

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

Работа допущена к защите

Директор ВШКТиИС

_____ В.А. Сушников

«____» _____ 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА
ПРИМЕНЕНИЕ МЕТОДОВ ОБРАБОТКИ ЕСТЕСТВЕННОГО
ЯЗЫКА В РЕКОМЕНДАТЕЛЬНЫХ СИСТЕМАХ

по направлению подготовки: 09.03.01 «Информатика и вычислительная техника»

Направленность (профиль): 09.03.01_02 «Технология разработки программного обеспечения»

Выполнил:
студент гр. 5130901/00201 *<подпись>* С. А. Шульгин

Руководитель:
к.т.н., доцент ВШКТиИС *<подпись>* Е. Н. Бендерская

Консультант
по нормоконтролю *<подпись>* А. Г. Новопашенный

Санкт-Петербург – 2024

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО**

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

УТВЕРЖДАЮ

Руководитель ОП

Б.А. Сушников

«_____» _____ 2024 г

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Шульгину Сергею Александровичу гр. 5130901/00201
фамилия, имя, отчество (при наличии), номер группы

1. Тема работы: Применение методов обработки естественного языка в рекомендательных системах.
 2. Срок сдачи студентом законченной работы: 22.05.2024
 3. Исходные данные по работе: научные работы по методам обработки естественного языка, открытый набор данных веб-сервиса «GoodReads», документация языка Python и библиотек Pandas, Numpy, Scikit-learn и SciPy.
 4. Содержание работы (перечень подлежащих разработке вопросов):
 - 4.1. Обзор предметной области;
 - 4.2. Проектирование рекомендательной системы;
 - 4.3. Реализация рекомендательной системы;
 - 4.4. Тестирование рекомендательной системы.
 5. Перечень графического материала (с указанием обязательных чертежей): графические материалы, используемые в презентации.
 6. Консультанты по работе:
 7. Дата выдачи задания: 25.04.2024

Руководитель ВКР

<подпись>

Е. Н. Бендерская

Задание принял к исполнению: 2 февраля 2024 г.

Студент

РЕФЕРАТ

На 92 с., 53 рисунка, 5 таблиц, 36 источников, 3 приложения.

КЛЮЧЕВЫЕ СЛОВА: РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА, ВЕКТОРИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКИХ ПРЕДПОЧТЕНИЙ, ПОЛЬЗОВАТЕЛЬСКОЕ ПРОСТРАНСТВО ЭМБЕДДИНГОВ, DOC2VEC, K-MEDOIDS, КОСИНУСНОЕ СХОДСТВО.

Тема выпускной квалификационной работы: «Применение методов обработки естественного языка в рекомендательных системах».

Цель работы заключается в разработке ядра конкурентноспособной рекомендательной системы с учётом последних достижений в области машинного обучения.

Были рассмотрены различные методы построения рекомендательных систем, а также проблемы, возникающие при их реализации, и возможные пути их решения. Кроме того проанализированы способы применения методов обработки естественного языка в разработке рекомендательных систем.

В рамках выпускной квалификационной работы была предложена модель векторизации пользовательских предпочтений Pref2Vec как основа гибридной рекомендательной системы. Модель использует один из наиболее актуальных методов обработки естественного языка – векторизацию, а также расширяет функциональность библиотеки Gensim. К тому же был предложен класс пользовательского пространства эмбеддингов UES, который используется при построении модели Pref2Vec и который позволяет векторизовать предпочтения конечного пользователя посредством применения нового способа векторизации текстовых данных, при помощи которого учитывается вид взаимодействия пользователя с рассматриваемыми данными.

Тестирование разработанной системы и сравнительный анализ полученных значений метрик Recall@k и NCDG@k показали, что результаты оценки качества модели Pref2Vec сравнимы с показателями моделей BPRMF, GRU4Rec и NextItRec, что говорит о перспективности разработанного ЯРС.

ABSTRACT

92 pages, 53 figures, 5 tables, 36 sources, 3 appendices.

KEYWORDS: RECOMMENDER SYSTEM, VECTORIZING OF USER PREFERENCES, USER EMBEDDINGS SPACE, DOC2VEC, K-MEDOIDS, COSINE SIMILARITY.

The topic of the thesis is «Application of natural language processing methods in recommendation issues».

The purpose of the thesis is to analyze the various machine learning methods in the development of recommender systems and to develop the core of a competitive recommender system.

Main aspects of the recommender system development were considered: methods for constructing recommender systems, problems of its implementation, ways to solve the problems and ways to apply machine learning methods in the development.

As one of the results, a model of vectorizing user preferences Pref2Vec was proposed as the basis of a hybrid recommender system. The model uses one of the most current methods of natural language processing – vectorization, and also expands the functionality of the Gensim library. In addition, a user space embedding class UES was proposed, which is used in building of the Pref2Vec model and which allows to vectorize end-user preferences by a new method of vectorizing text data, which allows to consider the type of user interaction with the data.

Testing of the Pref2vec and comparative analysis of the Recall@k and NCDG@k metrics indicate that the achieving quality of the Pref2Vec model is comparable with the baselines of the BPRMF, GRU4Rec and NextItRec models, which indicates the prospects of the developed core of a recommender system.

СОДЕРЖАНИЕ

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ.....	7
ВВЕДЕНИЕ	8
ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ	9
1.1. Задача рекомендательных систем	9
1.2. Методы построения.....	10
1.2.1. Контентная рекомендация.....	10
1.2.2. Совместная рекомендация	11
1.2.3. Рекомендация на основе знаний.....	13
1.2.4. Гибридная рекомендация	15
1.3. Проблемы построения и способы их решения	16
1.3.1. Оценка работы системы и доступность датасетов	16
1.3.2. Холодный старт.....	17
1.3.3. Переобучение моделей	17
1.3.4. Разреженность.....	18
1.3.5. Проблема «серых овец»	18
1.3.6. Масштабируемость.....	19
1.3.7. Синонимия	19
1.3.8. Контекстная зависимость.....	20
1.3.9. Шиллинговые атаки	20
1.3.10. Конфиденциальность	20
1.4. Применение методов машинного обучения	21
1.4.1. Эмбеддинги и модели 2Vec	21
1.4.2. Глубокая коллаборативная фильтрация	22
1.4.3. Извлечение полезной информации из контента	23
1.5. Цель и задачи выпускной квалификационной работы.....	24
1.6. Выводы по первой главе	25
ГЛАВА 2. ПРОЕКТИРОВАНИЕ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ	26
2.1. Требования к рекомендательной системе.....	26
2.2. Pref2Vec: векторизация пользовательских предпочтений	27
2.3. Построение рекомендаций.....	31
2.4. Методологии тестирования	32
2.4.1. Списковые метрики	33
2.4.2. Поэлементные метрики.....	34
2.4.3. Глобальные метрики.....	34
2.4.4. Выбор используемых метрик	35
2.5. Технологический стек	35

2.5.1.	Библиотеки методов обработки естественных языков.....	36
2.5.2.	Библиотеки предобработки данных.....	36
2.5.3.	Библиотеки методов машинного обучения	37
2.5.4.	Библиотеки визуализации данных	37
2.6.	Выводы по второй главе.....	38
ГЛАВА 3. РЕАЛИЗАЦИЯ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ.....		39
3.1.	Подготовка исходных данных	39
3.1.1.	Обзор исходного набора данных	39
3.1.2.	Обработка исходного набора данных	42
3.1.3.	Формирование корпуса книжных аннотаций	44
3.1.4.	Формирование датасета пользовательских предпочтений	45
3.2.	Формирование эмбеддингов	47
3.2.1.	Подготовка корпуса модели Doc2Vec	47
3.2.2.	Определение и обучение модели Doc2Vec	48
3.2.3.	Анализ работы модели Doc2Vec	50
3.3.	Разработка класса пользовательского пространства эмбеддингов	55
3.3.1.	Расширение исходных эмбеддингов	56
3.3.2.	Разработка класса UES	56
3.3.3.	Анализ работы класса UES	58
3.4.	Разработка модели векторизации пользовательских предпочтений	62
3.4.1.	Исследование пользовательских пространств эмбеддингов	62
3.4.2.	Разработка модели Pref2Vec	64
3.4.3.	Анализ работы модели Pref2Vec	70
3.5.	Выводы по третьей главе	74
ГЛАВА 4. ТЕСТИРОВАНИЕ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ.....		76
4.1.	Подготовка тестовых данных	76
4.2.	Разработка класса оценки работы рекомендательной системы	78
4.3.	Тестирование модели Pref2Vec.....	79
4.4.	Оценка перспективности разработанной модели	84
4.5.	Выводы по четвёртой главе	85
ЗАКЛЮЧЕНИЕ.....		87
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		89
ПРИЛОЖЕНИЕ 1. КОД КЛАССА ПОЛЬЗОВАТЕЛЬСКОГО ПРОСТРАНСТВА ЭМБЕДДИНГОВ		93
ПРИЛОЖЕНИЕ 2. КОД МОДЕЛИ ВЕКТОРИЗАЦИИ ПОЛЬЗОВАТЕЛЬСКИХ ПРЕДПОЧТЕНИЙ		99
ПРИЛОЖЕНИЕ 3. КОД КЛАССА ОЦЕНКИ РАБОТЫ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ		108

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

Модели 2Vec – модели преобразований сущностей в эмбеддинги и последующей работы с ними.

Оболочка рекомендательной системы (ОРС) – часть РС, в состав которой входят интерфейсы взаимодействия внешней среды, в которую внедрена РС, и собственно РС.

Рекомендательная система (РС) – система, которая осуществляет рекомендацию контента пользователю на основе извлечённой информации о пользователе, контенте и взаимодействии пользователя и контента.

Эмбеддинг – вектор скрытых признаков объектов, который используется в РС для инициализации представления объекта в более продвинутых алгоритмах.

Ядро рекомендательной системы (ЯРС) – часть РС, включающая в себя модель построения рекомендаций и методы предобработки входных данных.

Doc2Vec – модель представления документов (текстов) в виде эмбеддингов, которая построена на основе модели Word2Vec.

NCDG@k – метрика оценки работы РС, которая демонстрирует нормализованную сумму выигрышей (попаданий реальных взаимодействий пользователя с объектами в списке из k рекомендаций с учётом их положения).

Pref2Vec – предлагаемая модель векторизации пользовательских предпочтений.

Recall@k – метрика оценки работы РС, которая демонстрирует долю реальных взаимодействий пользователя с объектами в списке из k рекомендаций.

Word2Vec – модель представлений слов в виде эмбеддингов, использующая нейронную сеть для создания векторных представлений слов на основе их контекста в большом корпусе текста.

ВВЕДЕНИЕ

Разработка РС является активно развивающейся областью информационных технологий, которая находит активное применение в оптимизации поиска необходимого контента. РС – это довольно эффективный инструмент обработки большого объёма данных, который способен помочь пользователю в поиске предпочтительной информации путём анализа персональных потребностей пользователей.

Многие РС используют один из самых перспективных подходов в искусственном интеллекте — машинное обучение. Алгоритмы машинного обучения позволяют оптимизировать поиск того, что актуально для пользователя, при помощи обучения, основанного на выявлении эмпирических закономерностей в большом объёме данных.

Применение методов обработки естественного языка позволяет изменить устоявшуюся парадигму разработки РС: данные о контенте, пользователях и пользовательских сессиях рассматриваются в рамках модели естественного языка. Данный подход открывает новые возможности в интерпретации большого количества данных.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Задача рекомендательных систем

Задачей любой РС является извлечение и использование информации о пользователе, контенте и взаимодействии пользователя и контента в целях рекомендации объектов пользователю [11].

В процессе работы РС собирают данные о пользователях, используя сочетание явных и неявных методов.

Примеры явного сбора данных:

- Возможность пользователя отметить понравившийся ему объект (здесь и далее единица контента будет называться объектом);
- Запрос у пользователя оценки объекта по дифференцированной шкале;
- Предложение пользователю создать список понравившихся объектов.

Примеры неявного сбора данных:

- Наблюдение за тем, какие объекты рассматривает пользователь;
- Журналирование поведения пользователя;
- Отслеживание содержимого устройства пользователя.

Процесс извлечения информации о контенте так же является необъемлемой частью работы РС. С этой целью применяются методы интеллектуального анализа данных (Data Mining) [14].

В интеллектуальном анализе данных используются различные статистические методологии и различные алгоритмы, такие как модели классификации, кластеризации и регрессионные модели. Основная цель данных алгоритмов и методологий — это поиск закономерностей и выявление полезной информации в большом объёме данных. Более детально этот аспект будет рассмотрен в п. 1.4.

1.2. Методы построения

Ввиду наличия различных видов информации, обрабатываемой РС (данные о контенте, о пользователе и о взаимодействии пользователя с контентом), существуют различные методы построения РС. Последующее рассмотрение методов будет основываться на книге «Recommender Systems: An Introduction» [11] и докладе «Tutorial: Recommender Systems» [12].

1.2.1. Контентная рекомендация

РС на основе контентной фильтрации — это подмножество РС, которые адаптируют рекомендации для пользователей путём анализа внутренних характеристик и атрибутов объектов. На рис. 1.1 схематически изображена такая РС в виде черного ящика, который преобразует входные данные в ранжированный список элементов на выходе. Потенциальные типы входных данных представляют из себя данные пользователя, параметры контекста и атрибуты объектов.

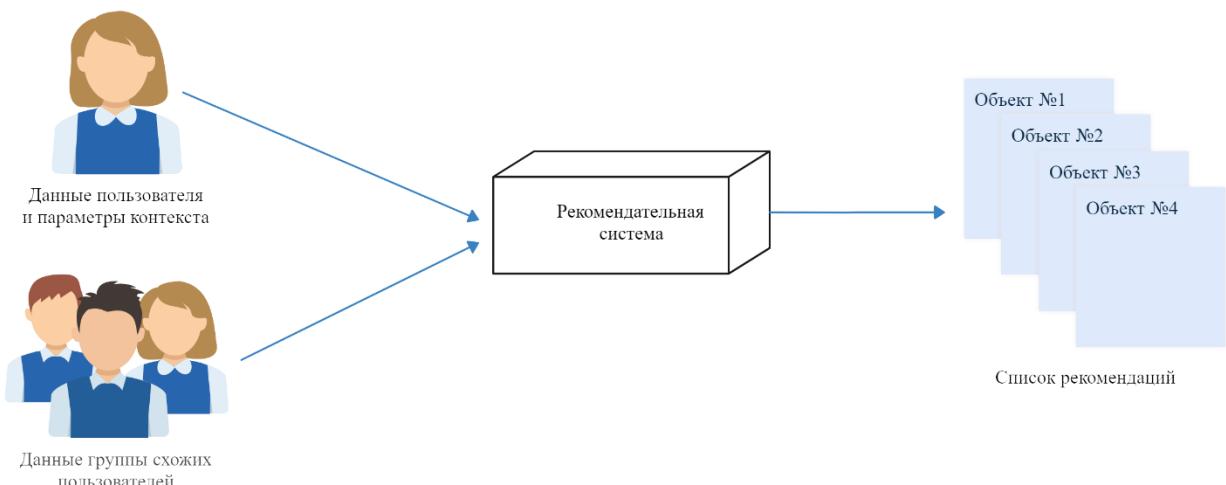


Рис. 1.1. Схема рекомендательной системы, построенной методом контентной рекомендации, в виде черного ящика.

Эти системы сосредоточены на понимании содержимого объектов и сопоставлении его с предпочтениями пользователей (рис. 1.2). В процессе не задействуются данные об истории взаимодействий пользователя с объектами, что делает контентные модели особенно полезными в сценариях, где история пользователей ограничена или недоступна. Благодаря контентно-ориентированному подходу рассматриваемые системы рекомендаций играют важную роль в улучшении опыта взаимодействия пользователя со средой, в которой работает система.

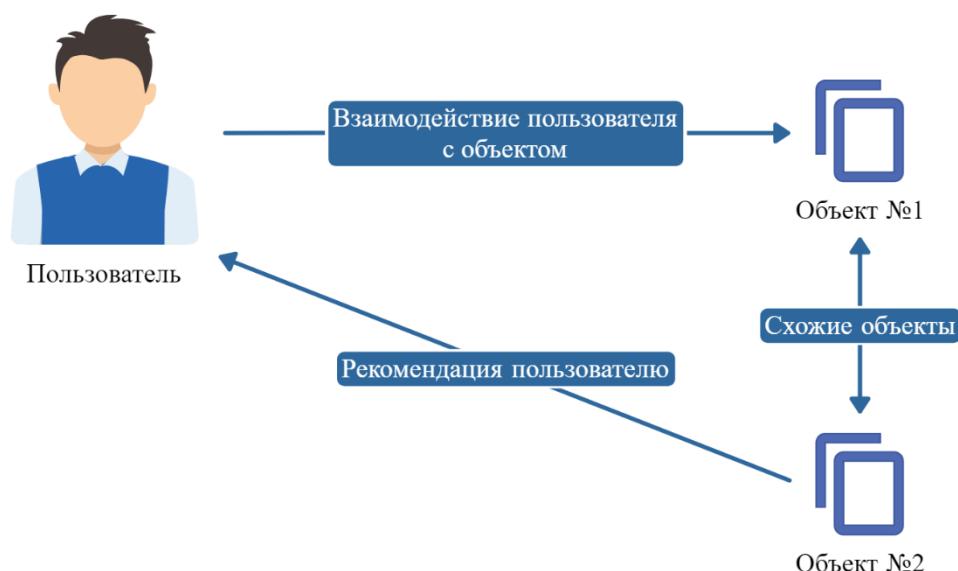


Рис. 1.2. Механика работы метода контекстной рекомендации.

Также необходимо отметить, что такой подход требует наличия характеристик и атрибутов объектов. Кроме того, при реализации данной модели встречается проблема холодного старта новых пользователей.

1.2.2. Совместная рекомендация

РС, построенные на методе совместной фильтрации, анализируют данные о взаимодействиях пользователей с объектами. Этот подход нацелен на прогнозирование поведения пользователя путём поиска схожих паттернов взаимодействий. Если схематически представить подобные РС в виде черного ящика, который преобразует входные данные в ранжированный список

элементов на выходе (рис. 1.3), то можно сказать, что на вход системы будут подаваться данные пользователя, параметры контекста и данные группы схожих пользователей.

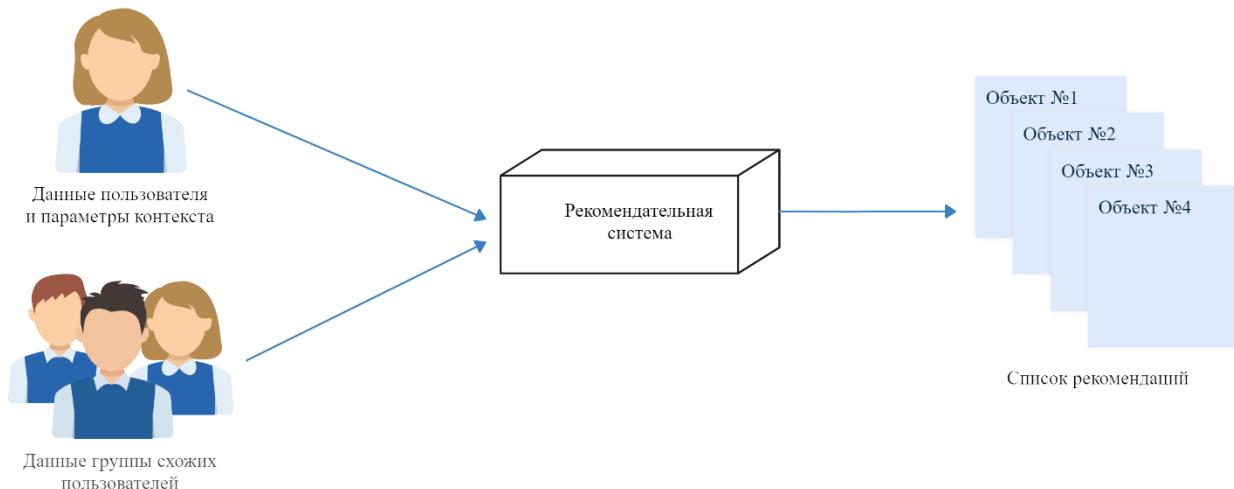


Рис. 1.3. Схема рекомендательной системы, построенной методом совместной рекомендации, в виде черного ящика.

