесть то, что в текстах встречаются символы пунктуации – от них также следует отказаться. В данном случае удаление символов пунктуации происходит с помощью регулярных выражений при подключении библиотеки re:

```
def preprocess_text(text):
    # убрали цифры и символы пунктуации
    text = re.sub(r"([0-9])|[^a-яа-z]", " ", text.lower())
```

```
    #лементизировали, исключили пустые строки и убрали стоп-слова
    tokens = mystem.lemmatize(text)
    tokens = [token for token in tokens if token not in russian_stopwords
              and token.isalpha()]
    text = " ".join(tokens)
    return text
```

Пакет стоп-слов скачивается библиотекой NLTK:

```
nltk.download("stopwords")
```

Для русского языка стоп-слова устанавливаются командой:

```
russian_stopwords = stopwords.words("russian")
```

Функция предобработки текста `preprocess_text` применялась к описанию книги, которое хранится в поле «description» таблицы «books» в базе данных, далее результат обработки был записан в поле «keys».

Определим свойство, которое будет характеризовать объекты, в данном случае книги. Этим свойством является «keys», который хранит нормализованный текст из «description». Чтобы применить алгоритм кластеризации для текстовых данных, необходимо тексты представить в виде характеристических векторов с помощью библиотеки sklearn [31], а именно TfidfVectorizer.

Векторизация с помощью TfidfVectorizer

Библиотека sklearn позволяет сделать подсчет N-грамм слов или иначе последовательностей символов с помощью CountVectorizer. В этом случае векторизатор строит словарь индексов признаков, где каждое значение индекса слова соответствует его частоте по всему корпусу.

Но необходим другой метод векторизации, так как длина описания книг может разниться или отсутствовать, из-за чего количество словоупотреблений в длинных текстах будет значительно отличаться от значений в коротких текстах, даже если они будут относиться к одной теме. Чтобы длина текста не влияла на значение частоты, частота должна определяться как количество употреблений каждого слова в тексте на общее количество слов, именно этот подход использует TfidfVectorizer.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=0.1, max_df=0.8, decode_error='ignore')
```

Конструктор TfidfVectorizer имеет следующие параметры:

- input – парамаетр, который определяет тип обрабатываемой информации и может принимать значения ‘filename’, ‘file’, ‘content’.
- encoding – параметр, который явно указывает кодировку, по умолчанию используется utf-8;
- decode_error – параметр, который определяет действия при работе с символами другой кодировки и может принимать значения ‘strict’, ‘ignore’, ‘replace’;
- strip_accents – параметр, который определяет метод нормализации, параметр может иметь значения ‘ascii’, ‘unicode’, None, по умолчанию None;
- lowercase – параметр, который при значении True приводит все символы к нижнему регистру, по умолчанию значение True;
- stop_words – параметр, которые позволяет учесть стоп-слова;
- token_pattern – параметр, который принимает регулярное выражение, определяющее, что считать токеном.
- max_df - параметр, который определяет порог, выше которого термины будут игнорироваться и может принимать значения от 0 до 1;
- min_df – параметр, который определяет порог, ниже которого термины будут игнорироваться и может принимать значения от 0 до 1;

  Вычисление векторов текстов поля keys:

  ```
  vectorized = vectorizer.fit_transform(dataframe_books['keys']).
  ```

Глава 3. Проектирование и разработка рекомендательной системы

3.1Проектирование рекомендательной системы

Рекомендательная система строится на данных с сайта www.mybook.ru, которые будут получены скриптом, написанном на языке Python, и записаны в базу данных PostgreSQL. После того как данные будут предобработаны, а именно нормализованы и векторизованы, они будут разбиты на кластеры (см. архитектуру на рисунке 5).

Сервис API представляет собой REST-сервис, который позволит обрабатывать запросы от пользователя к серверу. Сервис написал на языке Python с помощью микрофреймворка Flask [34].

Для получения результатов поиска книги пользователь отправляет запрос, который предварительно нормализуется и векторизуется, затем отправляется с помощью сервиса API к серверу, на котором формируется результат.

Рисунок 5. Архитектура системы

Пополнение базы данных будет происходить каждую неделю при извлечении данных скриптом, который написан на языке Python, из раздела «Новинки» сайта

mybook.ru. При извлечении книг, которых нет в базе данных будут происходить следующие действия:

1. Данные о новых книгах добавляются в базу данных и нормализуются.

2. Данные базы данных проходят векторизацию (векторизация выполняется повторно для всех данных, так как частота определяется как количество употреблений каждого слова в тексте на общее количество слов).

3. Повторное формирование модели.

Указанные действия будут выполняться в поздние часы, чтобы не помешать работе пользователей.

3.2Применение алгоритма кластеризации k-Means

После нормализации и векторизации данных для формирования рекомендательной системы необходимо разбить данные на кластеры.

Для алгоритма кластеризации k-Means необходимо выбрать количество кластеров (см. п. 1.5) – значение параметра n_clusters в конструкторе KMeans.

```
#кластеризация по методу k means
# кластеризатор
model = KMeans(n_clusters=num_clusters,
               n_init=1,
               verbose=1,
               random_state=3)
```

Конструктор KMeans имеет следующие параметры:

- n_clusters – количество кластеров для формирования, а также количество центроидов для генерации;
- init – метод для инициализации, по умолчанию ‘k-means ++’;
- n_init – количество раз, когда алгоритм будет запускаться с разными значениями центроидов. Окончательные результаты будут наилучшим результатом последовательных прогонов n_init с точки зрения инерции;
- verbose – режим логирования (подробный вывод);
- max_iter – максимальное количество итераций для алгоритма k-средних за один прогон, по умолчанию значение равно 300;

- precompute_distances - предварительное вычисление расстояний (ускоряет работу, но использует больше памяти);
- random_state – определяет генерацию случайного числа для инициализации центроида;
- copy_x – параметр, влияющий на вычисление центроидов;
- n_jobs – количество задач, используемых для кластеризации;
- algorithm – определяет алгоритм для k-Means («auto», «full» или «elkan», по умолчанию применяется «auto»)

```
#обучение
clustered = model.fit(vectorized)
```

Метод fit формирует модель.

Для вывода результатов центроидов кластера используется следующая команда:

```
# центроиды кластеров
print("km.centers_=%s" % model.cluster_centers_)
```

3.3Поиск по неструктурированному запросу

При поиске книги по уже сформированной модели пользователю требуется направить запрос, который пройдет такую же предобработку, как и текст в базе данных (функция обработки текста представлена в п. 2.4).