Большинство систем коллаборативной фильтрации применяют метод матричного разложения, который предполагает реализацию подхода к-ближайших соседей. То есть при построении рекомендации для конкретного пользователя происходит поиск пользователей, находящихся рядом в векторном пространстве, полученном из истории взаимодействий пользователей с объектами. Их близость в векторном пространстве можно интерпретировать как предположительную схожесть интересов, что, в свою очередь, позволяет экстраполировать поведение схожих пользователей на поведение текущего (рис. 1.4).

Однако, данный подход накладывает некоторые ограничения на виды взаимодействий с объектами: для наиболее эффективной работы системы требуется определенный вид обратной связи — дифференцированная оценка. Из-за этого недостатка произрастают проблемы холодного старта пользователей и объектов.

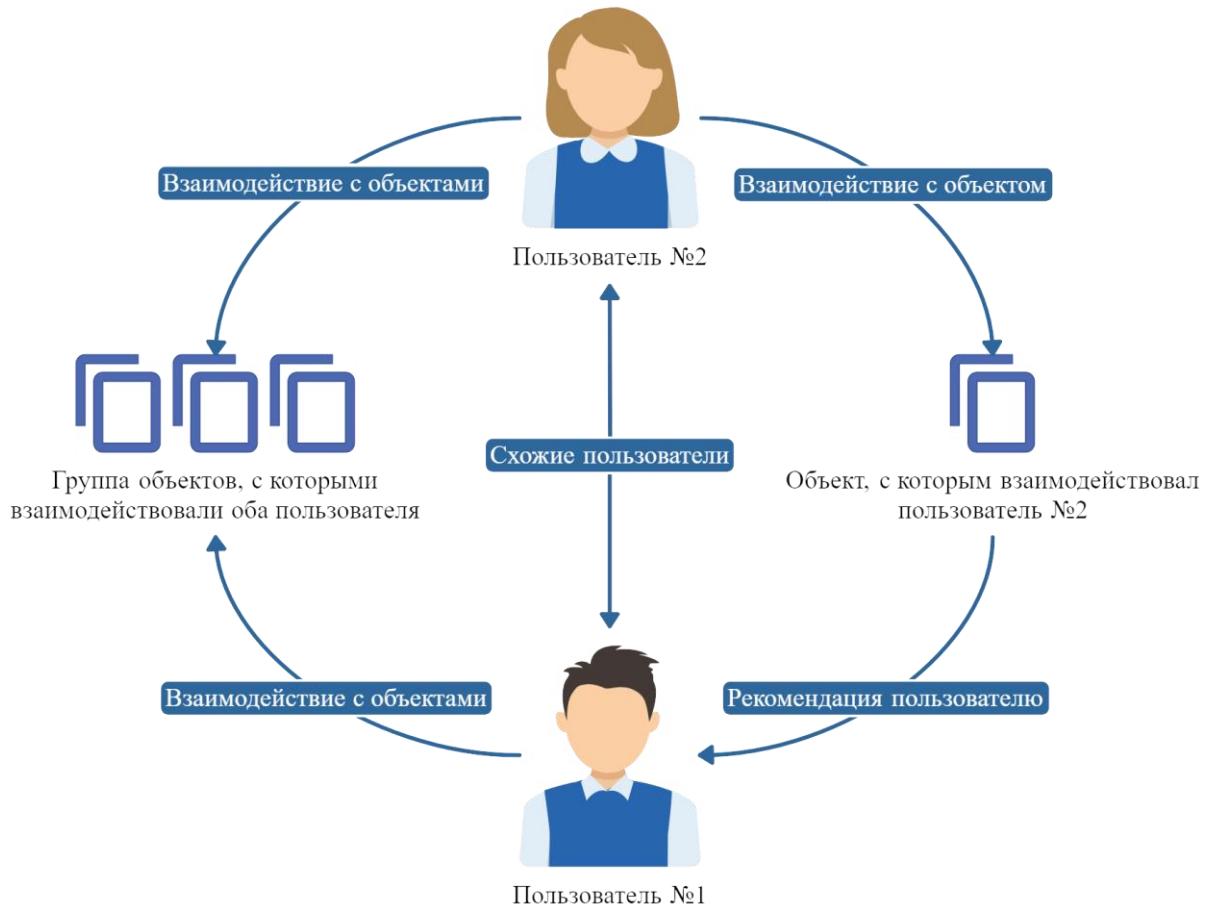


Рис. 1.4. Механика работы метода совместной рекомендации.

1.2.3. Рекомендация на основе знаний

Главное отличие РС, использующих рекомендацию на основе знаний, в том, что они фокусируются не на данных об объектах и пользователях, а на выявлении глубоких зависимостей, при помощи которых строятся более специфицированные рекомендации. Данный подход способствует повышению уровня персонализации поиска контента, так как подобные системы используют более детерминированные требования к поиску рекомендаций. Также схематически представим РС в виде черного ящика (рис. 1.5). Потенциальные типы входных данных будут включать в себя данные пользователя, параметры контекста, атрибуты объектов и данные моделей знаний.

Существуют два основных метода построения РС, основанных на знаниях: метод ограничений и метод прецедентов. Оба метода предполагают, что требования к рекомендациям будут трактоваться на основе знаний, полученных от взаимодействия пользователя с объектами. В методе ограничений предоставленные системе знания используются для построения явного определённого набора критериев, на основе которого будут производиться рекомендации: формулируются некоторые правила рекомендаций, и происходит поиск набора элементов, удовлетворяющих этим правилам. При методе прецедентов знания используются для поиска схожих элементов на основе различных мер сходства — требования поиска определяются прецедентами.

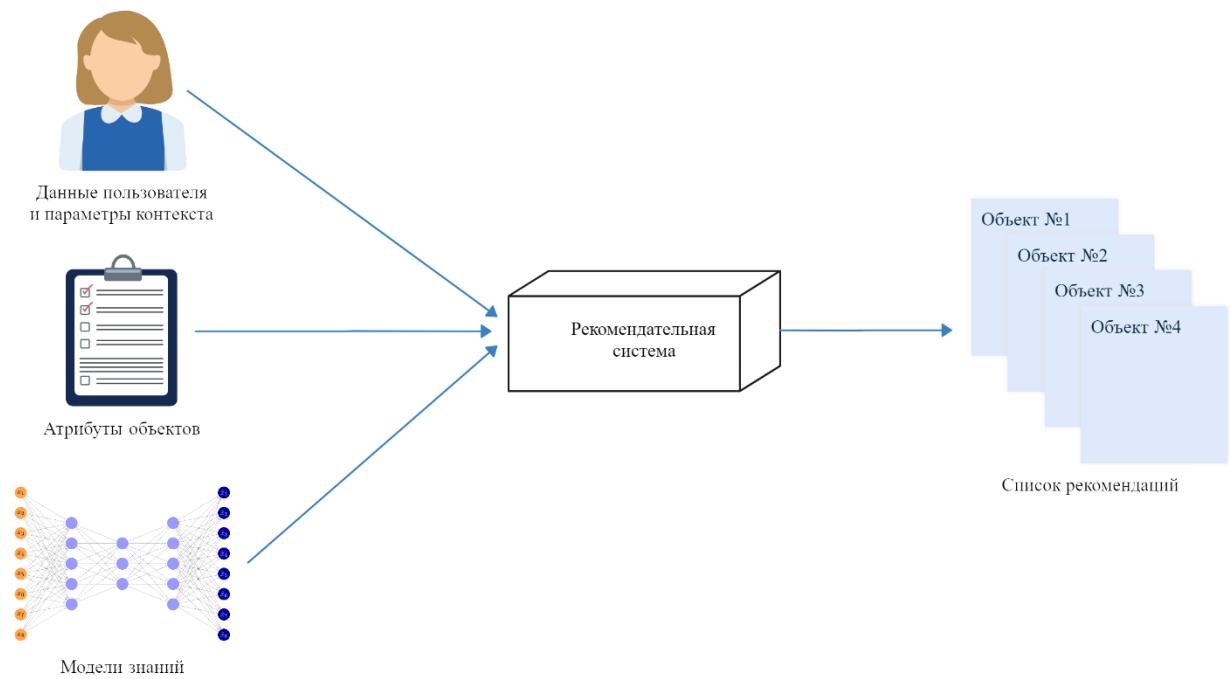


Рис. 1.5. Схема рекомендательной системы, построенной методом рекомендации на основе знаний, в виде черного ящика.

Использование подхода рекомендации на основе знаний даёт гарантию качества результата, также позволяет избавиться от проблемы холодного старта объектов. Но в то же время большинство моделей знаний не реагируют на краткосрочные тренды, что накладывает на работу РС определённые ограничения. Кроме того, следует учитывать, что применение только этого

подхода при разработке системы, предполагает её внедрение в среду, в которой не целесообразно рекомендовать пользователю похожие объекты.

1.2.4. Гибридная рекомендация

Гибридные РС — это системы, сочетающие в себе несколько реализаций алгоритмов или рекомендательных компонентов. При схематическом изображении такой РС в виде черного ящика (рис. 1.6) можно выяснить, что типы входной информации будут представлять из себя данные пользователя, параметры контекста, данные группы схожих пользователей, атрибуты объектов и данные моделей знаний.

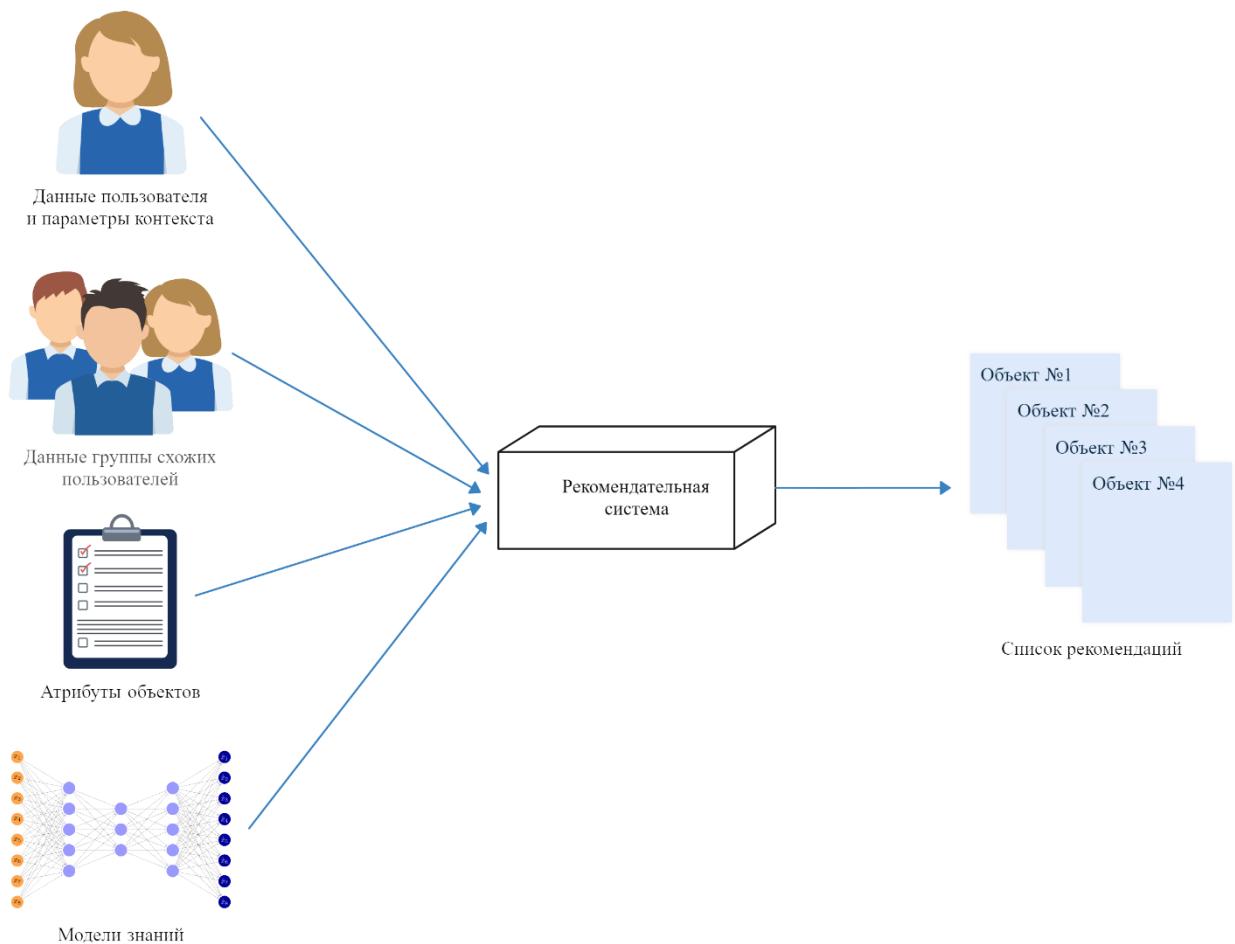


Рис. 1.6. Схема рекомендательной системы, построенной методом гибридной рекомендации, в виде черного ящика.

Раннее рассмотренные подходы хоть и дают результаты, которые считаются довольно персонализированными, работают с разной степенью успеха в разных областях применения. Совместная рекомендация использует пользовательские данные, подход рекомендации на основе знаний полагается на атрибуты объектов, алгоритмы, основанные на знаниях, обрабатывают информацию о взаимодействиях пользователя с объектами.

Ни один из базовых подходов не способен полностью использовать весь спектр данных. Таким образом, гибридная рекомендация обеспечивает более комплексный подход к решению проблемы построения рекомендаций. Именно поэтому целью последующих исследований стало построение гибридных систем, сочетающих сильные стороны различных алгоритмов и моделей.

1.3. Проблемы построения и способы их решения

Несмотря на многочисленные достижения в области исследований и разработок РС, интерес к этой области по-прежнему широк из-за растущих требований пользователей. Поэтому необходимо создавать высококачественные РС для предоставления точно настроенных рекомендаций пользователям. Для этого следует уделить должное внимание ряду проблем, с которым сталкиваются множество разработчиков [15].

1.3.1. Оценка работы системы и доступность датасетов

Ключевая проблема создания РС — это разработка критериев оценки работы систем и выбор подходящих показателей этой оценки.

В разработке большинства РС общего применения оценка получаемых результатов производится при помощи набора данных, в котором выделена часть тестовых данных, и метрик MAE, Precision и F-Measure. Для оценки контекстно-зависимых РС используются Contextual Precision и Contextual ROC. Также существуют методы анкетирования, исследования поведения

пользователей и проведения А/В-тестов, но эти подходы отнимают достаточно много времени.

К тому же существует проблема ограниченной доступности наборов эталонных данных, которые можно использовать при оценке РС в конкретной предметной области. Большая часть доступных наборов данных является собственностью компаний и поэтому не может использоваться в качестве эталонных наборов данных при оценке контекстно-зависимых РС.

1.3.2. Холодный старт

Проблема холодного старта — это ситуация, при которой в систему добавляются новые пользователи или объекты. В таких случаях невозможно определить предпочтения нового пользователя или неясная стратегия внедрения нового объекта ввиду отсутствия необходимой информации, что приводит к менее точным рекомендациям.

Существуют различные способы решения данной проблемы:

- Попросить нового пользователя оценить некоторые объекты;
- Попросить нового пользователя явно указать свои предпочтения;
- Предлагать пользователю объекты на основе собранной демографической информации;
- Производить обработку атрибутов и метаданных объектов или (и) пользователей.

1.3.3. Переобучение моделей

Контентная рекомендация полагается на обработку характеристик и атрибутов объектов методами интеллектуального анализа данных. Но ввиду ограниченного объёма контента возможно возникновение проблемы переобучения моделей, что приводит к снижению качества рекомендаций. Также эта проблема может возникнуть при большом объёме данных, так как в таких случаях применяется ограниченный контент-анализ.

Чтобы решить проблему переобучения, можно ввести дополнительные приемы внедрения случайности, чего можно достичь, например, с помощью генетических алгоритмов, которые вносят разнообразие в даваемые рекомендации. Но следует отметить, что проблема относительно невелика, и на неё более целесообразно обращать внимание по её возникновению.

1.3.4. Разреженность

Наличие большого объема данных об объектах и нежелание пользователей оценивать объекты приводят к разреженности матрицы профилей в системах совместной рекомендации, что приводит к менее точным рекомендациям, так как разреженность затрудняет точность прогнозирования.

В таких ситуациях возможно использование следующих методов:

- Многомерную модель рекомендаций;
- Методы разложения по одному значению (SVD);
- Использование контентно-ориентированного метода совместной рекомендации.

1.3.5. Проблема «серых овец»

При использовании чистого метода совместной рекомендации в разработке РС часто встречается проблема «серых овец» — наличие пользователей, чьи мнения не совпадают с мнениями какой-либо группы, к которой они принадлежат (в рамках совместной рекомендации). Следовательно, подобные пользователи не будут удовлетворены результатами работы системы.

Данная проблема решается путём внедрения метода контентной рекомендации. Также «серых овец» можно идентифицировать и отделить от других пользователей, применяя методы автономной кластеризации.

1.3.6. Масштабируемость

В современных реалиях стоит также учитывать то, что количество информации, обрабатываемой различными системами, постоянно увеличивается. Следовательно, при разработке РС необходимо принимать во внимание скорость работы используемых алгоритмов с большими объёмами данных.

Проблему масштабируемости можно решить при помощи следующих методов:

- Алгоритмы кластеризации совместной рекомендации, которые ищут пользователей в небольших кластерах вместо поиска по всей базе данных;
- Уменьшение размерности данных с помощью SVD;
- Предварительная обработка, сочетающая кластеризацию и анализ контента с алгоритмами совместной рекомендации.

1.3.7. Синонимия

Синонимия возникает, когда предмет представлен двумя или более разными именами, имеющими схожее значение. В таких ситуациях РС не может определить, представляют ли термины разные элементы или один и тот же элемент. Ввиду данного факта чрезмерное использование схожих описательных терминов снижает эффективность рекомендаций.

Чтобы облегчить эту проблему, можно использовать следующие методы:

- Измерение сходства на основе онтологий для решения проблемных ситуаций;
- Методы разложения по одному значению (Singular Value Decomposition — SVD);
- Скрытое семантическое индексирование (Latent Semantic Indexing — LSI).

1.3.8. Контекстная зависимость

Наличие контекстной зависимости в РС позволяет получить информацию о среде, в которой работает система, и использовать её для повышения точности рекомендаций. Контекстом может являться текущее местоположение, время, краткосрочная и долгосрочная история действий пользователя т. д.

Добиться внедрения контекстной зависимости в РС можно использованием метода рекомендации на основе знаний при разработке.

1.3.9. Шиллинговые атаки

Также возможно присутствие злонамеренных пользователей системы, которые способны давать ложные реакции на объекты для увеличения или снижения его популярности. Подобные атаки могут подорвать доверие к системе рекомендаций, а также снизить производительность и качество рекомендаций.

Для обнаружения шиллинговых атак можно использовать различные подходы, такие как сдвиг прогноза или коэффициент попадания.

1.3.10. Конфиденциальность

Использование личной информации РС приводит к повышению качества рекомендательных услуг, но может привести к проблемам безопасности данных, из-за чего пользователи неохотно передают личные данные.

Для предотвращения неправомерного использования данных можно использовать криптографические механизмы, представляя персонализированные рекомендации без привлечения третьих лиц и одноранговых пользователей. Другие методы включают использование методов рандомизированного возмущения или, например, технологий семантической паутины.

1.4. Применение методов машинного обучения

Применение методов машинного обучения в разработке РС позволяет добиться более точного представления данных о пользователях и объектах, что достигается при выявлении закономерностей в большом объёме данных.

Машинное обучение предоставляет разнообразные возможности обработки непосредственно содержимого объектов (изображений, текстов, звуков и т. д.) для выявления полезной информации и динамического моделирования поведения пользователя. Также стоит отметить, что методы машинного обучения оптимизируют обработку разнородных данных, что достаточно актуально в рамках РС.

1.4.1. Эмбеддинги и модели 2Vec

При реализации матричного разложения (см. п. 1.2.2) матрица совместных данных получается довольно большой и разреженной. Решить данную проблему возможно при помощи применения эмбеддингов (*embeddings*) — векторов реальных значений, представляющих некоторую сущность. Это векторы скрытых признаков объектов, которые используются в РС для инициализации представления объекта в более продвинутых алгоритмах [3].

Одна из популярных моделей работы с эмбеддингами — *Word2Vec*. Эта модель обработки естественного языка, которая используется для представлений слов (word embeddings), использующая нейронную сеть для создания векторных представлений слов на основе их контекста в большом корпусе текста.

Есть несколько путей применения моделей 2Vec в разработке РС. Рассмотрим основные:

- Можно производить обработку текстовых данных объектов «в лоб» — получать их эмбеддинги и производить дальнейшие манипуляции с ними для построения рекомендаций;
- Также можно сказать, что объекты — это слова, а последовательности взаимодействий с ними в рамках одной сессии — это предложение. Модель выделяет паттерны слов, которые часто встречаются вместе в одном контексте, и использует полученные данные для вычисления вероятности соседства двух слов.

В разработке РС находят применение модели *Paragraph2vec* и *Doc2vec*, которые работают по таким же принципам, как и Word2Vec, но используются для представления абзацев и документов соответственно, то есть более длинных последовательностей взаимодействий пользователей с объектами [17].

1.4.2. Глубокая коллаборативная фильтрация

Матричное разложение (см. п. 1.2.2) использует скалярное произведение как меру схожести двух элементов, но такой подход не позволяет извлечь из данных более сложные зависимости. Нейронная коллаборативная фильтрация, напротив, позволяет сделать меру схожести более сложной и обучаемой [8].

При данном подходе на входной слой подаются разреженные данные, закодированные при помощи one-hot encoding (способ преобразования категориальных переменных в числовые значения). Далее находится полносвязный слой, который отвечает за преобразование разреженного one-hot представления в плотные эмбеддинги. После эмбеддинги конкатенируются, дальнейшая архитектура носит имя *Neural Collaborative Filtering (NCF)* и отвечает за преобразование эмбеддингов пары пользователь-объект непосредственно в рекомендацию. NCF является более общим вариантом классического матричного разложения, которое можно в точности воспроизвести, упростив слои следующие за слоем эмбеддингов.

Большинство моделей работают только с одним источником данных (рейтинг или текст), в то время как модели на основе автокодировщика могут обрабатывать гетерогенные источники данных (рейтинг, текст, изображения, аудио, видео). Кроме того, автокодировщик помогает РС быть более адаптируемой при обработке мультимедийных данных.

В настоящее время в сфере разработки РС используется множество вариантов автокодировщиков. Наиболее распространенными являются:

- *Автокодировщик шумоподавления (Denoising Autoencoder — DAE)*, который искажает входные данные перед отображением их в скрытое представление, а затем восстанавливает исходные входные данные из их поврежденной версии. Идея состоит в том, чтобы заставить скрытый уровень приобрести более надежные функции и не дать сети просто изучить функцию идентификации;
- *Вариационный автокодировщик (Variational Autoencoder — VAE)* — это неконтролируемая модель скрытых переменных, которая изучает глубокое представление на основе многомерных данных. Суть в том, чтобы закодировать входные данные как распределение вероятностей, а не как точечную оценку, как в обычном автокодировщике. Затем VAE использует декодер для восстановления исходных входных данных, оперируя выборками из этого распределения вероятностей.

1.4.3. Извлечение полезной информации из контента

Для выявления зависимостей в большом количестве данных и повышения точности рекомендаций активно используются *свёрточные нейронные сети (Convolutional Neural Networks — CNN)*.

CNN позволяют производить захват сложных паттернов, поэтому появляется возможность изучения сущностей высокого уровня, которые определяют предпочтения пользователей, путём анализа взаимодействия пользователя с объектами [34]. Также следует отметить, что свёрточные

нейронные сети дают возможность адаптировать рекомендации в зависимости от контекста пользователя: время, местоположение и демографические данные.

Свёрточные нейронные сети могут эффективно обрабатывать атрибуты объекта, такие как текстовые описания, изображения и аудио, для создания вложенных данных. Эти данные обеспечивают комплексное представление объектов, позволяя РС делать более точные прогнозы. Обработка атрибутов объекта, в свою очередь, смягчает проблему холодного старта как и для пользователей, так и для объектов.

Кроме того, при помощи свёрточных нейронных сетей можно решить проблему междоменных рекомендаций – ситуации, в которых рекомендации находящихся в разных контекстных областях.

1.5. Цель и задачи выпускной квалификационной работы

Основываясь на анализе состояния предметной области разработки РС, была сформулирована цель работы: разработка ядра конкурентноспособной РС с учётом последних достижений в области методов обработки естественного языка.

Следовательно, в рамках выполнения выпускной квалификационной работы предстоит решить следующие задачи:

- Адаптация существующих методов обработки естественного языка для их применения в РС и разработка новых подходов для преодоления недостатков и расширения функциональных возможностей проектирования ЯРС;
- Проектирование ЯРС, обеспечивающего решение проблем, описанных в обзоре предметной области;
- Выбор метрик, необходимых для оценки эффективности работы системы;
- Выбор технологического стека;

- Выбор исходных данных и их подготовка;
- Реализация ЯРС;
- Тестирование системы и оценка полученных результатов;
- Оценка перспективности разработки.

1.6. Выводы по первой главе

По результатам обзора предметной области была определена цель работы и задачи, которые подлежат решению.

В ходе выполнения выпускной квалификационной работы будет разработано ядро гибридной РС ввиду того, что гибридный метод построения рекомендаций является наиболее целесообразным.

При проектировании ЯРС следует сосредоточиться на решении следующих проблем: проблема разреженности данных, проблема масштабируемости разрабатываемой системы, проблема наличия контекстной зависимости и проблемы «серых овец», поскольку вышеперечисленные проблемы являются наиболее значительными. При реализации ЯРС придётся столкнуться с проблемами доступности датасетов и, вследствие этого, с проблемами оценки качества работы разработанной системы. Поэтому выбор данных и их подготовка для использования в разработке выделены в отдельную задачу.

Также в ходе анализа предметной области было рассмотрено применение машинного обучения в разработке РС. Ввиду того, что наиболее перспективными являются методы обработки естественного языка, а именно методы 2Vec, ЯРС будет реализована на основе вышеупомянутых методов.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ

Одним из наиболее актуальных методов обработки естественного языка, применимым в разработке РС является векторизация – формирование эмбеддингов, отображающих семантику взятого за основу текста. Оперирование векторными представлениями текста позволяет использовать в разработке достаточно простые, эффективные и интуитивно понятные алгоритмы. Например, применим алгоритм поиска ближайших соседей в рамках поиска ближайших эмбеддингов в пространстве эмбеддингов.

Практически все известные модели векторизации текста используются в отрыве от разработки РС – обрабатывают только собственно текст [3]. В рамках выполнения выпускной квалификационной работы представляется целесообразным разработать модель векторизации текстовой информации, учитывающую вид взаимодействия пользователя с обрабатываемым текстом, так как рассматриваемый подход помогает придерживаться принципов как и контентной рекомендации, так и совместной фильтрации в разработке РС.

2.1. Требования к рекомендательной системе

Перед тем, как начать проектирование ЯРС определим функциональные требования к системе:

- Векторное представление объектов ЯРС, которое учитывает как и семантику самих объектов, так и семантику видов взаимодействия пользователей с рассматриваемыми объектами, и которое решает проблему контекстной зависимости;
- Построение пользовательских векторных пространств для каждого пользователя, которые будут состоять из всех векторизованных объектов, с которыми взаимодействовал пользователь;

- Персонализация показываемого пользователю контента путём исследования рассчитанных векторных пространств;
- Ранжирование рекомендуемых объектов в зависимости от их положения в векторном пространстве.

Также определим и нефункциональные требования:

- Размерность формируемых эмбеддингов должна обеспечивать корректное отображение всех особенностей объектов;
- Разработанное ЯРС должно иметь возможность работы с большим количеством разреженных данных, что позволит решить проблемы разреженности данных и масштабирования системы;
- Метод извлечения данных разработанной ЯРС должен сглаживать возможные выбросы пользовательских данных, что позволяет решить проблему «серых овец»;
- Показатели оценки качества работы ЯРС должны быть на уровне актуальных моделей в сфере разработки РС.

2.2. Pref2Vec: векторизация пользовательских предпочтений

Предлагается модель *Pref2Vec* – модель векторного представления текста, которая способна учесть вид пользовательского взаимодействия и отобразить его в формируемом эмбеддинге. Впоследствии на основе разработанной модели будет строиться алгоритм формирования рекомендаций.

Изначальные эмбеддинги (векторные представления текста) будут формироваться при помощи модели Doc2Vec, которая позволяет формировать векторы признаков фиксированной длины из документов (фрагментов текста) посредством модели PV-DM (модель распределенной памяти векторов параграфов) [17]. Работа алгоритма основана на модели Word2Vec: задача векторизации слова состоит в его статистическом анализе (модели CBOW или Skip-gram) относительно других слов – поиске контекста его употребления.

Векторизация достигается при помощи нейронной сети со стохастическим градиентным спуском и применением обратного распространения ошибки [23]. Дальнейшая векторизация документов заключается в том, чтобы учесть векторы всех употреблённых слов в документе при помощи модели PV-DM и вышеупомянутой нейронной сети.

Исходя из того, что требуется преодолеть проблему контекстной зависимости, предлагается внести в векторные представления текста дополнительный семантический смысл – вид взаимодействия. Возможны два пути решения поставленной задачи:

- 1) Изменение или замена методов обработки естественного языка и методов машинного обучения, применяемых в ранее разработанных моделях векторизации текста, в целях модификации модели;
- 2) Последующая обработка эмбеддингов, сформированных ранее упомянутыми моделями.

Второй вариант является довольно перспективным и, к тому же, менее трудоёмким, что позволяет реализовать его в рамках выполнения выпускной квалификационной работы.

Расширение исходного эмбеддинга текста можно сравнить с формированием тона в лингвистике. Например, различные тоны в китайском языке интерпретируют одно и того же слово различным образом. Также и вид взаимодействия пользователя с объектом определяет контекст его «употребления».

Множества эмбеддингов различных видов взаимодействий можно интерпретировать как кластеры, поэтому предлагается новое применение задачи кластеризации – построение рекомендаций при помощи информации о кластерах взаимодействия различных пользователей. Учитывая довольно большую размерность исходных данных, при построении рекомендаций следует использовать только ключевую информацию о формируемых кластерах – расположение центров кластеров взаимодействий в пространстве.

Можем заявить, что центры кластеров отображают «усреднённые» предпочтения пользователя в рамках определённого вида взаимодействия.

В рамках решения проблемы разреженности данных и проблемы «серых овец» задача кластеризации посредством поиска центров кластеров взаимодействий позволяет уменьшить количество данных, участвующих в вычислениях, и уменьшить влияние «выбросов» при наличии «серых овец» в системе. Следовательно, такое неклассическое применение задачи кластеризации является довольно выгодным решением в проектировании модели Pref2Vec.

Рассмотрим возможные реализации поиска центров кластеров взаимодействий. Наиболее популярным решением задачи кластеризации является применение алгоритма K-Means [21]. Данный алгоритм является неиерархическим и итерационным. Он получил большую популярность благодаря своей простоте, наглядности реализации и достаточно высокому качеству работы. Но стоит отметить, что центры формируемых кластеров в процессе реализации алгоритма, могут не являться объектами кластеров. Данная особенность алгоритма K-Means помогает решить проблему «серых овец» только частично, и, ввиду этого факта, следует использовать алгоритм K-medoids [29]. Алгоритм в целом схож с K-Means, но центры кластеров, формируемых в процессе его реализации, как раз таки являются элементами рассматриваемых кластеров, то есть медоидами.

Таким образом процесс преобразования текстовой информации и пользовательских данных будет строиться следующим образом:

- 1) Тексты преобразуются в эмбеддинги при помощи модели векторизации текстовых данных;
- 2) Полученные эмбеддинги модифицируются: добавляется дополнительное измерение, которое позволит отобразить принадлежность объекта к тому или иному виду взаимодействия пользователя с контентом. Впоследствии происходит построение пользовательского пространства эмбеддингов –

векторного пространства, в котором располагаются векторы, несущие в себе как и семантику преобразованного текста, так вид взаимодействия;

- 3) В результате построения пользовательского пространства эмбеддингов формируются кластеры взаимодействий. После необходимо их проанализировать путём поиска их центров. Найденные центры кластеров будут использоваться для поиска пользователей с близайшими предпочтениями.

Представим вышеописанный алгоритм в виде схемы:

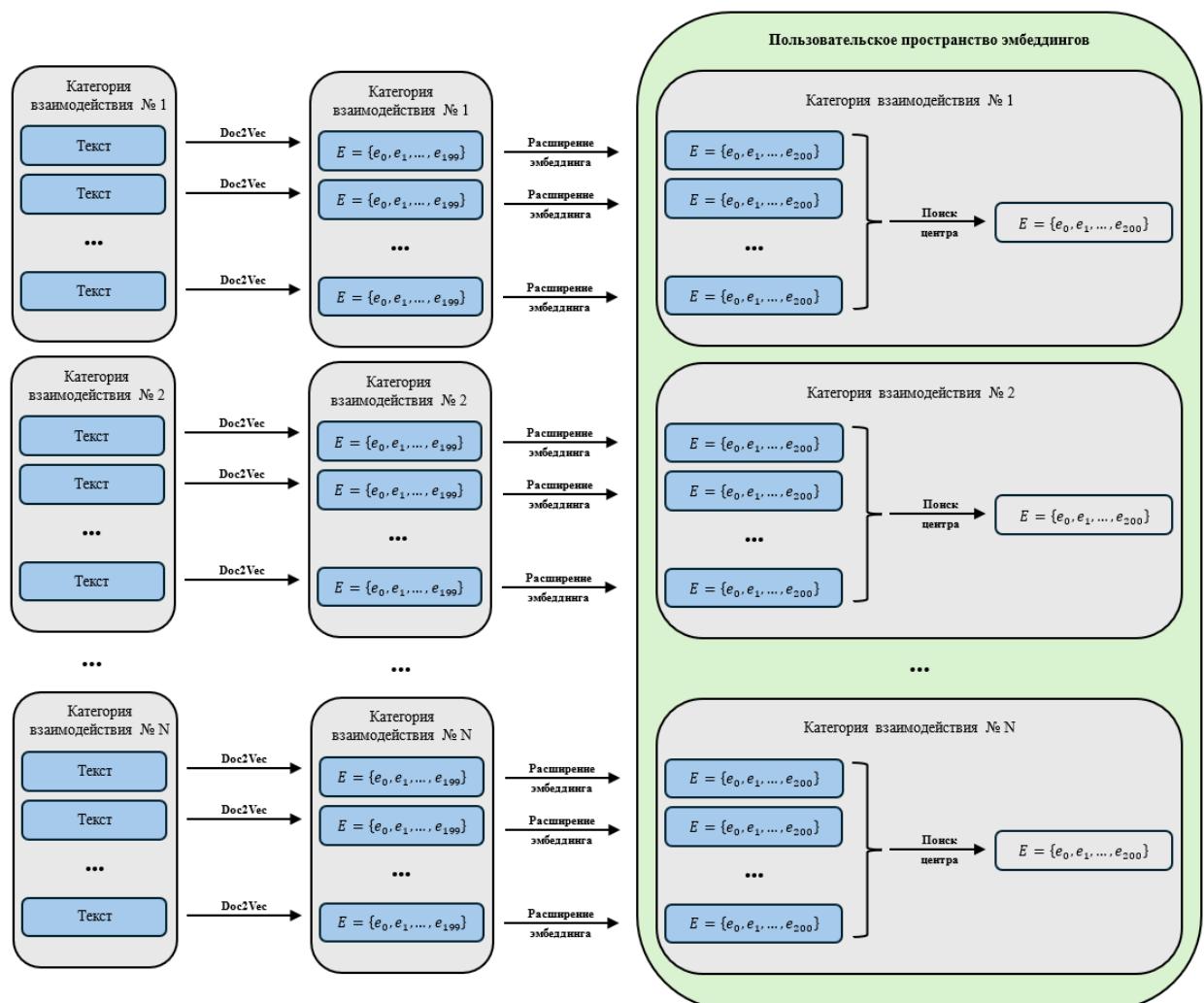


Рис. 2.1. Схема процесса преобразования текстовой информации и пользовательских данных моделью Pref2Vec.

2.3. Построение рекомендаций

В данной работе предлагается строить систему на более высоком уровне абстракции – предлагается алгоритм рекомендаций, при котором рекомендации будут строиться не в зависимости от рассматриваемого контента, а в зависимости от предпочтений конкретного пользователя, то есть от его пользовательского пространства эмбеддингов.

Алгоритм построения рекомендаций будет состоять в анализе множества пространств эмбеддингов и извлечении возможных рекомендаций из ближайших пользовательских пространств к целевому пространству. Механизм построения рекомендаций будет выглядеть следующим образом:

- 1) Для целевого пользователя вычисляется его пространство эмбеддингов (см. п. 2.2) – производится вычисление центров кластеров взаимодействий;
- 2) На основе рассчитанных центров производится поиск пользователей с похожими предпочтениями, то есть происходит исследование пользовательских пространств эмбеддингов, в ходе которого осуществляется поиск наиболее схожих пространств. Детальное описание поиска представлено в п. 3.4.1;
- 3) Элементы схожих пользовательских пространств сравниваются с элементами целевого пространства и ранжируются по уровню сходства. Более подробно анализ сходства так же описан в п. 3.4.1;
- 4) Ранжированный список элементов схожих пользовательских пространств является списком сформированных рекомендаций.

Вышеописанный алгоритм построения рекомендаций моделью Pref2Vec представлен в виде схемы на рис. 2.2.

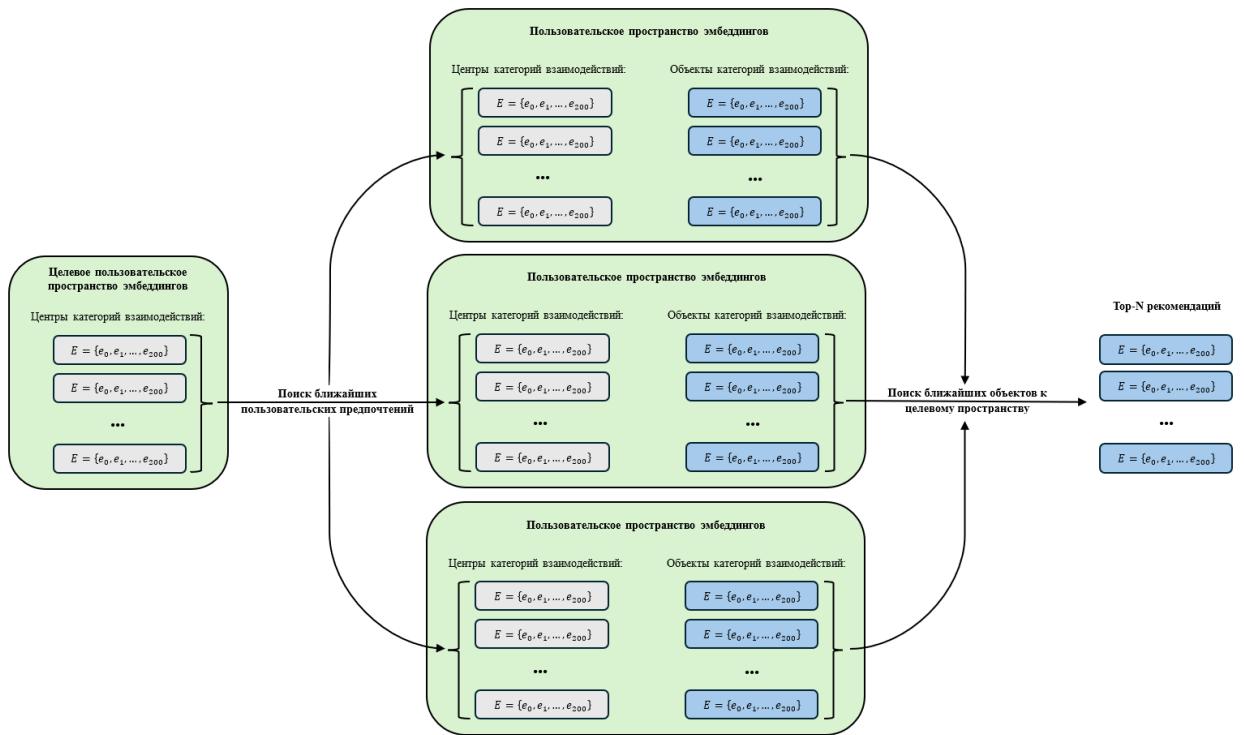


Рис. 2.2. Схема процесса построения рекомендаций моделью Pref2Vec.