```
custom_request = preprocess_text('Запрос пользователя')
```

Для поиска по краткому описанию была написана функция `get_list_books_description`

```
def get_list_books_description(custom_request):
    # векторизация и прогноз
    Y = vectorizer.transform([custom_request])
    # поиск наиболее близкого кластера
    prediction = model.predict(Y)

    similar_indices = (model.labels_ == prediction).nonzero()[0]
    similar = []

    for i in similar_indices:
        dist = sp.linalg.norm((Y - X[i]).toarray())
        author = str(dataframe_books['author'][i])
        if(len(author)>=2):
            author = author[:-2]
```

```
        name = str(dataframe_books['name'][i])

        url = dataframe_books['url'][i]
        similar.append((dist,'{"author": "'+author+'", "name": "'+name+'", "url":
"'+url+'"}'))

    similar = sorted(similar, key=itemgetter(0))
    print("Count similar: %i" % len(similar))

    i = 0
    result = '{"list":['
    #формируем json для вывода
    for el in similar:
        result = result + el[1] + ','
        if (i == 10):
            break;
        i = i + 1
    result = result[:-1] + ']}'
    return result
```

Для определения меры похожести вычисляется Евклидово расстояние между документами. Библиотека Scipy [35] предоставляет функцию `scipy.linalg.norm` для вычисления расстояния между векторами пользовательского запроса и всем корпусом документов.

При параметре n_clusters равному 260 были сформированы такие кластеры, что при поиске похожего по описанию ресурса, во-первых, определялся кластер, книги которого были близки по тематике с запрашиваемым пользователем описанием, во-вторых, если описание было составлено наиболее подробно, то искомый результат был в десятке наиболее вероятных вариантах, выдаваемых пользователю. При поиске книг с помощью данной функции было сделано пятнадцать запросов. Искомая книга в числе первых была девять раз, в остальных случаях книга попадала в число десяти выдаваемых пользователю. Данный результат подтвердил то, что алгоритм k-Means подходит для кластеризации полученных данных и организации поиска книг по краткому описанию.

3.4Поиск с помощью библиотеки psycopg2

Посредством библиотеки psycopg2 можно осуществить полнотекстовый поиск в PostgreSQL[25], для этого применяются такие функции как to_tsvector и plainto_tsquery. Функция to_tsvector переводит данные к нормализованному виду, то есть исключает стоп-слова, приводит к нижнему регистру и убирает окончания.

Функция plainto_tsquery, подобно функции to_tsvector обрабатывает текст, но после преобразования между словами указываются операторы & (И).

Полнотекстовый поиск будет применен для поиска по автору и названию книги. Данный подход, не применим для поиска по краткому описанию, так как при векторизации частота слов определяется только для одного документа, а не всего корпуса документов, что может привести к неточности поиска. Второй причиной является то, что инструменты PostgreSQL не позволяют сравнить документы, то есть вычислить меру похожести.

Для формирования списка книг при поиске по автору посредством возможностей PostgreSQL на языке Python была написана функция get_list_books_author, которой передается уже обработанный пользовательский текст. Оператор **@@** соотносит обработанные значение в поле author и запрос пользователя.

```
def get_list_books_author(custom_request):
    with postgresql.open('pq://postgres:@localhost:5432/Books') as db:
        similar = db.query("SELECT * FROM books WHERE to_tsvector(author) @@
plainto_tsquery(\'" + custom_request + "\') limit 10")
        result = '{"list":['
        # формирование списка книг в формате json для вывода
        for el in similar:
            author = str(el['author'])
            if (len(author) >= 2):
                author = author[:-2]

            name = el['name']

            url = el['url']
            one_book = '{"author": "' + author + '", "name": "' + name + '",
"url": "' + url +'"}'

            result = result + one_book + ','
        result = result[:-1] + ']}'
        return result
```

Для формирования списка при поиске по названию книги используется функция get_list_books_name:

```
def get_list_books_name(custom_request):
    with postgresql.open('pq://postgres:@localhost:5432/Books') as db:
        similar = db.query("SELECT * FROM books WHERE to_tsvector(name) @@
plainto_tsquery(\'" + custom_request + "\') limit 10")
        result = '{"list":['
        # формирование списка книг в формате json для вывода
        for el in similar:
            author = str(el['author'])
            if (len(author) >= 2):
                author = author[:-2]
```

```
            name = el['name']

            url = el['url']
            one_book = '{"author": "' + author + '", "name": "' + name + '",
"url": "' + url +'"}'