2.4. Методологии тестирования

Один из важных аспектов разработки РС — это выбор критериев оценки качества алгоритмов рекомендаций. Важно отметить, что определить качество работы РС достаточно сложно ввиду нетривиальности задачи рекомендации. По этой причине набор используемых критериев оценки индивидуален для каждой разрабатываемой системы [28].

Для рассмотрения метрик обратимся к документации модуля метрик RecPack [22], который содержит большое количество метрик, обычно используемых для оценки state-of-the-art алгоритмов рекомендаций. Большинство метрик модуля относятся к метрикам категории Топ-К, которые используются для оценки качества ранжирования (используется параметр K, определяющий количество оцениваемых объектов).

2.4.1. Списковые метрики

Эти метрики возвращают одно значение оценки для каждого пользователя (табл. 1.1). Чтобы получить глобальное значение (см. п. 2.4.3), все оценки для каждого пользователя усредняются.

Таблица 1.1

Списковые метрики оценки категории Топ-К

Название	Формула	Описание
$DCG@K$ (Discounted Cumulative Gain)	$\sum_{i \in Top-K(u)} \frac{y_{u,i}^{true}}{\log_2(\text{rank}(u, i) + 1)}$	Сумма выигрышей объектов в списке рекомендаций (релевантные объекты имеют больший выигрыш, если находятся выше в списке)
$nDCG@K$ (Normalized Discounted Cumulative Gain)	$nDCG@K(u) = \frac{DCG@K(u)}{IDCG@K(u)}$ $IDCG@K(u) = \sum_{j=1}^{\min(K, y_u^{true})} \frac{1}{\log_2(j + 1)}$	Нормализованная сумма выигрышей всех объектов в списке рекомендаций (аналогичен DCG@K, но происходит нормализация путём деления полученной суммы совокупного выигрыша на наилучший возможный выигрыш из списка рекомендаций). IDCG@K — Ideal Discounted Cumulative Gain
$Recall@K$	$\frac{\sum_{i \in Top-K(u)} y_{u,i}^{true}}{\sum_{1j \in I} y_{u,j}^{true}}$	Доля реальных взаимодействий пользователя с объектами, попавших в рекомендации Топ-К
$CalibratedRecall@K$	$\frac{\sum_{i \in topK(u)} y_{u,i}^{true}}{\min(\sum_{1j \in I} y_{u,j}^{true}, K)}$	Количество обращений к рекомендациям Топ-К, деленное на минимальное значение К и количество истинных взаимодействий пользователя с объектами
$Precision@K$	$\frac{\sum_{i \in Top-K(u)} y_{u,i}^{true}}{K}$	Доля рекомендаций Топ-К, которые соответствуют истинным взаимодействиям пользователя с объектами
$ReciprocalRank@K$	$\frac{1}{\min_{i \in Top-K(u), i \in y_u^{true}} rank(u, i)}$	Обратная величина оценки первого попадания объекта в список рекомендаций

2.4.2. Поэлементные метрики

Метрики возвращают оценку для каждой пары «пользователь-объект» в рекомендациях Топ-К (табл. 1.2). Чтобы получить глобальное значение метрики (см. п. 2.4.3), эти оценки суммируются для каждого пользователя, а затем усредняются.

Таблица 1.2

Поэлементные метрики оценки категории Топ-К

Название	Формула	Описание
<i>Hit@K</i>	—	Количество попаданий в список рекомендаций Топ-К
<i>DiscountedGain@K</i>	$\frac{y_{u,i}^{\text{true}}}{\log_2(\text{rank}(u, i) + 1)}$	Выигрыш для каждого объекта в рекомендациях пользователя Топ-К (релевантные объекты имеют больший выигрыш, если находятся выше в списке)

2.4.3. Глобальные метрики

Глобальные метрики — это метрики, которые возвращают одно глобальное значение оценки для всех пользователей при тестировании (табл. 1.3).

Таблица 1.3

Глобальные метрики оценки категории Топ-К

Название	Формула	Описание
<i>Coverage@K</i>	$\frac{ I }{ \{i \in I (\exists u \in U)[i \in \text{TopK}(u)]\} }$	Доля всех взаимодействий пользователя с объектами, попавших в число рекомендаций Топ-К для любого пользователя
<i>PercentileRanking</i>	$\text{rank}_{u,i} = \begin{cases} \frac{\text{rank}_{u,i} - 1}{ I } & \\ \frac{\max_{j \in I} (\text{rank}_{uj}) + I }{2 I } & \end{cases}$	Ожидаемый процентильный рейтинг (метрика, представленная на международной конференции IEEE 2008 года [10])

2.4.4. Выбор используемых метрик

Как видно, существует множество метрик оценки качества работы РС, что ещё раз подчеркивает сложность их разработки и тестирования.

Выбор ряда метрик для проверки работоспособности решений будет производиться с учётом возможности дальнейшего сравнения с решениями, описанными в научной литературе (см. п. 4.2).

2.5. Технологический стек

В реализации РС и применении методов обработки естественного языка будет использоваться язык программирования Python – один из наиболее популярных языков, используемых для реализации методов машинного обучения. Данный язык позволяет эффективно реализовывать математическое описание для построения нейронных сетей, что обусловлено довольно большой базой библиотек, необходимых для работы с искусственным интеллектом [9].

Программная реализация будет производиться в среде программирования Google Colaboratory. Это бесплатная среда для разработки и выполнения программного кода в облаке, которая позволяет производить вычисления на языке Python с использованием браузера без установки специальных программ на компьютер. Google Colaboratory основан на Jupyter Notebook – популярном программном обеспечении для написания и запуска кода, которое поддерживает интерактивное программирование. Код можно писать в отдельных ячейках и выполнять их по порядку, просматривая результаты выполнения после блока с кодом.

Далее будут рассмотрены библиотеки, которые необходимы для проектирования системы.

2.5.1. Библиотеки методов обработки естественных языков

Gensim – это библиотека с открытым исходным кодом, написанная Радимом Рехуреком [36], которая используется для тематического моделирования без учителя и обработки естественного языка (NLP). Она предназначена для извлечения семантических тем из документов. Предоставляет доступ к моделям Word2Vec, Doc2Vec, FastText и т. д.

Natural Language Toolkit – это пакет библиотек и программ для символьной и статистической обработки естественного языка. Содержит графические представления и примеры данных. Сопровождается обширной документацией, включая книгу с объяснением основных концепций, стоящих за теми задачами обработки естественного языка, которые можно выполнять с помощью данного пакета [4].

CoreNLP – это библиотека, предоставляющая набор инструментов для обработки текста, основанный на работах Stanford CoreNLP [5]. Она позволяет пользователям получать лингвистические аннотации к тексту, включая границы токенов и предложений, части речи, именованные объекты, числовые значения и значения времени, анализ зависимостей и групп, кореференцию, тональность, атрибуцию цитат и отношения.

2.5.2. Библиотеки предобработки данных

NumPy – это библиотека Python, которую применяют для математических вычислений: начиная с базовых функций и заканчивая линейной алгеброй [24]. Активно используется для анализа данных, машинного обучения и научных вычислений, так как позволяет значительно упростить работу с векторами и матрицами.

Pandas – это программная библиотека на языке Python для обработки и анализа данных [25]. Работа библиотеки строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Pandas предоставляет

специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами.

2.5.3. Библиотеки методов машинного обучения

Scikit-Learn – это библиотека, реализующая методы машинного обучения и написанная на языке Python, в состав которой входят различные алгоритмы, в том числе предназначенные для задач классификации, регрессионного и кластерного анализа данных, включая метод опорных векторов, метод случайного леса, алгоритм усиления градиента, метод k-средних и DBSCAN [26]. Библиотека была разработана для взаимодействия с численными и научными библиотеками языка программирования Python NumPy и SciPy.

SciPy – это библиотека для языка Python, основанная на расширении NumPy и использующаяся для более глубоких и сложных научных вычислений, анализа данных и построения графиков [30]. SciPy в основном написана на Python и частично на языках C, C++ и Fortran, поэтому отличается высокой производительностью и скоростью работы.

2.5.4. Библиотеки визуализации данных

Plotly – это графическая библиотека языка Python, которая позволяет создавать интерактивные графики [27]. Возможны визуализации посредством построения линейных графиков, точечных диаграмм, диаграмм с областями, гистограмм, полос ошибок, коробчатых диаграмм, гистограмм, тепловых карт, подграфиков, многоосных, полярных диаграмм, пузырьковых диаграмм и так далее.

Yellowbrick – это библиотека языка Python, расширяющая API Scikit-Learn, которая упрощает выбор модели и настройку гиперпараметров [35]. Yellowbrick предоставляет возможность визуализации результатов классификации, регрессий, кластеризации и других задач машинного обучения.

WordCloud – это библиотека для языка Python, с помощью которой реализуется метод визуализации данных облако слов, используемый для представления текстовых данных, в котором размер каждого слова указывает на его частоту или важность [33].

2.6. Выводы по второй главе

Для ЯРС была предложена новая модель Pref2Vec – модель векторизации пользовательских предпочтений. Это новый подход к построению РС, при котором решается проблема внедрения контекстной зависимости в ЯРС посредством перехода к более высокому уровню абстракции – формирование рекомендаций будет производиться в зависимости от предпочтений конкретного пользователя. Также данный подход позволяет придерживаться принципов построения гибридной рекомендации, так как анализу подлежат как статические данные (данные объектов), так и динамические данные (пользовательские данные).

Одна из основных особенностей предлагаемой модели – расширение эмбеддингов (векторных представлений объектов), формируемых при помощи модели Doc2Vec, которое позволит отобразить в эмбеддинге семантику взаимодействия пользователей с рассматриваемыми объектами. Дальнейший анализ пространств расширенных эмбеддингов будет являться основным механизмом построения рекомендаций.

При проектировании модели Pref2Vec также было предложено неклассическое применение задачи кластеризации, что позволило решить проблемы: разреженности данных, масштабируемости РС и «серых овец».

Кроме того, были рассмотрены метрики оценки работы РС и выбран технологический стек, с помощью которого будет производиться разработка.

ГЛАВА 3. РЕАЛИЗАЦИЯ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ

3.1. Подготовка исходных данных

При разработке, как и при тестировании, будет использоваться набор датасетов, представленный в научной работе «Item Recommendation on Monotonic Behavior Chains» на 12-ой конференции The ACM Recommender Systems conference (RecSys) [31], что позволит провести как разработку ЯРС с учетом ранее поставленных требований, так и произвести сравнительный анализ после выполнения тестовых запусков ЯРС.

3.1.1. *Обзор исходного набора данных*

Выбранный набор данных содержит информацию, полученную с книжного веб-сервиса «GoodReads» (goodreads.com) – сайта, предоставляющего свободный доступ к обширной базе данных книг, аннотаций и различных обзоров. «Goodreads» предоставляет пользователям довольно широкий спектр возможностей: например, регистрация книг, которые будут включены в общий библиотечный каталог; формирование собственных книжных полок – возможность классификации книг пользователем; отметка прочитанных книг; постановка дифференциальной оценки книге; написание обзора на книгу.

Выбранный набор данных представляет из себя совокупность датасетов, в которых содержатся метаданные книг, анонимизированная информация о взаимодействиях пользователей с книгами и обзоры книг, написанные пользователями. Вышеописанные наборы также разделены на различные жанровые категории: «Children», «Comics & Graphic», «Fantasy & Paranormal», «History & Biography», «Mystery, Thriller & Crime», «Poetry», «Romance» и «Young Adult».

Для непосредственно разработки ЯРС будут использоваться данные о книгах жанра «Children» и данные о взаимодействиях пользователей с

рассматриваемыми книгами, так как эти же данные будут использоваться в тестировании (см. п. 4.1). Набор данных представлен в виде двух GZ-архивов соответственно: *goodreads_books_children.json.gz* и *goodreads_interactions_children.json.gz*.

Далее детально рассмотрим их содержимое и проанализируем, какая информация понадобится в дальнейшей разработке.

Все наборы данных представлены в виде сжатых архивированных JSON-файлов. Обработка файлов такого формата занимает довольно много оперативной памяти платформы Google Colaboratory. Ввиду этого ограничения файлы будут рассматриваться частями – для этого определим функцию *read_json()*, позволяющую считать определённый фрагмент архивированного JSON-файла и обернуть его в объект *pandas.DataFrame*, который будет подвергнут дальнейшей обработке:

```
def read_json(filename, nrows, skiprows=0):
    with gzip.open(filename) as f:
        for i in range(skiprows):
            next(f)
        df = pd.read_json(
            path_or_buf=f,
            lines=True,
            nrows=nrows
        )
    return df
```

Листинг 3.1. Код функции считывания фрагмента архивированного JSON-файла.

Датасет *goodreads_books_children.json.gz* с данными книг содержит в себе информацию о примерно 124 082 зарегистрированных книг веб-сервиса «GoodReads» жанра «Children»: идентификатор на «GoodReads», название, авторы, аннотация, дата издания, издаатель, средний рейтинг, код ISBN, ссылка на страницу книги и т. д. Для демонстрации структуры датасета считаем часть датасета и представим данные книги одной из книг (рис. 3.1).

isbn	0152056580
text_reviews_count	76
series	[]
country_code	US
language_code	
popular_shelves	[{'count': '72', 'name': 'to-read'}, {'count': ...}
asin	
is_ebook	false
average_rating	4.11
kindle_asin	
similar_books	[8137055, 72658, 6289810, 276937, 916684, 1032...]
description	Big Brown Rooster is sick of chicken feed. So ...
format	Paperback
link	https://www.goodreads.com/book/show/1038901.Cook-a-Doodle-Doo!
authors	[{'author_id': '22372', 'role': ''}, {'author_id': '2933269', 'role': ''}]
publisher	HMH Books for Young Readers
num_pages	48
publication_day	1
isbn13	9780152056582
publication_month	8
edition_information	
publication_year	2005
url	https://www.goodreads.com/book/show/1038901.Cook-a-Doodle-Doo!
image_url	https://s.gr-assets.com/assets/nophoto/book/11...
book_id	1038901
ratings_count	329
work_id	2933269
title	Cook-a-Doodle-Doo!
title_without_series	Cook-a-Doodle-Doo!
Name: 98, dtype: object	

Рис. 3.1. Данные одной из книг датасета goodreads_books_children.json.gz.

Затем рассмотрим датасет пользовательских взаимодействий, в котором находятся 10 059 349 записей взаимодействий и который представляет из себя содержимое пользовательских книжных полок, по которым можно сказать: прочитал ли пользователь книгу, находящуюся на полке, и на сколько быстро он это сделал; какую дифференциальную оценку книге он поставил и написал ли он обзор на книгу. Для лучшего понимания приведён пример пользовательской книжной полки с веб-сервера «GoodReads» (рис. 3.2). По ней можно определить вышеупомянутые характеристики, за исключением длительности чтения и наличия обзора. Также представим значение одного из элементов набора данных для демонстрации структуры датасета (рис. 3.3).

cover	title	author	rating	my rating	read	added	
	Airplane Mode: An Irreverent History of Travel	Habib, Shahnaz	4.02	★★★★★	★★★★★	Apr 14, 2024	Dec 29, 2023 view
	A Ladder to the Sky	Boyne, John *	4.21	★★★★★	★★★★★	Jun 10, 2023	Jun 03, 2023 view
	Yellowface	Kuang, R.F. *	3.80	★★★★★	★★★★★	Jun 03, 2023	May 29, 2023 view (with text)
	Babies and Bylines: Parenting on the Move	Aiyar, Pallavi *	4.34	★★★★★	★★★★★	Jun 28, 2022	Jun 23, 2022 view
	The Silent Coup: A History of Joseph, Josy India's Deep State	Josy Joseph	4.39	★★★★★	★★★★★	Sep 2021	Aug 27, 2021 view

Рис. 3.2. Пример пользовательской полки с веб-сервера «GoodReads».

```

user_id          8842281e1d1347389f2ab93d60773d4d
book_id          33282947
review_id        f171a68daa8092d8aea3dccc2e025a81
is_read          False
rating           0
review_text_incomplete
date_added       Fri Feb 10 10:47:53 -0800 2017
date_updated     Fri Feb 10 10:48:21 -0800 2017
read_at
started_at
Name: 1, dtype: object

```

Рис. 3.3. Пример элемента датасета goodreads_interactions_children.json.gz.

3.1.2. Обработка исходного набора данных

В рамках разработки ЯРС основной интерес представляют текстовые данные (аннотации книг) и данные о взаимодействиях пользователей с книгами, при помощи которых можно категоризировать вышеупомянутые текстовые данные. Таким образом необходимо видоизменить ранее представленный датасет, оставив в них только необходимые данные.

Для извлечения книжных аннотаций из датасета *goodreads_books_children.json.gz* сформируем функцию *create_books_subdataset()* (листинг 3.2), позволяющую извлечь из фрагмента датасета все необходимые в рамках разработки данные и сохранить их в CSV-файл. При помощи функции *create_books_subdataset()* получим шесть CSV-файлов, и объединим их в единый датасет *books.csv*, который содержит в себе данные 108 844 книг и имеет структуру, представленную на рис. 3.4. Число рассматриваемых книг уменьшилось, так как у некоторых из представленных книг не было аннотаций.

```
def create_books_subdataset(json_filename, csv_filename, start_index, finish_index):
    books = read_json(
        filename=json_filename,
        nrows=finish_index-start_index,
        skiprows=start_index
    )
    books = books.set_index('book_id')
    books = books.drop(
        columns=['isbn', 'series', 'country_code', 'language_code', 'similar_books',
                 'popular_shelves', 'asin', 'is_ebook', 'kindle_asin', 'format',
                 'publisher', 'publication_day', 'isbn13', 'publication_month',
                 'edition_information', 'link', 'image_url', 'work_id', 'num_pages',
                 'title_without_series', 'text_reviews_count', 'authors',
                 'ratings_count', 'publication_year', 'url', 'average_rating']
    )
    books = books[books['description'] != '']
    books.to_csv(csv_filename)
```

Листинг 3.2. Код функции обработки общего датасета метаданных книг.

	description	title
book_id		
287141	Relates in vigorous prose the tale of Aeneas, ...	The Aeneid for Boys and Girls
6066812	To Kara's astonishment, she discovers that a p... All's Fairy in Love and War (Avalon: Web of Ma...	
89378	In Newbery Medalist Cynthia Rylant's classic b...	Dog Heaven
1698376	WHAT DO YOU DO?\nA hen lays eggs...\nA cow giv...	What Do You Do?
2592648	Ben draws a train that takes him to all sorts ...	It's Funny Where Ben's Train Takes Him

Рис. 3.4. Фрагмент датасета *books.csv*.

Чтобы оставить в датасете *goodreads_interactions_children.json.gz* информацию, характеризующую взаимодействия пользователей с книгами, при помощи которых можно категоризировать книжные аннотации, определим функцию *create_interactions_subdataset()* (листинг 3.3), которая позволит извлечь из фрагмента датасета все необходимые в рамках разработки данные и сохранить их в CSV-файл. В результате обработки подготовленных датасетов были сформированы шесть CSV-файлов, которые далее были объединены в один файл *interactions.csv*. Файл содержит в себе 10 059 349 взаимодействий и имеет структуру, представленную на рис. 3.5.

```
def create_interactions_subdataset(json_filename, csv_filename, start_index,
                                    finish_index):
    interactions = read_json(
        filename=json_filename,
        nrows=finish_index-start_index,
        skiprows=start_index
    )
    interactions = interactions.drop(
        columns=['review_text_incomplete', 'date_added', 'date_updated',
                 'read_at', 'started_at'])
    interactions.to_csv(csv_filename)
```

Листинг 3.3. Код функции обработки датасета пользовательских взаимодействий с книгами.

	user_id	book_id		review_id	is_read	rating
0	8842281e1d1347389f2ab93d60773d4d	10893214	5d0e4e8825c68740703f65a18813fc93	False	0	
1	8842281e1d1347389f2ab93d60773d4d	33282947	f171a68daa8092d8aea3dcc2e025a81	False	0	
2	8842281e1d1347389f2ab93d60773d4d	11387515	2fd3cd1acb30b099c135e358669639da	False	0	
3	8842281e1d1347389f2ab93d60773d4d	24396144	d210e41fcc7e6dc6ae896844a38a024	False	0	
4	8842281e1d1347389f2ab93d60773d4d	20484662	a99f9fa4ec4fd94cc2419c78af2086a8	False	0	

Рис. 3.5. Фрагмент датасета *interactions.csv*.