            result = result + one_book + ','
        result = result[:-1] + ']}'
        return result
```

3.5Клиентская часть системы

Для поиска книги необходимо поле для ввода, а также минимальные опции, как например, выбор, по какому типу проводить поиск – по автору, названию или описанию книги. Затем предложить пользователю выбрать тип поиска (один из трех вариантов), получить текст пользователя и запросить книги. Но результат может не устраивать пользователя – в этом случае потребуется вернуться к одному из предыдущих шагов: выбору типа поиска или вводу текста (алгоритм описан на рисунке 6).

Данный алгоритм взаимодействия с пользователем можно реализовать с помощью чат-бота. Бот — это программа, которая имитирует некоторые действия человека, например, выполняет однотипные действия.

Чат-бот был выбран с открытым API социальной сети «ВКонтакте» (далее ВК). Для получения доступа к управлению настроек чат-бота в ВК необходимо создать сообщество. В сообществе внести настройки для работы с чат-ботом и получить ключ доступа сообщества. Доступ к API ВК появляется при наличии ключа доступа, который позволяет работать с API от имени группы и отвечать пользователям группы на сообщения.

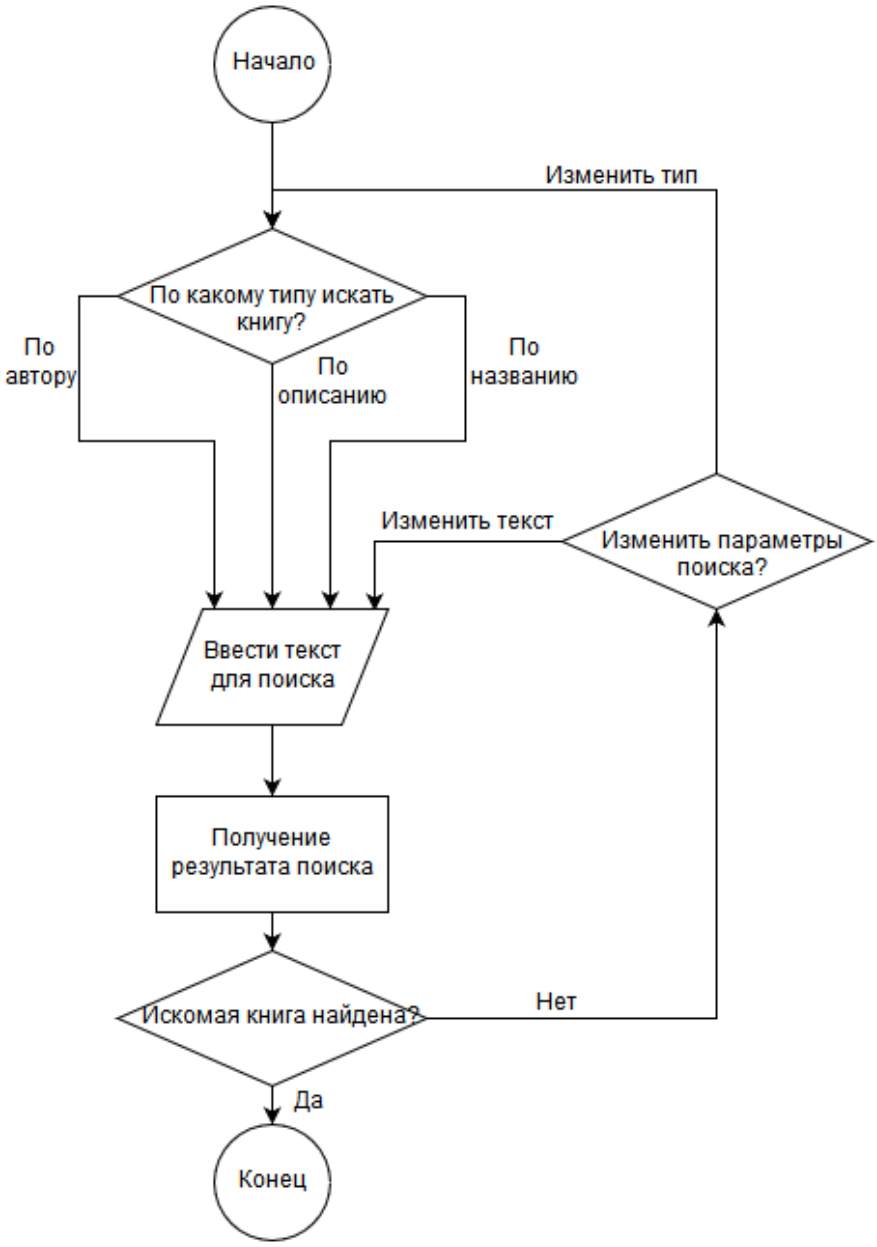

Рисунок 6. Алгоритм формирования запроса поиска

При работе с чат-ботом требуется:

- Отслеживать, на каком шаге находится пользователь (начало работы, выбор типа поиска, ввод текста для поиска, изменение запроса)
- Запоминать выбранный тип поиска – данная информация требуется в случае повторного ввода текста для поиска.

Для определения шага пользователя и типа выбранного поиска была создана таблица «users», содержит поля:

- id_user (идентификатор пользователя в социальной сети ВК);
- step (шаг, на котором пользователь находится);
- type (тип поиска).

В приложении А описана программная реализация алгоритма, указанного на рисунке 6.

Интерфейс клиентской части представлен в приложении Б.

3.6API сервис

Для формирования системы, а именно обеспечения взаимодействия клиентской части и серверной необходимо разработать API-сервис. Клиентская часть представляет собой чат-бота, который должен отправлять запрос, поэтому требуется простой инструмент для обеспечения взаимодействия между независимыми системами. Архитектурой, обеспечивающей такое взаимодействие в данной системе, является REST (Representational State Transfer) [32] при использовании веб-фреймворка Flask-Python. Flask – это микро-фреймворк разработанный для проектов Python, который позволяет разработчикам создавать веб-приложения [33].

Для того, чтобы при вызове запроса фреймворк Flask определил, какую функцию вызывать используется route. URL запроса будет иметь вид: http://<адрес сервера>/get_list_books

```
@app.route('/get_list_books', methods=['POST'])
def get_list_books():
    #получение данных, передаваемых в формате json
    data = request.get_json() or {}
    #в объекте с ключом type передается число, которое характеризует тип поиска
(по автору-1, названию-2 и описанию-3)
    type = data['type']
    #в объекте с ключом custom_request передается текст пользователя, по которому
происходит поиск
    #функция preprocess_text приводит к нормализованному виду текст пользователя
    custom_request = preprocess_text(data['custom_request'])
    #поиск по автору при type равному 1
    if (type == 1):
        result = get_list_books_author(custom_request)
    # поиск по названию при type равному 2
    elif (type == 2):
        result = get_list_books_name(custom_request)
    # поиск по описанию при type равному 3
    elif(type == 3):
```