3.1.3. Формирование корпуса книжных аннотаций

Необходимо произвести препроцессирование книжных аннотаций – форматирование текстовых данных, которое повысит эффективность работы модели Doc2Vec. Для этого воспользуемся функцией *preprocess_documents()* библиотеки Gensim, которая позволяет произвести предварительную

обработку «сырого» текста (удаление знаков препинания, множественных знаков пробела, предлогов, сведение всех символов к нижнему регистру и т. д.), для обработки данных датасета *book.csv*:

```
books = pd.read_csv(
    filepath_or_buffer='/content/drive/MyDrive/БКР/Исходные данные/books.csv',
    index_col='book_id'
).drop(columns=['title'])
df_corpus_annotations = df_corpus_annotations.sample(frac=1)
df_corpus_annotations = books.rename(columns={'description': 'annotation'})
df_corpus_annotations['annotation'] =
df_corpus_annotations['annotation'].astype(str)

df_corpus_annotations['annotation'] =
preprocess_documents(df_corpus_annotations['annotation'])
df_corpus_annotations['annotation'] =
df_corpus_annotations['annotation'].apply(lambda x: ' '.join(x))

df_corpus_annotations.to_csv(
    '/content/drive/MyDrive/БКР/Исходные данные/corpus_annotations.csv'
)
```

Листинг 3.4. Код, формируемый корпус книжных аннотаций

В результате получим датасет *corpus_annotations.csv*:

	annotation
book_id	
287141	relat vigor prose tale aenea legendari ancesto...
6066812	kara astonish discov portal open bedroom close...
89378	newberi medalist cynthia rylant classic bestse...
1698376	hen lai egg cow give milk eleph squirt water b...
2592648	ben draw train take sort wonder place return bed

Рис. 3.6. Фрагмент датасета *corpus_annotations.csv*.

3.1.4. Формирование датасета пользовательских предпочтений

Далее требуется извлечь данные о пользовательских предпочтениях из информации о взаимодействиях пользователей с книгами, полученной в п. 3.1.2. С этой целью сформируем датасет, в котором для каждого пользователя будут храниться идентификаторы книг, разделённые на различные категории в зависимости от вида взаимодействия пользователя с книгой.

Определим категории взаимодействий:

- *read* – пользователь прочитал книгу, находящуюся на его полке;
- *shelved* – пользователь добавил книгу на полку, но не прочитал;
- *rating_0* – пользователь поставил книге оценку «0»;
- *rating_1* – пользователь поставил книге оценку «1»;
- *rating_2* – пользователь поставил книге оценку «2»;
- *rating_3* – пользователь поставил книге оценку «3»;
- *rating_4* – пользователь поставил книге оценку «4»;
- *rating_5* – пользователь поставил книге оценку «5»;

Датасет *interactions.csv* содержит информацию о 542 144 уникальных пользователей. В процессе создания набора данных, который будет использоваться для обучения моделей, ввиду временных ограничений были обработаны данные 100 000 пользователей и сохранены в датасет *user_data.csv*:

		read	shelved	rating_0	rating_1	rating_2	rating_3	rating_4	rating_5
	user_id								
8842281e1d1347389f2ab93d60773d4d		[23310161 18296097 817720 502362 1969280 ...	[10893214 33282947 11387515 24396144 20484662 ...					[23310161 1027760 130580 14366 235324]	[18296097 817720 502362 1969280 17290220 ...
72fb0d0087d28c832f15776b0d936598		[5 3562 3008 78411 8127 2839 91...]	[9673436]				[30119]	[37741 767680]	[5 3562 3008 78411 8127 2839 91...]
ab2923b738ea3082f5f3efcbbfacb218		[240007 370493 30119 5]					[5]	[240007 370493 30119]	
d986f354a045ffb91234e4af4d1b12fd		[9252036 12924291 11737313]	[24396876 8127 3304291 966757 38709 ...]	[9252036 11737313]				[12924291]	
7504b2aee1ecb5b2872d3da381c6c91e		[23302416 3636]						[3636]	[23302416]

Рис. 3.7. Фрагмент датасета *user_data.csv*.

3.2. Формирование эмбеддингов

Модели 2Vec позволяют осуществлять представление текста (или отдельных слов) в виде эмбеддинга (вектора), значение которого зависит от семантики преобразуемого текста (или отдельного слова). Воспользуемся моделью Doc2Vec для векторного представления аннотаций рассматриваемых книг. Полученные эмбеддинги будут использоваться в дальнейших разделах при разработке алгоритма рекомендаций.

3.2.1. Подготовка корпуса модели *Doc2Vec*

Сначала необходимо подготовить корпус модели Doc2Vec, на котором будет происходить обучение и который будет состоять из препроцессированных книжных аннотаций. Создадим корпус из ранее препроцессированных аннотаций датасета *corpus_annotations.csv*. Токены каждой аннотации (набор слов, входящих в аннотацию) обрабатываются в объект *TaggedDocument* библиотеки *Gensim* при помощи функции *create_corpus()*:

```
def create_corpus(text, tag):
    output = []
    for index, row in zip(text.index, text):
        output.append(TaggedDocument(row, [tag + str(index)]))
    return output
```

Листинг 3.5. Код функции создания корпуса модели Doc2Vec.

Приведём пример элемента корпуса модели:

Исходная аннотация книги:

Relates in vigorous prose the tale of Aeneas, the legendary ancestor of Romulus, who escaped from the burning city of Troy and wandered the Mediterranean for years before settling in Italy. Patterned after the Iliad and the Odyssey, the Aeneid was composed as an epic poem by Virgil, to glorify the imperial city of Rome.

Сформированный документ:

TaggedDocument<['relat', 'vigor', 'prose', 'tale', 'aenea', 'legendari', 'ancestor', 'romulu', 'escap', 'burn', 'citi', 'troi', 'wander', 'mediterranean', 'year', 'sett1', 'itali', 'pattern', 'iliad', 'odyssei', 'aeneid', 'compos', 'epic', 'poem', 'virgil', 'glorifi', 'imperi', 'citi', 'rome'], ['annotation_287141']>

Листинг 3.6. Пример элемента корпуса модели Doc2Vec.

3.2.2. Определение и обучение модели Doc2Vec

Воспользуемся модулем *Doc2Vec* библиотеки *Gensim*, определив следующие гиперпараметры:

- *dm* – определяет алгоритм обучения (если *dm* = 1, используется «распределенная память» (PV-DM), в противном случае используется «распределенный пакет слов» (PV-DBOW));
- *vector_size* – размерность формируемых векторов;
- *window* – окно (максимальное расстояние) между текущим и предсказанным словом в предложении;
- *seed* – начальное значение для генератора случайных чисел. Начальные векторы для каждого слова заполняются хэшем конкатенации слова и `str(seed)`;
- *min_count* – частота, при которой слова с меньшей частотой будут игнорироваться;
- *workers* – количество рабочих потоков модели;
- *negative* – если параметр больше 0, будет использоваться негативное сэмплирование. Целое число указывает, сколько негативных примеров должно быть использовано в процессе обучения;
- *dm_concat* – если параметр равен 1, используется конкатенация векторов контекста, а не сумма или среднее значение.

Для определения оптимальных значений гиперпараметров модели обратимся к работе «Distributed Representations of Sentences and Documents» [17]:

- Используется алгоритм Paragraph Vector – Distributed Memory (PV-DM). Векторы документов получаются путём обучения нейронной сети синтетической задаче прогнозирования центрального слова на основе среднего значения векторов слов контекста и вектора слов всего документа;

- Для более эффективного обучения модели используется негативное семплирование (Negative Sampling): модели предоставляются слова, которые не являются контекстными соседями;
- Оптимальный размер вектора для Doc2Vec зависит от конкретного приложения и размера набора данных. Как правило, больший размер вектора позволяет захватывать больше семантической информации, но также может потребовать больше вычислительных ресурсов и времени на обучение. Общий диапазон размера вектора составляет от 100 до 300;
- Размер оптимального текстового окна равен от 5 до 12.

Далее определим модель, используя значения гиперпараметров, представленные на листинге 3.7, и произведём её обучение (листинг 3.8).

Используемые гиперпараметры:

1) Используемый алгоритм обучения	- PV-DM
2) Размерность формируемых векторов	- 200
3) Размер текстового окна	- 10
4) Начальное значение генератора случайных чисел	- 83
5) Наименьшая частота рассматриваемых слов	- 2
6) Количество рабочих потоков модели	- 1
7) Количество негативных примеров	- 7
8) Принцип формирования векторного пространства векторов	- конкатенация контекстных векторов

Листинг 3.7. Значения гиперпараметров модели Doc2Vec.

```
filename_d2v = '/content/drive/MyDrive/BKP/children/children_model_d2v.d2v'

if os.path.isfile(filename_d2v):
    # Загрузка предобученной модели
    model_d2v = Doc2Vec.load(filename_d2v)
else:
    # Определение и обучение модели
    model_d2v = Doc2Vec(
        dm=DM, vector_size=VECTOR_SIZE, window=WINDOW_SIZE,
        seed=SEED, min_count=MIN_COUNT, workers=WORKERS,
        negative=7, dm_concat=DM_CONCAT
    )
    model_d2v.build_vocab(corpus_annotations)
    model_d2v.train(corpus_annotations, total_examples=model_d2v.corpus_count,
epoches=EPOCHS)
    model_d2v.save(filename_d2v)
```

Листинг 3.8. Код определения и обучения модели Doc2Vec.

3.2.3. Анализ работы модели Doc2Vec

Далее произведём некоторый анализ эмбеддингов, получаемых моделью Doc2Vec. Сперва произведём выгрузку эмбеддингов, полученных при обучении на корпусе книжных аннотаций, из модели в массив. Приведём пример эмбеддинга:

Исходная аннотация книги:

Relates in vigorous prose the tale of Aeneas, the legendary ancestor of Romulus, who escaped from the burning city of Troy and wandered the Mediterranean for years before settling in Italy. Patterned after the Iliad and the Odyssey, the Aeneid was composed as an epic poem by Virgil, to glorify the imperial city of Rome.

Полученный эмбеддинг:

```
[-8.29084963e-03 -1.76829763e-03 4.83798375e-03 9.47105791e-03 -6.07674429e-03
-3.08325374e-03 1.20961736e-03 -6.06394792e-03 1.12760952e-02 7.09235296e-03
7.12088170e-03 -3.76315508e-03 7.93726742e-03 1.17525281e-02 -5.50322467e-03
-5.58801647e-03 3.04289279e-05 -6.90393941e-03 1.96768017e-03 7.72711821e-04
2.39320658e-03 3.88640558e-07 1.58490390e-02 -1.00999209e-03 5.07306121e-03
-4.96758753e-03 2.51308456e-03 6.79001026e-03 3.63722001e-03 -2.51871464e-03
9.91912190e-04 1.39052151e-02 7.81932613e-04 1.62359315e-03 -1.54879235e-03
5.88215771e-04 -5.75025845e-03 -3.52314184e-03 8.04545730e-03 3.88107891e-03
-6.12365082e-03 -3.63941625e-04 -1.07077640e-02 -5.89995761e-05 3.05010332e-03
7.75952125e-03 -7.93504808e-03 -3.08488263e-03 2.72575719e-03 -5.94739104e-03
1.02308551e-02 -1.07663525e-02 -5.15445555e-03 3.80896032e-03 1.25097204e-02
7.23522296e-03 -3.77254630e-03 -8.35438073e-03 8.45283386e-04 -2.37995737e-05
-8.20609741e-03 8.42885021e-03 -8.17498751e-03 -8.51257006e-04 -8.88068508e-03
-7.51588074e-03 3.47555079e-03 -5.25793945e-03 -2.31778467e-04 -2.13756459e-03
-3.38105205e-03 -8.03331472e-03 -5.12535218e-03 -1.87078386e-03 9.11569106e-04
2.19015608e-04 -7.22676190e-03 -8.25251080e-03 1.77503389e-03 8.42185691e-04
-5.43190446e-03 8.42987560e-03 -2.21100077e-03 6.45868015e-03 1.79814803e-03
7.15137983e-04 -2.20710807e-03 -8.99923127e-03 -1.25898863e-03 1.19127529e-02
6.90246327e-03 5.49275661e-04 1.31622963e-02 -9.27368272e-03 1.02363862e-02
-7.60573021e-04 -2.96532898e-03 -9.14416555e-03 5.62821049e-03 2.25622114e-03
1.51841657e-03 -1.01162679e-02 -1.86934776e-03 2.44013802e-03 8.11415911e-03
4.49731573e-03 3.14098946e-03 3.75715620e-03 5.94362663e-03 -1.24331214e-03
5.47736650e-03 1.92408322e-03 -9.22188908e-03 1.07197920e-02 -1.50248816e-03
-3.25381348e-04 -3.76069592e-03 5.99786872e-03 -9.70905833e-03 1.08834554e-03
8.94819666e-03 -3.19623016e-03 7.84983765e-03 -7.76069285e-03 -8.64670333e-03
-8.09455058e-04 -5.07477112e-03 -1.19027216e-02 2.24162638e-03 1.56576615e-02
-4.77067370e-04 1.83327633e-04 2.74404022e-03 -2.05204077e-03 1.54312036e-03
-9.80139896e-03 2.30810512e-03 3.25410947e-04 -4.20462899e-03 -1.35875260e-03
-2.79270834e-03 3.59509489e-03 2.40283110e-03 3.16460710e-03 -5.11239003e-03
-5.67643787e-04 7.95330503e-04 -1.62383716e-03 7.44687067e-03 1.36612169e-02
-1.72220112e-03 -8.68490338e-03 2.61086301e-04 -4.94189002e-03 -1.11998469e-02
-5.99700212e-03 -4.62409202e-03 -5.49388304e-03 -5.04117412e-03 8.91116541e-03
-5.84687665e-03 6.19017938e-03 1.57528452e-03 2.29038182e-03 -2.02904543e-04
1.46000786e-02 4.50403756e-03 -9.86891985e-03 -2.71090376e-03 1.97621225e-03
2.33797752e-03 6.16558536e-04 1.34257060e-02 -3.18543334e-03 -6.92707812e-03
-7.14545324e-03 -1.34094618e-03 -7.63477106e-03 1.18060587e-02 1.06132694e-03
3.37355467e-03 1.23215234e-03 1.25595171e-03 -1.39310036e-03 -2.88435607e-03
1.12001039e-02 -6.99977856e-04 1.42613612e-03 4.97897156e-03 2.79830839e-03
5.18903881e-03 -9.87178739e-03 -1.10693118e-02 -1.91555906e-03 -6.50996668e-03
-1.33904291e-03 7.24752760e-03 -8.73855595e-03 -8.40388797e-03 -2.57383636e-03]
```

Листинг 3.9. Пример элемента корпуса модели Doc2Vec.

Чтобы определить наличие семантического смысла в рассчитанных эмбеддингах, произведём их кластеризацию. Таким образом появляется возможность выделить некоторые смысловые группы, которые будут выражаться в принадлежности тому или иному кластеру. Для решения задачи

кластеризации прибегнем к алгоритму K-Means библиотеки *Scikit-learn*. Алгоритм K-Means группирует данные, пытаясь разделить выборки на n групп с равной дисперсией, минимизируя критерий, известный как инерция или сумма квадратов внутри кластера.

Для определения модели K-Means воспользуемся модулем *sklearn.cluster.KMeans*, определив следующие гиперпараметры:

- *init* – метод инициализации модели (*k-means++* - выбирает исходные центроиды (центры тяжести кластеров) кластера с использованием выборки на основе эмпирического распределения вероятностей вклада точек в общую инерцию);
- *n_init* – количество запусков алгоритма K-means с разными начальными значениями центроида. Конечный результат — лучший результат последовательных запусков *n_init* с точки зрения инерции;
- *max_iter* – максимальное количество итераций алгоритма K-means за один прогон;
- *random_state* – определяет генерацию случайных чисел для инициализации центроида. Используется тип *int* для детерминированности случайности.

Этот алгоритм требует указания количества кластеров. Ввиду этого ограничения необходимо определить оптимальное количество кластеров, на которые будут разделены исходные данные. Для этого будет использоваться метод локтя, который помогает по рассматриваемым данным выбрать оптимальное количество кластеров, снабжая модель K-Means диапазоном значений k (количество кластеров) [35]. Для каждого значения k вычисляется значение искажения — сумма квадратов расстояний от каждой точки до назначенного ей центра. В результате получается график искажения, который далее подлежит исследованию. Если график напоминает руку, то «локоть» (точка перегиба кривой) является хорошим показателем того, что базовая модель лучше всего подходит для этой точки. Для реализации метода локтя

воспользуемся модулем *KElbowVisualizer* библиотеки *Yellowbrick*. В визуализаторе «локоть» отмечен пунктирной линией (рис. 3.8).

Опираясь на результаты алгоритма метода локтя (рис. 3.8), можно сказать, что оптимальное количество кластеров равно 14. Кластеризуем эмбеддинги аннотаций с полученным значением кластеров при помощи алгоритма K-Means и перенесём полученные метки кластеров на датасет с корпусом книжных аннотаций (рис. 3.9).

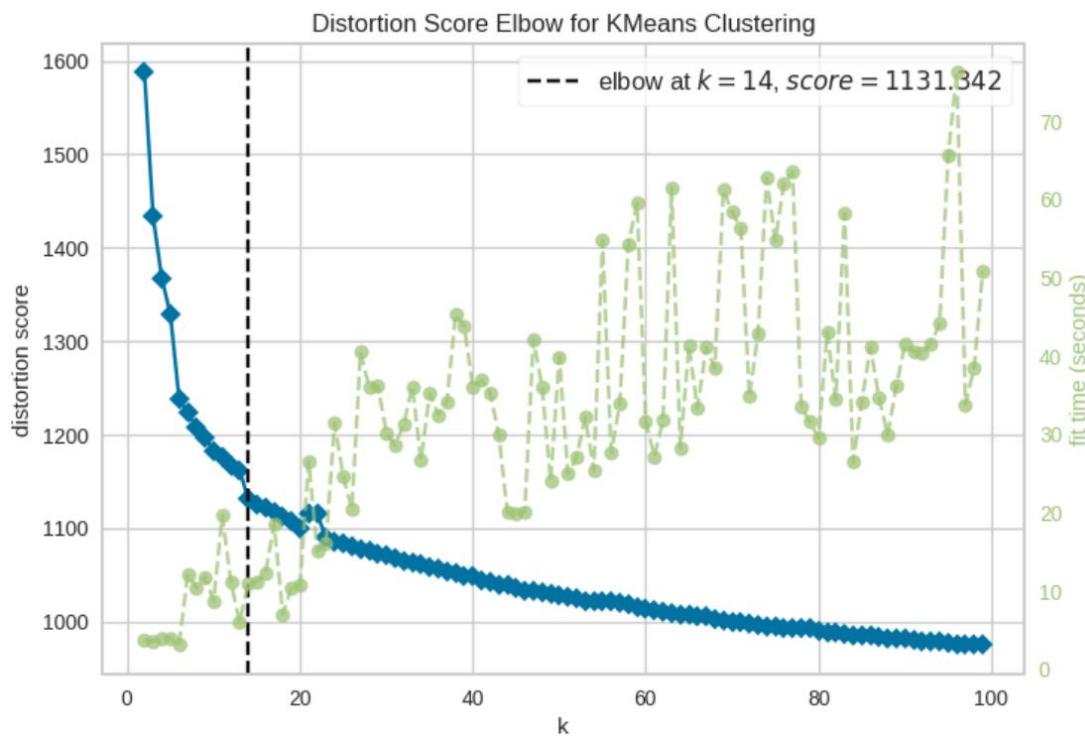


Рис. 3.8. Результат работы метода локтя.

	annotation	cluster_label
book_id		
287141	relat vigor prose tale aenea legendari ancesto...	8
6066812	kara astonish discov portal open bedroom close...	8
89378	newberi medalist cynthia rylant classic bestse...	2
1698376	hen lai egg cow give milk eleph squirt water b...	8
2592648	ben draw train take sort wonder place return bed	6

Рис. 3.9. Фрагмент датасета corpus_annotations_clustered.csv.