```
        result = get_list_books_description(custom_request)
    parsed = json.loads(result)
    return json.dumps(parsed, indent=4, ensure_ascii=False)
```

Формат запроса для получения списка по тексту пользователя представлен в таблице 1.

Таблица 1. Формат запроса на получение списка книг

Имя метода	**Описание метода**	
get_list_books	Формирование списка книг по запросу пользователя (POST)	
Имя параметра	**Тип параметра**	**Описание**
Body		
type	Int	Тип поиска
custom_request	String	Текст, по которому происходит поиск

Пример запроса:

Адрес: http://{адрес-сервера}/get_list_books

```
{
      "type": 3,
      "custom_request": "глухонемой дворник спасает щенка, а тот лает"
}
```

Пример ответа:

```
{
    "list": [
        {
            "author": "Иван Тургенев",
            "name": "Муму",
            "url": "https://mybook.ru/author/ivan-sergeevich-turgenev/mumu-2/"
        },
        …
```

```
        {
            "author": "Холли Вебб",
            "name": "Щенок Тимми, или Я иду искать!",
            "url": "https://mybook.ru/author/holli-vebb/shenok-timmi-ili-ya-idu-iskat/"
        }
    ]
}
```

Заключение

В ходе работы были проанализированы базовые алгоритмы рекомендательных систем, извлечено 173 тысячи записей о книгах с сайта www.mybook.ru с помощью скрипта, написанного на языке Python и библиотеки BeautifulSoup. Изучены и применены алгоритмы для нормализации с помощью библиотеки NLTK и векторизации текстов с помощью векторизатора TfidfVectorizer. В качестве алгоритма кластеризации был выбран алгоритм K-Means. Для взаимодействия серверной части и клиентской был разработан API-сервис с помощью фреймворка Flask на языке Python. В качестве клиентской части был подключен чат-бот социальной сети ВКонтакте.

Спроектированная рекомендательная система дополняет полнотекстовый поиск сайта www.mybook.ru и предоставляет более гибкий поиск с помощью методов машинного обучения. Архитектурой системы предусмотрено периодическое пополнение базы данных.

Функционал рекомендательной системы позволяет в несколько простых действий с помощью чат-бота найти книги. Пользователь, следуя инструкции чат-бота, может выбрать тип поиска: по автору, названию или описанию, затем ввести текст для поиска, а при необходимости изменить запрос и обратиться к боту снова. Для поиска по типам «по автору» и «по названию» применяется метод полнотекстового поиска, для поиска по типу «по описанию» применяется метод машинного обучения.

Лучшие результаты при поиске по типу «по описанию» достигаются тогда, когда пользовательский текст имеет схожие слова с указанными в описании к книге. Преимуществом данной рекомендательной системы является то, что, если запрос пользователя был не точен или не имел слов, которые были указаны в описании книги, то система выдаст результат из десяти книг, в котором будут указаны те книги, которые похожи по тематике с искомой – искомая книга может оказаться в данном списке результатов.

С помощью API-сервиса данную систему в будущем можно расширить, подключив к веб-сайту www.mybook.ru, а также настроив взаимодействие с чат-ботами таких мессенджеров, как Телеграм, Whatsapp и Viber. На данный момент поиск организуется только для книг, но спроектированная система может быть незначительно реорганизована для поиска других ресурсов, например, фильмов, курсов и других ресурсов, которые имеют описание.

Список используемой литературы

1. Isinkaye F.O., Folajimi Y.O., Ojokoh B.A.: «Recommendation systems: Principles, methods and evaluation», 2015 г. - [Электронный ресурс]: https://www.sciencedirect.com/science/article/pii/S1110866515000341(дата обращения 01.02.2019)
2. Гастон В. «Introduction to Recommender Systems in 2018», 2018 г. - [Электронный ресурс]: www.tryolabs.com/blog/introduction-to-recommender-systems (дата обращения 01.02.2019)
3. Гончаров М. «Data mining: рекомендательные системы. Collaboration Filtering», 2012 г. (дата обращения 01.02.2019)
4. А. Константинов «Рекомендательные системы: тематический обзор» // Национальный Открытый Университет «ИНТУИТ», 2012- [Электронный ресурс]: www.hse.ru/data/2012/12/20/1303750691/Block-3.pdf (дата обращения 07.02.2019)
5. Scrapy, документация «Scrapy 1.5 documentation», 2018 г. - [Электронный ресурс]: www.doc.scrapy.org/en/latest/index.html (дата обращения 07.02.2019)
6. Beautiful Soap, документация «Beautiful Soap 4.4.0 documentation», 2015 г. - [Электронный ресурс]: www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата обращения 13.03.2019)
7. Козлов А. «Web scraping на Node.js и защита от ботов», 2016 - [Электронный ресурс]: www.habr.com/post/303726/ (дата обращения 13.03.2019)
8. Мансурова М. «Web Scraping с помощью python», 2016 г. - [Электронный ресурс]: www.habr.com/post/280238/ (дата обращения 13.03.2019)
9. Баранцев А. «Что такое Selenium?», 2012 г. - [Электронный ресурс]: www.habr.com/post/152653/ (дата обращения 13.03.2019)
10. Статья на портале от Microsoft «Задачи по подготовке данных для расширенного машинного обучения» 2017 г. - [Электронный ресурс]: https://docs.microsoft.com/ru-ru/azure/machine-learning/team-data-science-process/prepare-data (дата обращения 05.04.2019)