Визуализируем результаты работы в виде гистограммы распределения аннотаций по кластерам:

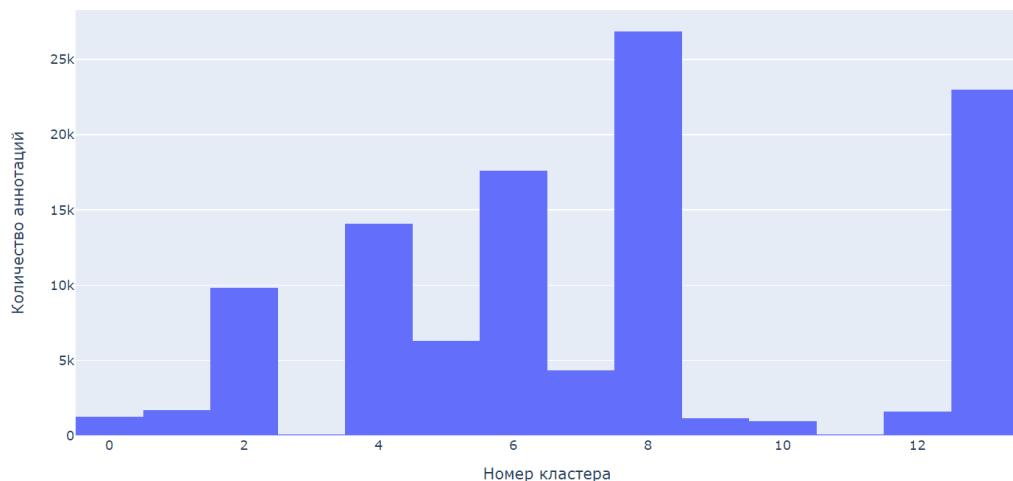


Рис. 3.10. Гистограмма распределения книжных аннотаций по кластерам.

Также представим облака слов аннотаций для каждого кластера. Для этого определим функцию `cluster_wordclouds()`, в которой будет производиться расчёт облаков слов, и представим результаты её работы:

```
def cluster_wordclouds(cluster_amount):
    """
    Функция расчёта облаков слов аннотаций для каждого кластера данных.

    Аргументы:
    - cluster_amount (int) - количество рассматриваемых кластеров.
    """

    fig = make_subplots(
        rows=int(cluster_amount/3 + 1), cols=3,
        subplot_titles=[['Кластер №{}'.format(i) for i in range(cluster_amount)])
    )

    for cluster_num in range(cluster_amount):
        row_index = int(cluster_num / 3 + 1)
        col_index = int(cluster_num % 3 + 1)
        text = '\n'.join(df_corpus_annotations[df_corpus_annotations['cluster_label'] ==
cluster_num]['annotation'].values)
        wordcloud = WordCloud(
            width=500, height=500, min_font_size=10, background_color="white"
        ).generate(text)
        fig.add_trace(px.imshow(wordcloud).data[0], row=row_index, col=col_index)
        fig.update_xaxes(visible=False, row=row_index, col=col_index)
        fig.update_yaxes(visible=False, row=row_index, col=col_index)

    fig.update_layout(
        height=1300, width=850,
        title=dict(text='Облака слов кластеризованных аннотаций',
        font=dict(size=18))
    )
    fig.show()
```

Листинг 3.10. Код функции расчёта облаков слов для каждого кластера аннотаций.

В результате вызова функции `cluster_wordclouds()` получим следующие облака слов некоторых кластеров:



Рис. 3.11. Облака слов книжных аннотаций для кластеров №0 – 2.



Рис. 3.12. Облака слов книжных аннотаций для кластеров №3 – 5.

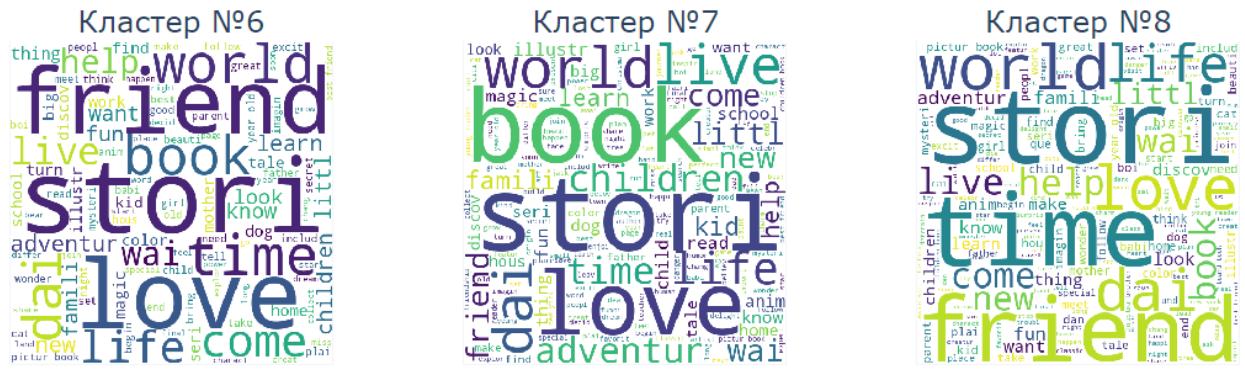


Рис. 3.13. Облака слов книжных аннотаций для кластеров №6 – 8.

По полученным результатам (рис. 3.11 – 3.13) видно, что разбиение данных на кластеры даёт вполне осмысленные группы аннотаций, что, в свою очередь, говорит о том, что векторное представление текстовых данных при помощи модели Doc2Vec учитывает семантическое значение документов (аннотаций). Таким образом, удалось удостовериться в валидности формирования эмбеддингов.

3.3. Разработка класса пользовательского пространства эмбеддингов

Далее воспользуемся ранее обученной моделью Doc2Vec для обработки пользовательских данных веб-сервиса «GoodReads» - рассмотрим применение модели Doc2Vec в разработке ЯРС.

В рамках разработки будем придерживаться следующего подхода: книги, с которыми взаимодействовал пользователь, будут представлены в виде эмбеддингов, расположение в пространстве которых будет зависеть от вида взаимодействия. То есть каждая книга будет преобразована в эмбеддинг при помощи обработки её аннотации, далее будет производиться его расширение в зависимости от вида взаимодействия. Таким образом, для каждого пользователя будет представлено своё собственное пользовательское пространство эмбеддингов – *User Embeddings Space (UES)*.

Стоит отметить, что при определении центров кластеров стоит использовать алгоритм K-Medoids, в котором центр кластера является одним из элементов кластера (является медоидом). Данный подход позволяет уменьшить влияние возможных выбросов на расположение центров кластеров.

Достаточно важное средство анализа кластеров взаимодействия – их визуализация. Но, учитывая высокую размерность эмбеддингов, стоит заранее произвести уменьшение их размерности. Для решения поставленной задачи воспользуемся алгоритмом *t-SNE* – инструментом для визуализации многомерных данных. Он преобразует сходство между точками данных в совместные вероятности и пытается минимизировать расхождение Кульбака-Лейблера между совместными вероятностями низкоразмерного встраивания и многомерных данных.

3.3.1. Расширение исходных эмбеддингов

Формирование «тона» рассматриваемых книг будет производиться добавлением ещё одного измерения к эмбеддингам, получаемым при помощи модели Doc2Vec.

Проанализировав множество эмбеддингов, формируемых при помощи метода *infer_vector()* модели Doc2Vec, и научную работу, в которой предлагается данная модель [17], можем прийти к выводу о том, что значения компонент эмбеддингов являются нормализованными, то есть варьируются от -1 до 1. Таким образом, при добавлении дополнительной размерности, учитывающей характер взаимодействия пользователя, следует взять значения, лежащие в том же диапазоне.

Учитывая ранее определённые виды пользовательских взаимодействий, получим следующие значения добавляемой компоненты эмбеддинга:

Характеристики формирования расширяющей компоненты эмбеддинга:

- Левая граница диапазона - -1
- Правая граница диапазона - 1
- Количество реакций пользователя - 8
- Шаг формирования значения - 0.2857142857142857

Значения расширяющей компоненты эмбеддинга:

- 1) Категория «read» - -1.0
- 2) Категория «shelved» - -0.7142857142857143
- 3) Категория «rating_0» - -0.4285714285714286
- 4) Категория «rating_1» - -0.1428571428571429
- 5) Категория «rating_2» - 0.1428571428571428
- 6) Категория «rating_3» - 0.4285714285714284
- 7) Категория «rating_4» - 0.7142857142857142
- 8) Категория «rating_5» - 1.0
- 9) Значение по умолчанию - nan

Листинг 3.11. Значения расширяющей компоненты эмбеддинга.

3.3.2. Разработка класса *UES*

Перейдём к разработке класса *UES* (*User Embedding Space*), который будет описывать пространство эмбеддингов конкретного пользователя. Он будет содержать в себе набор эмбеддингов, учитывающих, как и объект, так и вид взаимодействия с ним, и набор средств анализа рассчитанного пространства.

На вход будет подаваться серия библиотеки *Pandas* датасета *user_data.csv*, в которой содержится информация о взаимодействиях пользователя с книгами в виде категоризированного набора идентификаторов книг (фрагмент датасета приведён на рис. 3.7). Также предполагается использование корпуса препроцессированных книжных аннотаций *corpus_annotations.csv* (рис. 3.6).

Код разработанного класса *UES*, реализующего пользовательское пространство эмбеддингов, представлен в приложении 1. Класс содержит следующие методы:

- *`__init__()`* – метод инициализации класса, в котором производится обращение к методам, формирующими собственно пространство эмбеддингов;
- *`__construct_embedding_space()`* – метод, при помощи которого происходит расчёт эмбеддингов: формирование эмбеддинга с помощью модели Doc2Vec и его последующее расширение;
- *`__get_interaction_value()`* – метод, который предоставляет значение дополнительной компоненты эмбеддинга в зависимости от вида взаимодействия пользователя с книгой;
- *`__compute_centers()`* – метод, позволяющий получить центры (медоиды) кластеров взаимодействий;
- *`retrieve_centers()`* – метод, который возвращает ранее вычисленные центры кластеров взаимодействий в виде фрагмента датасета;
- *`show_tsne()`* – метод, при помощи которого происходит визуализация пространства эмбеддингов пользователя сниженной размерности при помощи алгоритма T-SNE;
- *`show_wordclouds()`* – метод, визуализирующий кластеры взаимодействий посредством формирования облака слов.

Все рассчитанные эмбеддинги пространства хранятся в датасете, обращение к которому происходит через *self.df*.

3.3.3. Анализ работы класса UES

Далее произведём анализ работы разработанного класса, рассчитав пространство эмбеддингов для пользователя с идентификатором `8842281e1d1347389f2ab93d60773d4d`. Приведём фрагмент датасета `user_data.csv`, который описывает предпочтения пользователя и который будет обработан моделью далее:

	read	shelved	rating_0	rating_1	rating_2	rating_3	rating_4	rating_5
user_id								
8842281e1d1347389f2ab93d60773d4d	[23310161 18296097 817720 502362 1969280 ...	[10893214 33282947 11387515 24396144 20484662 ...					[23310161 1027760 130580 14366 235324]	[18296097 817720 502362 1969280 17290220 ...

Рис. 3.14. Данные пользователя с идентификатором `8842281e1d1347389f2ab93d60773d4d`.

В результате создания экземпляра класса `UES` рассчитывается пользовательское пространство эмбеддингов, данные которого хранятся в датасете `df`.

	embedding	interaction	type	tokens
id				
shelved_10893214	[0.0068172430619597435, -0.0037543573416769505...	shelved	object	cavendish home boi girl definit learn lesson a...
shelved_33282947	[0.006353838834911585, 0.0017659572185948491, ...	shelved	object	year old alex petroski love space rocket mom b...
shelved_11387515	[0.0026427125558257103, 0.00624890998005867, 0...	shelved	object	won look like think probabl wor august auggi p...
shelved_24396144	[-0.01435781642794609, -0.017460936680436134, ...	shelved	object	pierr maze detect new case stolen maze stone p...
shelved_20484662	[0.0008134141098707914, -0.01196880079805851, ...	shelved	object	juni jone ivi bean come lovabl energet littl s...

Рис. 3.15. Фрагмент датасета пользовательского пространства эмбеддингов.

Вслед за этим произведём выгрузку всех центров кластеров взаимодействий и получим следующий датасет:

	embedding	interaction	type	tokens
id				
shelved_809849	[-0.0010988630820065737, 0.0035453320015221834...	shelved	center	child write zoo pet zoo send seri unsuit pet r...
read_42407	[0.0005407451535575092, 0.002594402525573969, ...	read	center	babar establish celestevil beauti happi citi q...
rating_4_235324	[0.0014150608330965042, 0.8569533494883217e-05...	rating_4	center	littl sophi snatch bed dead night bfg fear wor...
rating_5_42407	[0.00014276373258326203, 0.002246614545583725, ...	rating_5	center	babar establish celestevil beauti happi citi q...

Рис. 3.16. Датасет центров кластеров взаимодействий.

С целью визуализации полученного пространства эмбеддингов с помощью алгоритма t-SNE воспользуемся методом `show_tsne()`. При помощи данного метода можно получить два варианта визуализации: с учётом расширения исходных эмбеддингов и без. В результате выполнения метода выводятся два графика: трёхмерная визуализация эмбеддингов (точек) и матрица диаграмм рассеяния (рис. 3.17-3.20).

Визуализация UES при помощи метода T-SNE (с учётом взаимодействий)



Рис. 3.17. Визуализация пользовательского пространства эмбеддингов при помощи метода t-SNE с учётом видов взаимодействий.

Матрица диаграмм рассеяния

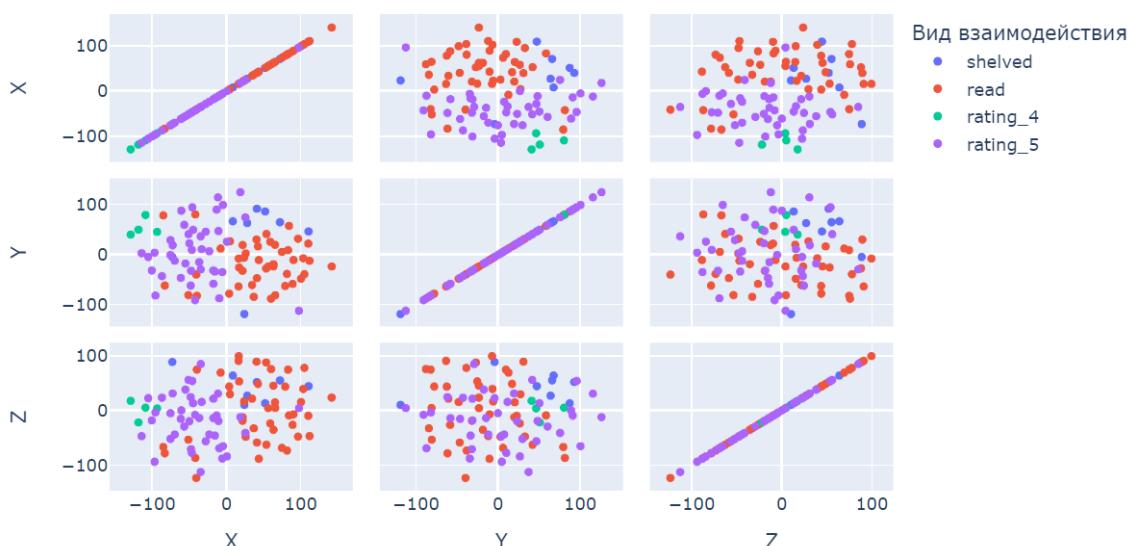


Рис. 3.18. Матрица диаграмм рассеяния пользовательского пространства эмбеддингов при помощи метода t-SNE с учётом видов взаимодействий.

Визуализация UES при помощи метода T-SNE (без учёта взаимодействий)

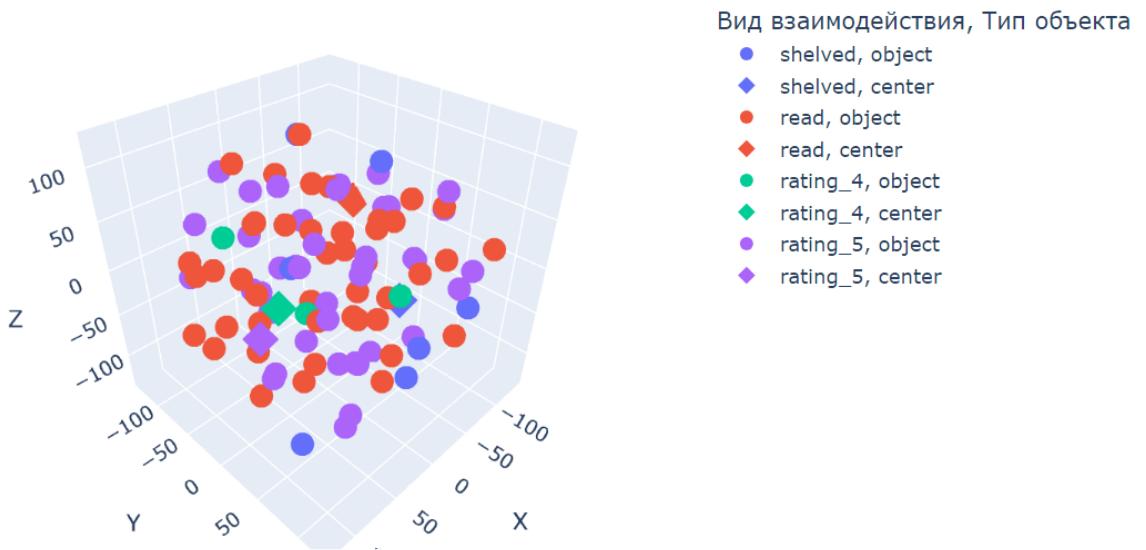


Рис. 3.19. Визуализация пользовательского пространства эмбеддингов при помощи метода t-SNE без учёта видов взаимодействий.

Матрица диаграмм рассеяния

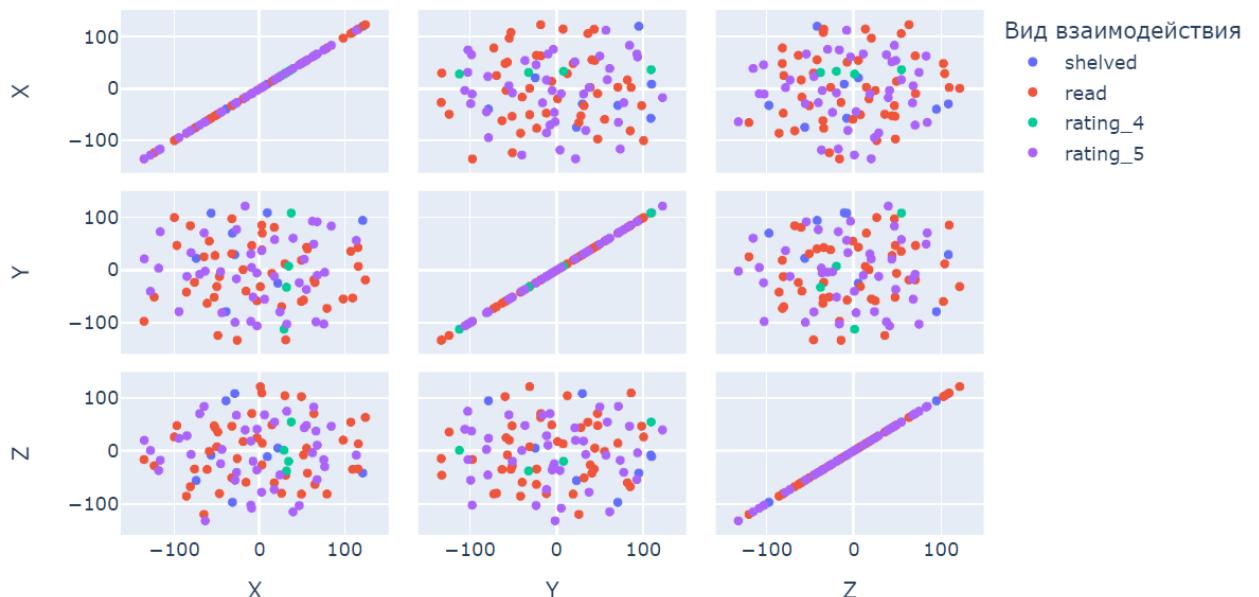


Рис. 3.20. Матрица диаграмм рассеяния пользовательского пространства эмбеддингов при помощи метода t-SNE без учёта видов взаимодействий.

Также выведем облака слов книжных аннотаций для каждого вида взаимодействия, на которых размер слова пропорционален частоте его появления в каждой из категорий (чем больше размер слова, тем больше его частота):

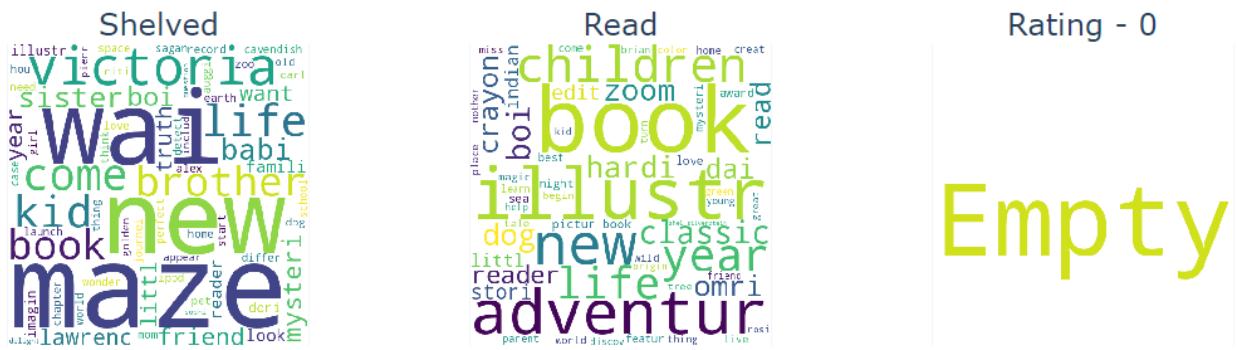


Рис. 3.21. Облака слов книжных аннотаций категорий shelved, read и rating_0.



Рис. 3.22. Облака слов книжных аннотаций категорий rating_1, rating_2 и rating_3.

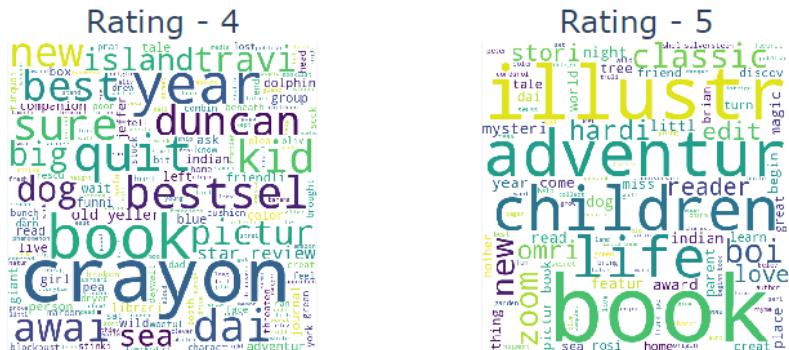


Рис. 3.23. Облака слов книжных аннотаций категорий rating 4 и rating 5.

По полученным результатам можно сказать, что разработанный класс UES позволяет произвести расчёт пользовательского пространства эмбеддингов при помощи предобученной в р. 3.2.2 модели Doc2Vec. Класс позволяет найти центры кластеров взаимодействий посредством алгоритма K-Medoids, которые далее будут использованы в качестве основный характеристик предпочтений пользователя. Также предоставляется возможность визуализации пространства и построение облаков слов.

3.4. Разработка модели векторизации пользовательских предпочтений

При помощи класса UES появляется возможность векторизации всех взаимодействий пользователя с объектами – построение пользовательского пространства эмбеддингов. Рассчитав пространства эмбеддингов для ряда пользователей, открывается перспектива исследования схожих пространств и построение рекомендаций на основе полученных данных.

3.4.1. Исследование пользовательских пространств эмбеддингов

Пользовательское пространство эмбеддингов представляет из себя набор векторов, характеризующих семантическое значение объектов и вид взаимодействия пользователя с ними. Эмбеддинги объединены в кластеры взаимодействий: shelved, read, rating_0, rating_1, rating_2, rating_3, rating_4 и rating_5. Каждый кластер взаимодействия имеет центр – элемент, описывающий «усреднённое» значение всех эмбеддингов кластера и «усреднённые» пользовательские предпочтения в рассматриваемом виде взаимодействия. Каждое рассчитанное пространство представляет собой векторизованные пользовательские предпочтения. В рамках разработки ЯРС можно использовать полученные данные как основное средство построения рекомендаций – можно рекомендовать пользователям объекты на основе данных о схожих пользовательских пространствах. На рис. 3.24 приведён пример рассчитываемого пространства для пользователя с идентификатором 7b2e5fe9fd353fecf3eeebb4850b88d3, визуализированного при помощи алгоритма t-SNE.

Таким образом необходимо определить способ поиска схожих пользовательских пространств эмбеддингов (схожих пользовательских предпочтений). Ранее было сказано о том, что каждое пользовательское пространство имеет набор центров кластеров взаимодействий (см. п. 3.3). Тот

факт, что набор центров описывает основные «направления» интересов пользователя в каждом из видов взаимодействий, позволяет ограничиться в дальнейших расчётах только ими, то есть исключить из расчётов все остальные точки (пример получаемого пространства приведён на рис. 3.25). Следовательно поиск схожих пользовательских пространств будет происходить посредством сравнения центров кластеров взаимодействий. Данный подход позволяет избежать большого количества вычислений и в тоже время обеспечивает достаточно высокий уровень точности поиска схожих пространств.

Визуализация UES при помощи метода T-SNE
(с учётом взаимодействий)

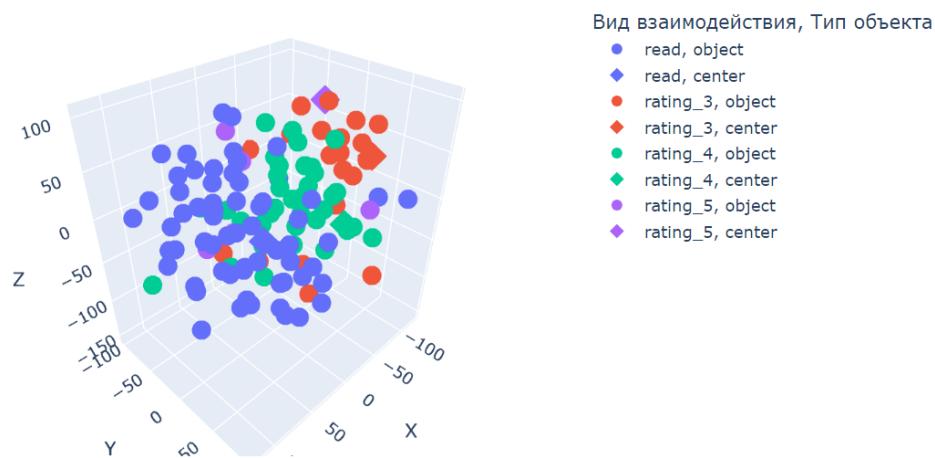


Рис. 3.24. Визуализация всех элементов пользовательского пространства эмбеддингов при помощи метода t-SNE с учётом видов взаимодействий.

Визуализация UES при помощи метода T-SNE
(с учётом взаимодействий)

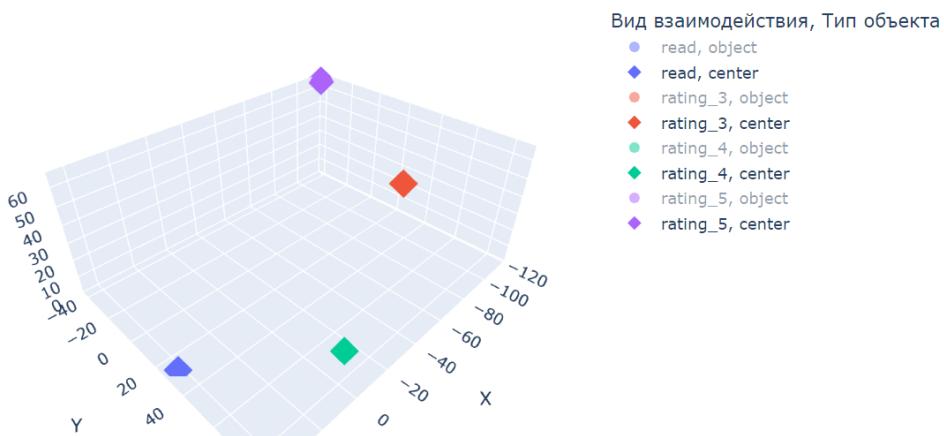


Рис. 3.25. Визуализация центров пользовательского пространства эмбеддингов при помощи метода t-SNE с учётом видов взаимодействий.

Для сравнения центров кластеров взаимодействий (векторов) следует прибегнуть к вычислению косинусного сходства [17] между рассматриваемыми центрами. Косинусное сходство – это мера сходства между двумя векторами предгильбертового пространства (вещественное или комплексное линейное пространство с определённым на нём скалярным произведением), которая используется для измерения косинуса угла между ними [20]. Для двух векторов А и В косинусное сходство $\cos(\theta)$ может быть представлено следующим образом:

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

Далее, рассчитав данные о косинусных сходствах между центрами целевого пространства эмбеддингов и всеми остальными центрами других пространств, получим некоторый список ближайших пользовательских пространств. Способ определения этого списка будет рассмотрен в п. 3.4.2.

Содержимое каждого пространства из списка – потенциальные объекты рекомендации, так как это объекты, с которыми взаимодействовали пользователи со схожими предпочтениями по отношению к целевому пользователю. В силу этого для построения рекомендации требуется ранжировать содержимое пользовательских пространств – для решения этой задачи так же воспользуемся косинусным сходством. Эмбеддинги объектов пользовательских пространств будут сравниваться с эмбеддингами объектов, с которыми взаимодействовал целевой пользователь. На основе рассчитанных сходств будет сформирован список ближайших объектов, его же и будем интерпретировать как список рекомендаций.

3.4.2. Разработка модели Pref2Vec

Перейдём к разработке модели Pref2Vec, которая будет реализовывать исследование пользовательских пространств эмбеддингов, описанное в предыдущем пункте. Код разработанной модели приведён в приложении 2.

В Pref2Vec определены поля *self.depth* (глубина поиска ближайших пользовательских пространств эмбеддингов в рамках одного вида взаимодействия), *self.corpus* (датасет, в котором хранится корпус центров кластеров взаимодействий рассматриваемых пользовательских пространств), *self.df_most_similar_users* (датасет, который используется для расчёта ближайших пользовательских пространств) и *self.df_embeddings_target* (датасет, используемый в построении рекомендаций). Также определены следующие методы:

- *__init__()* – метод инициализации класса, в котором производится определение полей *self.depth* и *self.corpus*;
- *build_corpus()* – метод, при помощи которого производится построение корпуса пользовательских предпочтений. Для каждого пользователя рассчитывается пространство эмбеддингов (UES), далее полученные центры кластеров взаимодействий сохраняются в корпус модели;
- *__save_corpus()* – метод, который сохраняет корпус модели в CSV-файл;
- *load_corpus()* – метод, позволяющий загрузить корпус модели из CSV-файла;
- *most_similar_users()* – метод, при помощи которого происходит поиск ближайших *topn* пользовательских пространств к целевому пространству. Поиск осуществляется с помощью расчёта косинусных сходств между центрами кластеров категорий целевого пространства и соответствующими центрами пространств корпуса модели *self.corpus*. Для каждой из категорий происходит расчёт ближайших *self.depth* пользователей. Далее выводятся самые близкие пространства к целевому;
- *__most_similar_users_by_interaction()* – метод, осуществляющий расчёт косинусных сходств между искомым центром категории и всеми остальными центрами категории корпуса *self.corpus*. Сходство вычисляется при помощи модуля *metrics.pairwise.cosine_similarity* библиотеки *sklearn*. Для увеличения скорости вычислений промежуточные

результаты сохраняются и обрабатываются при помощи библиотеки *pandas*;

- *recommend_for_user()* – метод, при помощи которого происходит расчёт *topn* рекомендаций для целевого пользователя. Сначала происходит поиск 10 схожих пользовательских пространств к целевому. Далее рассматриваются все объекты схожих пространств на предмет косинусного сходства с объектами целевого пространства – происходит расчёт сходства между рассматриваемым объектом и всеми целевыми, наибольшее сходство сохраняется. После происходит сортировка объектов по полученным сходствам и вывод *topn* ближайших объектов;
- *__get_nearest_distance()* – метод, который позволяет найти ближайшее косинусное расстояние к одному из целевых объектов датасета *self.df_embeddings_target*;
- *__get_element_or_nan()* – метод, который возвращает элемент массива, если его адрес находится в пределах массива. В ином случае возвращается элемент *pr.nan*;

Для работы модели Pref2Vec необходимо произвести построение корпуса *self.corpus* после её инициализации. Построение реализуется при помощи функции *build_corpus()*, в аргументы которой передаётся датасет *user_data.csv* (или его фрагмент) с данными о взаимодействиях пользователя с объектами, процесс подготовки которого приведён в п. 3.1.4. На основе переданного датасета формируются пользовательские пространства эмбеддингов для каждого пользователя, центры кластеров взаимодействий сохраняются в датасет *self.corpus*. Для наглядности на рис. 3.26 приведён фрагмент корпуса модели. По нему видно, что для всех кластеров взаимодействий, если в них есть объекты, происходит расчёт их центров при помощи метода K-Medoids. В ином случае центр кластера равняется значению *pr.nan*.

		center_read	center_shelved	center_rating_0	center_rating_1	center_rating_2	center_rating_3	center_rating_4
	user_id							
8842281e1d1347389f2ab93d60773d4d		[-0.00265882583, 0.0017797444, -0.00103694596,...	[0.0390687212, -0.0572869144, -0.0117275408, 0...]	NaN	NaN	NaN	NaN	NaN
d986f354a045ffb91234e4af4d1b12fd		[-0.0434201919, -0.0175371468, -0.0312601067,...	[-0.00291965134, 0.000432312285, 0.00143234897...]	[-0.04883755, -0.0241887048, -0.0431392603, -0...]	NaN	NaN	NaN	NaN
7504b2aee1ecb5b2872d3da381c6c91e		[-0.0128626153, -0.0352191627, -0.0125934482,...]	NaN	NaN	NaN	NaN	NaN	[0.0315045267, -0.0158717521, 0.00558423856, ...]
f8a89075dc6de14857561522e729f82c		[0.000429262844, 0.00850135926, 0.00678327354,...]	[0.00335145555, 0.0024496296, 0.000192135238, ...]	NaN	NaN	NaN	[0.00685263891, -0.00243242714, -0.00927158445...]	[0.0112979729, 0.0265071075, -0.0577949062, -0...]
47d4a49ae9ee68ae78d891b3a259b695		[-0.00216132333, 0.00797456037, 0.0041134879,...]	NaN	NaN	NaN	NaN	[-0.0013365848, 0.000884450914, -0.00214958843...]	[-0.00597673748, -0.016614547, 0.00689799013, ...]

Рис. 3.26. Фрагмент корпуса модели Pref2Vec.

После детально рассмотрим реализацию алгоритма поиска ближайших пользователей:

- 1) При помощи метода `most_similar_users()` определяется целевое пользовательское пространство – вычисляется UES;
- 2) Для каждого вида взаимодействия посредством метода `_most_similar_users_by_interaction()` вычисляются косинусные сходства между центром кластера текущего вида взаимодействия целевого пространства и центрами кластеров текущего вида взаимодействия всех остальных пространств. Все пространства корпуса ранжируются по рассчитанным значениям сходства, в дальнейших расчётах будут использоваться первые N элементов каждого вида взаимодействия, где значение N равно значению `self.depth`. Результаты вычислений сохраняются в датасете `self.df_most_similar_users` (фрагмент приведён на рис. 3.27), в котором содержатся значения косинусного сходства, рассматриваемый вид взаимодействия и место в ранжированном списке в рамках вида взаимодействия для каждого из пользовательских пространств эмбеддингов;

```
[17] model_p2v.df_most_similar_users.head()
```

	user_id	interaction	interaction_index	distance
0	2dc46b43e67b031df1c47fd377a3c9dd	read	1	0.999474
1	36ed8ae476efdebadc90715eeb0b2e01	read	2	0.999454
2	9d73bf5d4f05e490d759dc8f36e336ec	read	3	0.999436
3	686deb2f328c1f9348cb9bf4a4fa043d	read	4	0.999428
4	a5b852e370a8489b8a52996c4fd047e8	read	5	0.999425

Рис. 3.27. Фрагмент датасета self.df_most_similar_users модели Pref2Vec.

- 3) Данные поля self.df_most_similar_users группируются по значениям идентификаторов пользователей и агрегируются: вычисляется количество видов взаимодействий (столбец *interaction*), в рамках которых пользовательское пространство оказалось схожим с целевым, и среднее значение места в ранжированном списке в рамках вида взаимодействия (столбец *interaction_index*). Результаты вышеописанных манипуляций приведены на рис. 3.28.

user_id	interaction	interaction_index
002a15f1655b5330f6e844229a327b72	1	113.0
007ad78b340556c84244acec53a58875	1	185.0
00e029b79ddb8c6a70b937ad64ce748b	1	36.0
00f4927f15c58ca5d0b7f71d98aa5983	1	187.0
0150e283198c0dd64c35d3962486e047	1	158.0

Рис. 3.28. Фрагмент датасета self.df_most_similar_users модели Pref2Vec после группировки и агрегации.

- 4) После датасет сортируется: по убываю относительно количества видов взаимодействий (столбец *interaction*), в рамках которых пользовательское пространство оказалось схожим с целевым, и по возрастанию относительно среднего значения места в ранжированном списке в рамках вида взаимодействия (столбец *interaction_index*). Выбираются первые *topn* элементов как ближайшие к целевому. Этот способ сортировки позволяет найти пользовательские пространства эмбеддингов, схожие по

наибольшему количеству видов взаимодействий и наименьшему среднему значению мест в ранжированных списках в рамках видов взаимодействий:

user_id	interaction	interaction_index
8f451697cae31799ae47af86b2b5334	2	18.5
36c885627cb4289198a5bdb2e68211c	2	114.0
bad4164e62cbb33a9613d87b47d1ffca	2	121.0
72c0616c598fc9573bccbf29c702397e	2	125.0
f7f4573a2dbcf4d88acdb51ef5d43f6f	2	145.0
db10048081045becfa1f9a9d82ada8f7	2	154.0
e960dc8bd1ed318b52ab318842e4167e	2	154.5
dc03926d9766ee1a7ff1078a71e02270	2	182.5
8d41c71675029f04674dd6d2ad42a5ea	2	205.0
71b5087c42dc3c1919e606328f9dd9ce	2	211.5

Рис. 3.29. Фрагмент датасета self.df_most_similar_users модели Pref2Vec после группировки, агрегации и сортировки.

Также следует подробно описать алгоритм построение рекомендаций для определённого пользователя в модели:

- 1) Производится вызов метода *recommend_for_user()*, который определяет целевое пространство эмбеддингов. К тому же заранее определяются виды взаимодействий, для которых будет производиться рекомендация (*target_interactions*);
- 2) С помощью вышеописанного алгоритма поиска схожих пользовательских пространств (метод *most_similar_users()*) формируется список *topn* наиболее похожих пространств;
- 3) После этого производится анализ всех найденных ранее пространств на предмет схожести элементов с элементами целевого пространства (метод *_get_nearest_similarity()*): для каждого эмбеддинга элемента из найденных пространств рассчитывается косинусное сходство с каждым эмбеддингом объекта целевого пространства в рамках видов взаимодействий *target_interactions*, сохраняются только самые похожие эмбеддинги.

Промежуточные результаты расчётов сходств для каждого элемента найденных пространств сохраняются в датасет *self.df_embeddings_target* (рис. 3.30), ближайшие элементы хранятся в датасете *self.df_recommendations* (рис. 3.31);

		embedding similarity
25948653	[-0.004693891853094101, 0.005665168631821871, ...]	0.791816
27869168	[-0.006349078379571438, 0.004655817989259958, ...]	0.773331
27874358	[-0.004721857141703367, 0.012064071372151375, ...]	0.765639
27874359	[-0.006328290328383446, -0.00676772091537714, ...]	0.762065

Рис. 3.30. Фрагмент датасета *self.df_embeddings_target* модели Pref2Vec.

	book_id similarity
123	107017 0.998462
34	12903909 0.998430
46	128651 0.998306
30	10574407 0.998299
16	263145 0.998278

Рис. 3.31. Фрагмент датасета *self.df_recommendations* модели Pref2Vec.

- 4) Из полученного набора с ближайшими эмбеддингами объектов (датасет *self.df_recommendations*) выбираются *topn* элементов с наибольшим сходством – они же и берутся за рекомендации.

3.4.3. Анализ работы модели *Pref2Vec*

Для демонстрации и анализа работы модели Pref2Vec сперва требуется произвести построение корпуса модели: рассчитать центры кластеров взаимодействий для каждого из предоставляемых пользовательских пространств. Сформируем корпус из данных о взаимодействиях 20 000 пользователей с различными книгами (часть датасета *user_data.csv*) и заметим, что рассчитанные данные сохраняются в поле *self.corpus* (рис. 3.32).