11. Мюллер А., Гвидо С. «Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными. O'Reilly Media, 2017. — 392 с.- [Электронный ресурс]: https://www.twirpx.com/file/2164153/ (дата обращения 05.04.2019)
12. Ekstrand M. D., Riedl J. T., Konstan J. A. Collaborative Filtering Recommender Systems //Human–Computer Interaction. – 2010. – Т. 4. – №. 2. – С. 81-173. - [Электронный ресурс]: https://www.nowpublishers.com/article/Details/HCI-009 (дата обращения 05.04.2019)

13. E. Rich, “User modeling via stereotypes,”Cognitive Science, vol. 3, no. 4,pp. 329–354, October 1979 - [Электронный ресурс]: https://onlinelibrary.wiley.com/doi/epdf/10.1207/s15516709cog0304_3 (дата обращения 05.04.2019)
14. Negroponte N. Soft architecture machines. – Cambridge, MA : MIT press, 1975. – С. 353-366. - [Электронный ресурс]: http://www.medientheorie.com/doc/Negroponte.pdf (дата обращения 05.04.2019)
15. Kay A. C. Computers, networks and education //Scientific American. – 1991. – Т. 265. – №. 3. – С. 138-149. - [Электронный ресурс]: http://vpri.org/pdf/sci_amer_article.pdf (дата обращения 05.04.2019)
16. Foner L. What’s an agent, anyway? a sociological case study. – Agents Memo 93, 1993. – Т. 1. - [Электронный ресурс]: http://www.student.nada.kth.se/kurser/kth/2D1381/JuliaHeavy.pdf (дата обращения 05.04.2019)
17. Adomavicius G., Tuzhilin A. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions - [Электронный ресурс]: https://homepages.dcc.ufmg.br/~nivio/cursos/ri13/sources/recommender-systems-survey-2005.pdf (дата обращения 05.04.2019)
18. Salton G. Automatic text processing: The transformation, analysis, and retrieval of //Reading: Addison-Wesley. – 1989. [Электронный ресурс]: http://www.iro.umontreal.ca/~nie/IFT6255/Introduction.pdf (дата обращения 10.04.2019)
19. Murthi B. P. S., Sarkar S. The role of the management sciences in research on personalization //Management Science. – 2003. – Т. 49. – №. 10. – С. 1344-1362. - [Электронный ресурс]: https://pdfs.semanticscholar.org/51da/0aeaa2ffee87235755f09566d75b3d802f1d.pdf (дата обращения 10.04.2019)
20. Нефедова Ю. С. Архитектура гибридной рекомендательной системы GEFEST (Generation–Expansion–Filtering–Sorting–Truncation) //Системы и средства информатики. – 2012. – Т. 22. – №. 2. – С. 176-196. - [Электронный ресурс]: http://www.mathnet.ru/links/b5afb75e85a85b57a0509788131d9701/ssi286.pdf (дата обращения 10.04.2019)
21. NLTK, документация «NLTK 3.4 documentation», 2018 г. - [Электронный ресурс]: https://www.nltk.org/api/nltk.stem.html (дата обращения 01.05.2019)
22. MyStem, документация «Использование MyStem», 2019 г. - [Электронный ресурс]: https://tech.yandex.ru/mystem/doc/index-docpage/ (дата обращения 01.05.2019)

23. Segalovich I. A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine //MLMTA. – 2003. – С. 273-280. (дата обращения 01.05.2019)
24. PyCharm 2019.1 Help, документация «Quick Start Guide»», 2019 г. - [Электронный ресурс]: https://www.jetbrains.com/help/pycharm/quick-start-guide.html#Quick_Start_Guide.xml (дата обращения 01.05.2019)
25. PostgreSQL, документация «Документация PostgreSQL и Postgres Pro», 2019 г. - [Электронный ресурс]: https://postgrespro.ru/docs (дата обращения 01.05.2019)
26. DBeaver, документация «Documentation», 2019 г. - [Электронный ресурс]: https://dbeaver.com/files/documentation.pdf (дата обращения 01.05.2019)
27. Peewee ORM, документация «peewee Documentation Release 3.9.5», 2019 г. - [Электронный ресурс]: https://buildmedia.readthedocs.org/media/pdf/peewee/latest/peewee.pdf (дата обращения 01.05.2019)
28. Fasulo D. An analysis of recent work on clustering algorithms. – Technical report, 1999. – №. 01-03. – С. 02. (дата обращения 04.05.2019)
29. Gan G., Ma C., Wu J. Data clustering: theory, algorithms, and applications. – Siam, 2007. – Т. 20. (дата обращения 04.05.2019)
30. Шитиков В. К., Мастицкий С. Э. Классификация, регрессия и другие алгоритмы Data Mining с использованием R //Режим доступу до ресурсу: http://ranalytics. blogspot. com/2012/03/f. html#. WnmcePll_IW. – 2017. (дата обращения 04.05.2019)
31. Bruning E. C. Streamed clustering of lightning mapping data in Python using sklearn //Scientific Computing With Python. – 2013. – Т. 2. (дата обращения 04.05.2019)
32. Fielding R. T., Taylor R. N. Architectural styles and the design of network-based software architectures. – Doctoral dissertation : University of California, Irvine, 2000. – Т. 7. (дата обращения 06.05.2019)
33. Alemu M. B. et al. REST API: Implementation with Flask-Python. – 2014. (дата обращения 06.05.2019)
34. Ronacher A. Flask documentation //Retrieved August. – 2017. – Т. 15. – С. 2018. (дата обращения 17.05.2019)
35. Van der Walt S. J., Vaught T., Millman J. The scipy documentation project (technical overview) //SciPy Conference–Pasadena, CA. – 2008. (дата обращения 01.06.2019)

Приложение А

Программная реализация клиентской части

```
#Модель ORM Peewee, которая соответствует таблице Users
#Класс для таблицы "books"
class Users(BaseModel):
    id_user = BigIntegerField()
    step=BigIntegerField(default=0)
    type=BigIntegerField(default=0)
    class Meta:
        primary_key = False
        db_table = "users"
        order_by = 'id_user'

#Базовый класс с данными о подключении к БД
class BaseModel(Model):
    class Meta:
        database = dbhandle


#Проверка на наличие пользователя, если его нет, то добавить в базу
def check_add_user(id):
    try:
        user = Users.get(Users.id_user == id)
    except Users.DoesNotExist:
        user = Users(
            id_user=id)
        user.save()
    finally:
        return user

#Приведение к коротким ссылкам
def getShortLink(long_url):
    json_string =
requests.get('https://api.vk.com/method/utils.getShortLink?url=' + long_url +
'&access_token=' + token + '&private=0&v=5.63')
    parsed_string = json.loads(json_string.text)
    return parsed_string['response']['short_url']

#Получение и формирование списка книг
def get_list_books(type, body):
    data = ({"type": type, "custom_request": body})
    json_string = requests.post(url, json=data)

    #Переменная parsed_string имеет тип данных словарь
    parsed_string = json.loads(json_string.text)

    index = 1#для нумерации элементов списка
    result = ""
    for one_book in parsed_string["list"]:
        result = result + str(index) + ". "

        flag = False
        author = one_book['author']
        if (len(author)>2):
            result = result+author
            flag = True

        name = one_book['name']
        if ((len(name) > 2) and flag):
            result = result + ' "' + name + '"'
```

```
        else:
            result = result + '"' + name + '"'

        url_book = getShortLink(one_book['url'])
        result = result + " (" + url_book + ")\n"
        index = index + 1
    return result

while True:
    random_id = 0

    #бесконечный цикл
    messages = vk.method("messages.getConversations", {"offset":0,
"count":20, "filter":"unread"})
    #если есть непрочитанное письмо
    if(messages["count"]>=1):
        #Получение идентификатора пользователя
        id = messages["items"][0]["last_message"]["from_id"]
        user = check_add_user(id)
        step = user.step
        body = (messages["items"][0]["last_message"]["text"]).lower()
        print(body)
        if (step == 0):
            vk.method("messages.send", {"peer_id":id,
                                        "message":"&#8252;Здравствуйте&#8252;
Я чат-бот сообщества 'Книги' и я могу помочь Вам в поиске книг
&#128270;\nПросто напишите мне и я расскажу, как это сделать.",
                                        "random_id":random_id})
            step = step + 1#1
            query = Users.update(step=step).where(Users.id_user == id)
            query.execute()
        elif (step == 1):
            vk.method("messages.send", {"peer_id": id,
                                        "message": "Для поиска книги выберите
тип поиска:\n&#128309;по автору, нажмите - 1\n&#128310;по названию, нажмите -
2\n&#128311;по описанию, нажмите - 3",
                                        "random_id": random_id})
            step = step + 1#2
            query = Users.update(step = step).where(Users.id_user == id)
            query.execute()
        elif (step == 2):
            if(body.isnumeric()):
                type = int(body)
                if(type == 1):
                    vk.method("messages.send", {"peer_id": id,
                                                "message": "Вы выбрали поиск
по автору. \n&#9997;Пожалуйста, введите текст для поиска книги",
                                                "random_id": random_id})
                    step = step + 1#3
                    query = Users.update(step=step,
type=1).where(Users.id_user == id)
                    query.execute()
                elif (type == 2):
                    vk.method("messages.send", {"peer_id": id,
                                                "message": "Вы выбрали поиск
по названию. \n&#9997;Пожалуйста, введите текст для поиска книги",
                                                "random_id": random_id})
                    step = step + 1#3
                    query = Users.update(step=step,
type=2).where(Users.id_user == id)
                    query.execute()
                elif (type == 3):
                    vk.method("messages.send", {"peer_id": id,
```

```
                                                "message": "Вы выбрали поиск по
описанию. \n&#9997;Пожалуйста, введите текст для поиска книги",
                                                "random_id": random_id})
                    step = step + 1#3
                    query = Users.update(step=step,
type=3).where(Users.id_user == id)
                    query.execute()
            else:
                vk.method("messages.send", {"peer_id": id,
                                                "message": "Такого типа поиска
нет &#128533;\nПожалуйста, введите тип поиска еще раз:\n&#128309;по автору,
нажмите - 1\n&#128310;по названию, нажмите - 2\n&#128311;по описанию, нажмите
- 3",
                                                "random_id": random_id})
                step = 2#2
                query = Users.update(step=step).where(Users.id_user == id)
                query.execute()
        elif (step == 3):
            #Запрос на получение списка по body
            type = user.type
            list_books = get_list_books(type, body)

            vk.method("messages.send", {"peer_id": id,
                                            "message": "Возможно, Вы искали
&#128218;:\n" + list_books + "\n&#10067;Вы нашли книгу, которую искали? Если
книга найдена, введите - 'Да'. Если книги нет, введите - 'Нет'",
                                            "random_id": random_id})
            step = step + 1  # 4
            query = Users.update(step=step).where(Users.id_user == id)
            query.execute()
        elif (step == 4):
            if(body == "да"):
                vk.method("messages.send", {"peer_id": id,
                                                "message":
"&#127881;&#127881;&#127881;Мы рады, что смогли помочь! \nЕсли Вам снова
потребуется помощь, напишите.",
                                                "random_id": random_id})
                step = 0
            elif(body == "нет"):
                step = step + 1#5
                vk.method("messages.send", {"peer_id": id,
                                                "message": "Если в списке нет
искомой книги, Вы можете:\n&#128312;изменить тип поиска, нажмите -
1\n&#128313;изменить запрос для поиска, нажмите - 2",
                                                "random_id": random_id})
            else:
                vk.method("messages.send", {"peer_id": id,
                                                "message": "К сожалению,
обращение не было распознано&#128532; \nПожалуйста, попробуйте еще раз.",
                                                "random_id": random_id})
                step = 1#возвращаем к диалогу
            query = Users.update(step=step).where(Users.id_user == id)
            query.execute()
        #Возникла проблема - необходимо вернуться к предыдущим шагам
        elif (step == 5):
            if (body.isnumeric()):
                i = body.isnumeric()
                if(i == 1):
                    vk.method("messages.send", {"peer_id": id,
                                                    "message": "Для поиска книги
выберите тип поиска:\n&#128309;по автору, нажмите - 1\n&#128310;по названию,
нажмите - 2\n&#128311;по описанию, нажмите - 3",
                                                    "random_id": random_id})
```

```
                    step = 2
                elif(i==2):
                    vk.method("messages.send", {"peer_id": id,
                                                "message": "Пожалуйста,
введите текст для поиска книги &#9997;",
                                                "random_id": random_id})
                    step = 3
                else:
                    vk.method("messages.send", {"peer_id": id,
                                                "message": "К сожалению,
обращение не было распознано&#128532; \nПожалуйста, попробуйте еще раз.",
                                                "random_id": random_id})
                    step = 1  # возвращаем к диалогу
                query = Users.update(step=step).where(Users.id_user == id)
                query.execute()
            else:
                vk.method("messages.send", {"peer_id": id,
                                            "message": "Такого типа поиска
нет &#128533;",
                                            "random_id": random_id})
                step = 1  # Возвращаем к началу диалога
                query = Users.update(step=step).where(Users.id_user == id)
                query.execute()
    random_id = random_id + 1
    time.sleep(0.5)
```

Приложение Б

Интерфейс клиентской части рекомендательной системы

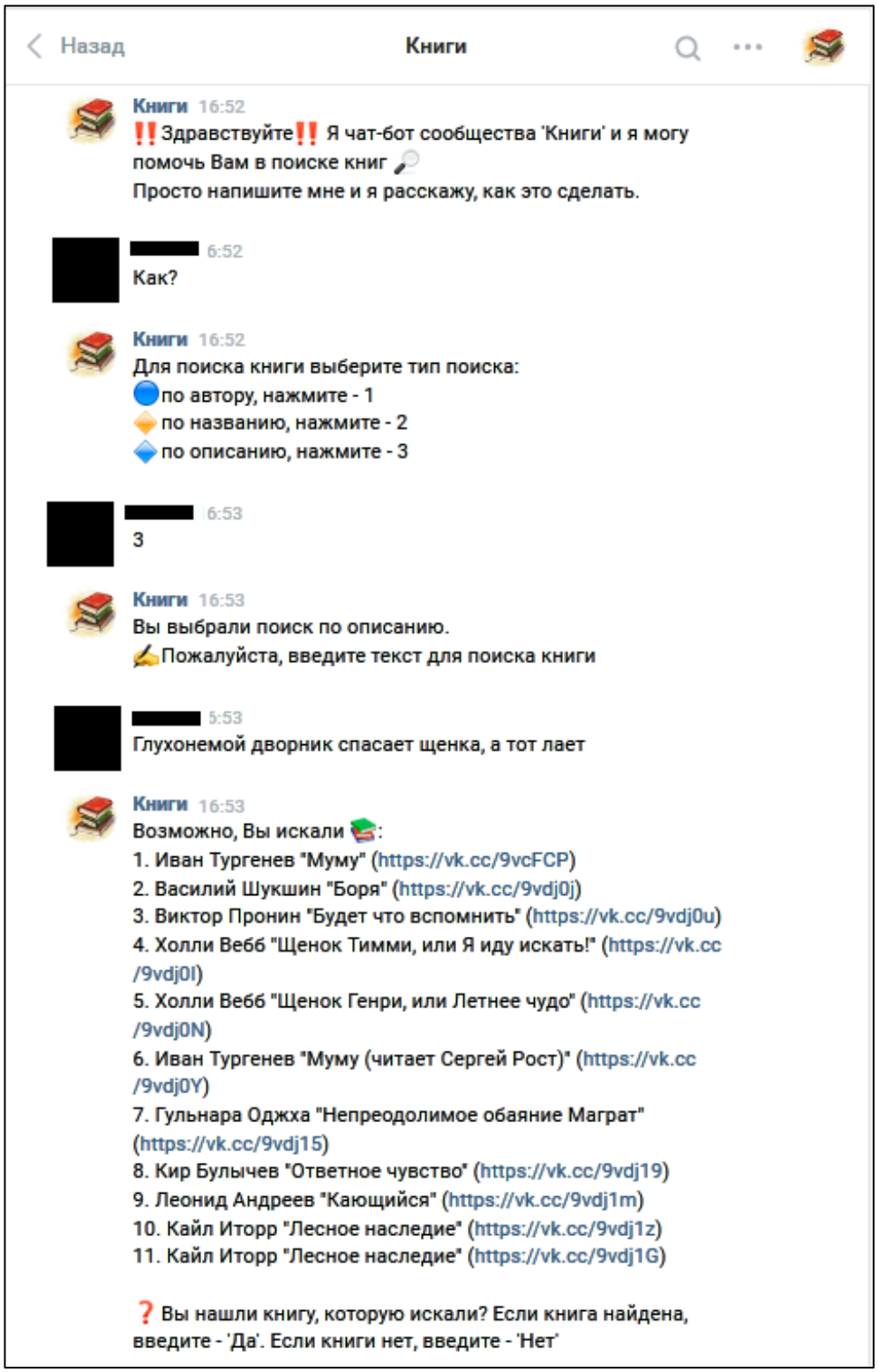

Рисунок 1. Поиск по описанию

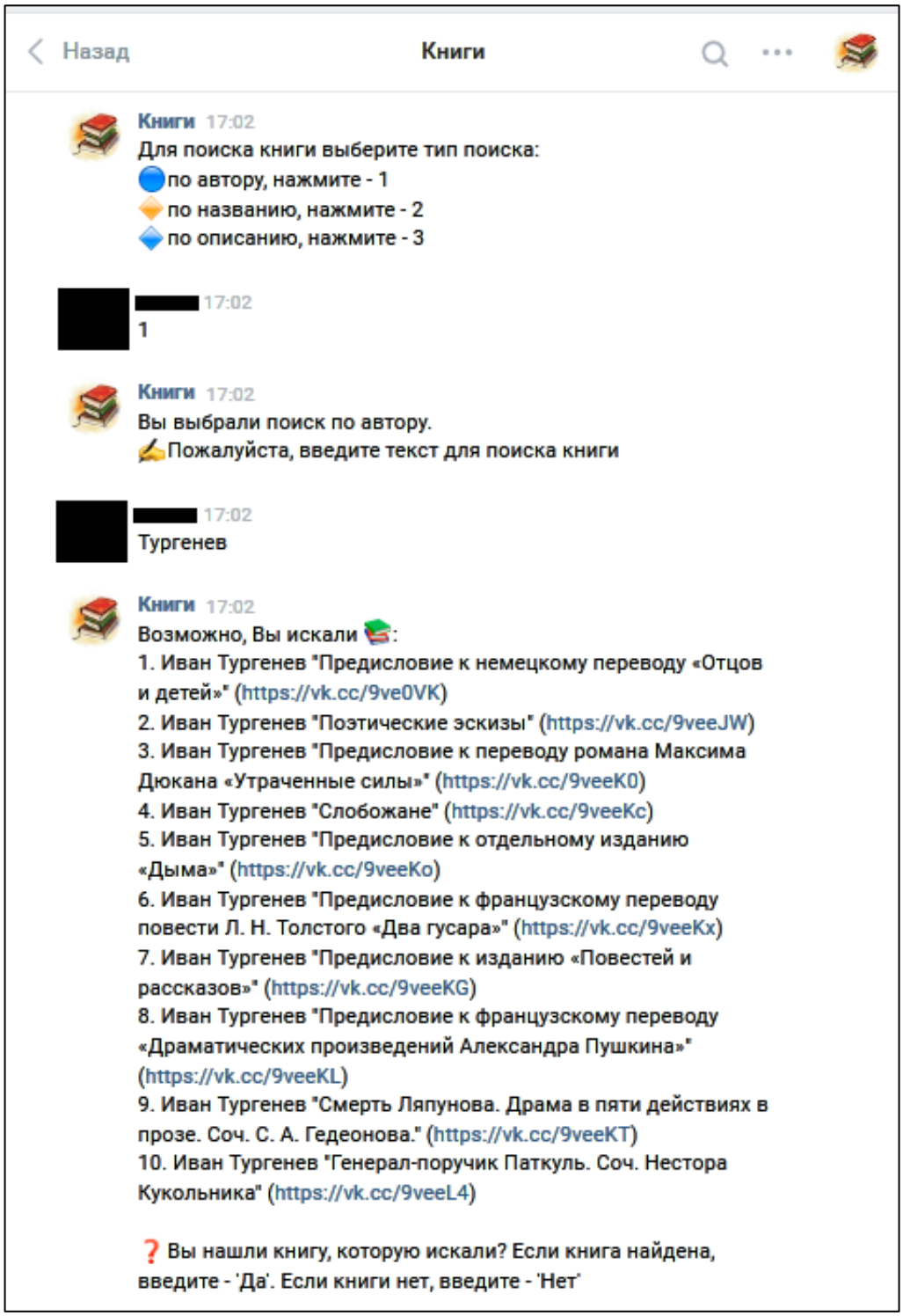

Рисунок 2. Поиск по автору

Рисунок 3. Поиск по названию

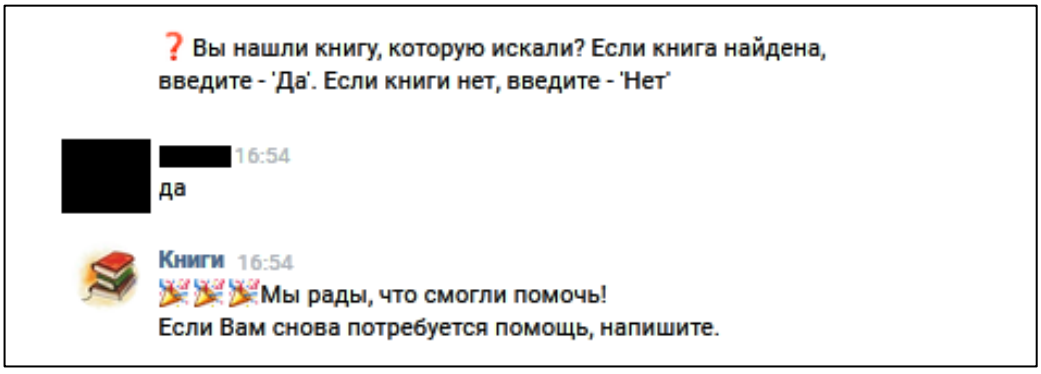

Рисунок 4. Искомая книга найдена

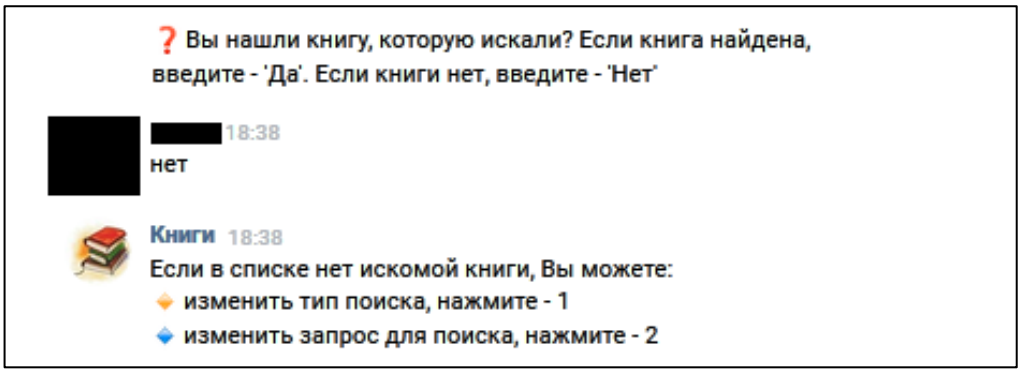

Рисунок 5. Возврат к предыдущим шагам