Вычисление корпуса заняло практически 7 часов и что корпус сохраняется в указанный при вызове метода файл (*children_corpus_pref2vec_8.csv*):

```
model_p2v = Pref2Vec(depth=500)
model_p2v.build_corpus(
    users_data=df_user_data.iloc[40000:60000],
    filepath_or_buffer='/content/drive/MyDrive/BKP/children/' +
        'children_corpus_pref2vec_8.csv'
)
```

Листинг. 3.12. Код расчёта корпуса модели Pref2Vec.

	center_read	center_shelved	center_rating_0	center_rating_1	center_rating_2	center_rating_3	center_rating_4	center_rating_5
user_id								
966a49a655bc0139b81eedd945d19e95	[0.00322946464, -0.00210853759, -0.0044602935...	[-0.01905206, -0.0647061, -0.06223375, 0.07401...	NaN	NaN	[-0.02316315, -0.0791113, -0.04018758, 0.03838...	[-0.00351813, -0.03050093, -0.04097866, 0.0649...	[0.0024348381, 0.00345087145, 0.000884951674, ...	[0.0016037426, -0.00295122433, -0.00115017127,...
da18d9daa2e186ef75ba3c1b8e0251ff	[0.00341412169, 0.00673753629, 0.000876085542,...	[-0.000329168019, 0.00255199429, -0.0008348192...	NaN	NaN	NaN	[0.02328764, -0.11614487, -0.06096634, 0.0991...	[0.00518707, 0.00199469016, -0.00183946628, 0...	[0.0566545959, -0.0272693653, -0.0400875621,...
c5f458ef8717602d463a83df9e742e29	[-0.0123874648, -0.0146778096, -0.0417053923, ...	[-0.00257342821, -0.0077736848, -0.00257894443...	NaN	NaN	NaN	NaN	[-0.0081034461, -0.00730984239, -0.0211602021...	[0.00333358184, 0.0159215834, 0.00740838191,...
b771ce0ba2d5841afc2d74ef764fe7da	[0.00192710804, 0.000242090842, 0.000270974502...	[0.0522565357, -0.0089023672, -0.0380353443, ...	NaN	NaN	NaN	NaN	[-0.00726822764, -0.00434292015, 0.00160620746...	[0.00116420898, -5.50426812e-05, -0.001124514...
d3409ae8bd22979302f2681bb6164189	[0.00184287957, -0.0175736547, -0.00128198112...	NaN	NaN	NaN	NaN	NaN	NaN	[0.0108982343, -0.00859883148, 0.00517631229, ...

Рис. 3.32. Фрагмент рассчитанного корпуса модели Pref2Vec.

В то же время имеется возможность загрузки ранее рассчитанного корпуса в инициализированную модель из CSV-файла:

```
model_p2v = Pref2Vec(depth=500)
model_p2v.load_corpus(
    filepath_or_buffer = '/content/drive/MyDrive/BKP/children/' +
        'children_corpus_pref2vec.csv'
)
```

Листинг. 3.13. Код загрузки корпуса модели Pref2Vec.

После построения или загрузки корпуса открывается возможность поиска ближайших пользовательских пространств к целевому и возможность построение рекомендаций так же для целевого пользователя. Продемонстрируем поиск пяти ближайших пользовательских пространств

эмбеддингов к пространству пользователя с идентификатором 7e4945a6266566e39e3f251fff0fdde1:

```
[36] most_similar = model_p2v.most_similar_users(
    df_user_data.loc['7e4945a6266566e39e3f251fff0fdde1'],
    topn=5,
    show_info=True
)

→ Ближайшие пользователи к 7e4945a6266566e39e3f251fff0fdde1:
1) Пользователь №ab8b074511e34f4f69c77055df8783dd
   read - 0.9995276216430901
   rating_4 - 0.9991808308440763
   rating_5 - 0.9994571928729661
2) Пользователь №7e4945a6266566e39e3f251fff0fdde1
   shelved - 0.985019776041554
   read - 0.9995162070021204
3) Пользователь №8522eba1011e52d7672f9d9aeccc044c
   read - 0.9995273464771803
   rating_4 - 0.9991496382676844
4) Пользователь №ef85b7b9576d703fedad657ae85be031
   rating_4 - 0.9992045809804649
   rating_5 - 0.9994605067343586
5) Пользователь №5f934adf91e212e1f0623434d7e5c849
   rating_4 - 0.9991456038322032
   rating_5 - 0.9994849095384216
```

Рис. 3.33. Код поиска пяти ближайших пользовательских пространств к пространству пользователя с идентификатором 7e4945a6266566e39e3f251fff0fdde1.

По полученным результатам видно, что были получены идентификаторы пяти пользователей с ближайшими пользовательскими пространствами. Так же выводится информация о том, в каком виде взаимодействия было найдено пространство и с каким косинусным сходством.

Вслед за этим воспользуемся методом построения пяти рекомендаций для пользователя с идентификатором 7e4945a6266566e39e3f251fff0fdde1. По результатам выполнения (рис. 3.34) видно, что были получены пять рекомендаций – пять книг с указанными идентификаторами и значениями косинусных сходств.

Чтобы проанализировать семантическую взаимосвязь результатов, полученных при построении рекомендаций, и целевого пространства, рассмотрим облака слов пространства эмбеддингов пользователя 7e4945a6266566e39e3f251fff0fdde1 (рис. 3.35 – 3.37). Также обратимся к датасету *books.csv* и выведем содержимое сформированных рекомендаций.

```
[46] recommendations = model_p2v.recommend_for_user(
    target_user_data=df_user_data.loc['7e4945a6266566e39e3f251fff0fdde1'],
    topn=5,
    show_info=True
)
```

- Рекомендации для пользователя 7e4945a6266566e39e3f251fff0fdde1:
- 1) Книга #4948 - 0.9999574549703204
 - 2) Книга #857501 - 0.9996459536245544
 - 3) Книга #327194 - 0.9996342351790272
 - 4) Книга #263636 - 0.9996246445146233
 - 5) Книга #491940 - 0.9996014773972516

Рис. 3.34. Код построения пяти рекомендаций для пользователя с идентификатором 7e4945a6266566e39e3f251fff0fdde1.

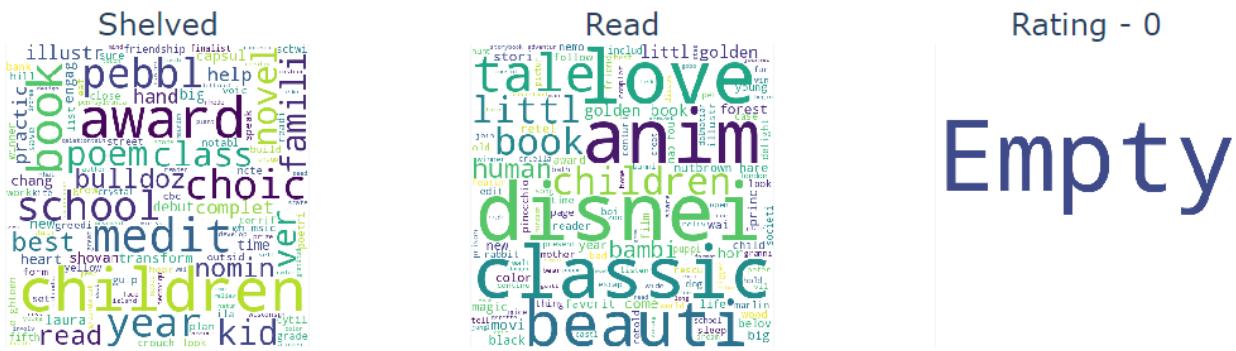


Рис. 3.35. Облака слов книжных аннотаций категорий shelved, read и rating_0 пользователя с идентификатором 7e4945a6266566e39e3f251fff0fdde1.



Рис. 3.36. Облака слов книжных аннотаций категорий rating_1, rating_2 и rating_3 пользователя с идентификатором 7e4945a6266566e39e3f251fff0fdde1.



Рис. 3.37. Облака слов книжных аннотаций категорий rating_4 и rating_5 пользователя с идентификатором 7e4945a6266566e39e3f251fff0fdde1.

book_id	description	title
491940	Arthur is unhappy about going on vacation with...	Arthur's Family Vacation (Arthur Adventure Ser...
4948	Eric Carle's The Very Hungry Caterpillar is a p...	The Very Hungry Caterpillar
263636	Brother and Sister Bear are not greedy children...	The Berenstain Bears and the Trouble with Comm...
857501	Illus. in full color. A madcap band of dancing...	Hand, Hand, Fingers, Thumb
327194	Three little kittens lose, find, soil, and wash...	Three Little Kittens

Рис. 3.38. Книги, рекомендуемые пользователю 7e4945a6266566e39e3f251fff0fdde1.

Можем убедиться в том, что между результатами, полученными при построении рекомендаций, и целевым пространством эмбеддингов пользователя 7e4945a6266566e39e3f251fff0fdde1 определённо присутствует взаимосвязь – так же, как и в категориях «rating_4» и «rating_5», среди рекомендаций присутствуют сказки. Дальнейший анализ будет произведен в гл. 4.

3.5. Выводы по третьей главе

Для реализации ЯРС и его тестирования был выбран набор данных веб-сервиса «GoodReads» по причинам распространённости его использования и наличия как и данных об объектах, так данных о взаимодействиях пользователей с ними. На основе набора данных была произведена подготовка исходных данных: формирование корпуса книжных аннотаций и датасета пользовательских предпочтений.

Вслед за этим было реализовано формирование эмбеддингов при помощи модели Doc2Vec. Анализ эмбеддингов, полученных на основе корпуса аннотаций книг жанра «Children», на основе решения задачи кластеризации показал, что механизм их формирования позволяет отобразить семантику книжных аннотаций в вычисляемых векторных представлениях.

Далее был разработан класс пользовательского пространства эмбеддингов User Embeddings Space (UES), который позволяет векторизовать предпочтения

конечного пользователя. Объект класса UES содержит набор расширенных эмбеддингов, которые отображают семантику векторизуемого объекта и семантику взаимодействия пользователя с объектом. Имеется возможность их визуализации при помощи метода t-SNE. Класс пользовательского пространства эмбеддингов также позволяет решить задачу кластеризации путём поиска центров кластеров взаимодействий, которые используются далее в построении рекомендаций. Была продемонстрирована работа класса на примере вычисления пользовательского пространства эмбеддингов для одного из пользователей набора данных веб-сервиса «GoodReads».

Затем была реализована модель векторизации пользовательских предпочтений Pref2Vec. Модель позволяет построить множество пользовательских пространств эмбеддингов – корпус объектов UES, представленных в виде набора вычисленных центров кластеров взаимодействий. На основе сформированного корпуса открывается возможность исследования пользовательских пространств эмбеддингов, которое позволяет строить рекомендации для конечного пользователя посредством поиска схожих пользовательских предпочтений, то есть схожих пространств. Дальнейшая рекомендация объектов происходит при помощи анализа расширенных эмбеддингов наиболее подобных пользовательских пространств. Также была продемонстрирована работа модели Pref2Vec посредством поиска схожих пользовательских пространств эмбеддингов к одному из пользователей набора данных веб-сервиса «GoodReads» и построением рекомендаций для него.

ГЛАВА 4. ТЕСТИРОВАНИЕ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ

Тестирование является неотъемлемой частью процесса разработки РС, которая позволяет оценить качество работы рассматриваемой системы. Существуют различные способы тестирования РС, некоторые из них были рассмотрены ранее в п. 2.4.

Дальнейшее тестирование будет проводиться на основе результатов, полученных в научной работе «Hierarchical Gating Networks for Sequential Recommendation» [18], так как в ней происходит оценка работы системы на тех же датасетах, на которых велась разработка модели Pref2Vec. К тому же в результатах тестирования модели Hierarchical Gating Networks приводится сравнительный анализ с довольно актуальными моделями РС.

4.1. Подготовка тестовых данных

Для тестирования разработанного ЯРС так же будет использоваться набор данных веб-сервиса «GoodReads» [31]. Проверка работы системы будет проводиться основе данных, полученных в п. 3.1.4 – датасеты *comics_graphic_user_data.csv* и *children_user_data.csv*. Данные каждой записи о взаимодействиях какого-либо пользователя с различными книгами будут разбиты на две части:

- 80% взаимодействий – выборка X, которая будет использоваться для формирования пользовательского пространства эмбеддингов и на основе которой будет происходить рекомендация книг;
- 20% взаимодействий – выборка Y, которая будет браться за эталон рекомендации.

Пример разбиения данных для пользователя с идентификатором dd1457ebb737a8444bb9d942d1b6e4b7:

	user_id	read	shelved	rating_0	rating_1	rating_2	rating_3	rating_4	rating_5
dd1457ebb737a8444bb9d942d1b6e4b7		['30282136', '28765232', '22839420', '25172436...']	['25903298', '6593831', '17445446', '9427629',...]		['16002637', '25066580', '18749019', '22728459...']	['33982653', '33621854', '33100438', '32768409...']	['33835683', '33384113', '31512272', '33561590...']	['28765232', '25172436', '23437328', '27483331...']	['30282136', '22839420', '15726505', '13035053...']

Рис. 4.1. Выборка X для данных пользователя с идентификатором dd1457ebb737a8444bb9d942d1b6e4b7.

	user_id	read	shelved	rating_0	rating_1	rating_2	rating_3	rating_4	rating_5
dd1457ebb737a8444bb9d942d1b6e4b7		['3194962', '2282807', '1266104', '1954201', ...]	[]	[]	[]	[]	[]	['1954197', '1953983', '1953855', '1953842', ...]	['1953741', '1119046', '160841', '100723', '59...']

Рис. 4.2. Выборка Y для данных пользователя с идентификатором dd1457ebb737a8444bb9d942d1b6e4b7.

Ввиду того, что основные метрики тестирования РС направлены на оценку формируемых системой списков рекомендаций [22], требуется привести выборки Y к списковому виду. Есть два варианта решения этой задачи:

- 1) «В лоб» объединить все объекты различных видов взаимодействий выборки Y в единый список. Данный вариант позволяет протестировать систему на предмет вхождения рекомендованных элементов в список выборки Y без учёта ранжирования;
- 2) Вычислить косинусное сходство каждого объекта выборки Y с каждым эмбеддингом выборки X, сохранив максимальные значения сходства. Далее ранжировать элементы выборки Y по вычисленным сходствам и сформировать список. Этот подход учитывает тестирование и вхождения рекомендованных элементов в список выборки Y, и ранжирование списка рекомендованных элементов

В дальнейшем будем придерживаться второго способа приведения выборки Y к списковому виду, так как он позволяет протестировать РС на предмет качества ранжирования рекомендуемых объектов. Приведём пример

ранжированной выборки Y для пользователя с идентификатором dd1457ebb737a8444bb9d942d1b6e4b7:

```
array(['1953831', '1994852', '2282807', '1953817', '1954201', '59980',
       '59980', '6057544', '100712', '1188314', '1188314', '1198715',
       '1198715', '1953842', '1953842', '1954197', '1954197', '707896',
       '707896', '6456427'], dtype=object)
```

Рис. 4.3. Ранжированная выборка Y для данных пользователя с идентификатором dd1457ebb737a8444bb9d942d1b6e4b7.

4.2. Разработка класса оценки работы рекомендательной системы

По процессу тестирования, приведённом в работе «Hierarchical Gating Networks for Sequential Recommendation» [18], можно прийти к выводу, что на рассматриваемых наборах данных следует воспользоваться метриками NCDG@k и Recall@k. Для их реализации был использован код Брэндина Уайта [32].

Для оптимизации процесса проверки работы РС был разработан класс *Evaluator*, с помощью которого будет производиться вышеописанная подготовка тестовых данных и собственно расчёт значений метрик NCDG@k и Recall@K. Код разработанного класса приведён в приложении 3.

Класс *Evaluator* имеет поля *self.model_p2v* (модель Pref2Vec, работа которой будет тестироваться), *self.test_percent* (процент разбиения данных на выборки X и Y), *self.df_testing* (датасет с результатами тестирования), *self.k* (параметр k, используемый в метриках NDCCD@k и Recall@k), *self.x* (формируемая выборка X), *self.y* (формируемая выборка Y) и *self.y_true* (ранжированная выборка Y). Реализуются следующие методы:

- *__init__()* – метод инициализации класса, в котором производится инициализация полей *self.model_p2v* (модель Pref2Vec) и *self.test_percent* (процент разбиения данных на выборки X и Y);

- *test()* – метод, который производит тестирование модели *model_p2v* на пользовательских данных *users_data*. Используются метрики NDCD@k и Recall@k. Результаты тестирования сохраняются в поле *self.df_testing*. Кроме того, имеется возможность загрузки результатов в CSV-файл;
- *__sample_preparation()* – метод разбиения тестовых данных на выборки X и Y и ранжирования выборки Y;
- *__get_y_true()* – метод, осуществляющий ранжирование рассматриваемой выборки Y. Вычисляется косинусное сходство каждого объекта выборки Y с каждым эмбеддингом выборки X, сохранив максимальные значения сходства. Далее элементы выборки Y ранжируются по вычисленным сходствам и формируется список *y_true*;
- *__get_nearest_distance()* – метод, при помощи которого вычисляется косинусное сходство эмбеддинга объекта выборки Y с каждым эмбеддингом выборки X. Выводит максимальное значение сходства;
- *recall_at_k()* – метод, позволяющий получить значение метрики Recall@k;
- *dcg_at_k()* – метод, позволяющий получить значение метрики DCG@k;
- *ndcg_at_k()* – метод, позволяющий получить значение метрики NDCG@k.

4.3. Тестирование модели Pref2Vec

В вышеупомянутой работе [18] при оценке работы ЯРС используются два набора данных веб-сервиса «GoodReads» [31]: «Children» и «Comics & Graphic», описывающие детские книги и комиксы и графические романы соответственно и взаимодействия пользователей с ними. Эти же наборы данных используются далее в тестировании разработанной модели.

Процесс тестирования модели Pref2Vec при помощи класса Evaluator представляет из себя следующую последовательность действий:

- 1) Пользовательские данные для одного пользователя разбиваются на выборки X, Y;
- 2) Выборка Y ранжируется относительно выборки X (см. п. 4.1);
- 3) Выборка X подаётся на вход модели Pref2Vec, вычисляются рекомендации;
- 4) Полученный список рекомендаций сравнивается с ранжированной выборкой Y посредством метрик NCDG@k и Recall@k;
- 5) Вышеописанный алгоритм повторяется для N пользователей. После вычисляются средние значения метрик NCDG@k и Recall@k и выводятся как результат тестирования.

Алгоритм начинает свою работу по выполнении кода, представленном на листинге 4.1. Ввиду того, что процесс построения рекомендаций модели Pref2Vec для одного пользователя занимает достаточно много времени (около четырёх минут), тестирование производится для 200 пользователей. Также в связи с наличием временных ограничений при выполнении выпускной квалификационной работы, тестируемая модель содержит корпус из данных о 100000 пользовательских пространств эмбеддингов.

```
model_p2v = Pref2Vec(depth=500)
model_p2v.load_corpus(
    filepath_or_buffer='/content/drive/MyDrive/BKP/comics_graphic/comics_graphic_corpus_pref2vec.csv'
)

evaluator = Evaluator(model_p2v=model_p2v, test_percent=20)
evaluator.test(
    users_data=df_user_data.iloc[:200], k=15,
    filepath_or_buffer='/content/drive/MyDrive/BKP/comics_graphic/comics_graphic_testing_depth_500_percent_20_k_15_1.csv'
)
```

Листинг. 4.1. Код инициализации класса Evaluator и запуска тестирования модели Pref2Vec.

При тестировании разработанного ЯРС на наборах данных «Children» и «Comics & Graphic» были получены значения, представленные в табл. 4.1.

Также было произведено сравнение полученных значений с показателями наиболее актуальных моделей (рис. 4.4 – 4.7).

Таблица 4.1

Показатели метрик NCDG@k и Recall@k при тестировании модели Pref2Vec

	Recall@10	Recall@15	Recall@20	NDCG@10	NDCG@15	NDCG@20
<i>GoodReads-Children</i>	0,0904	0,1056	0,1585	0,0996	0,0769	0,0628
<i>GoodReads-Comics</i>	0,0542	0,1033	0,1627	0,1140	0,1099	0,0933

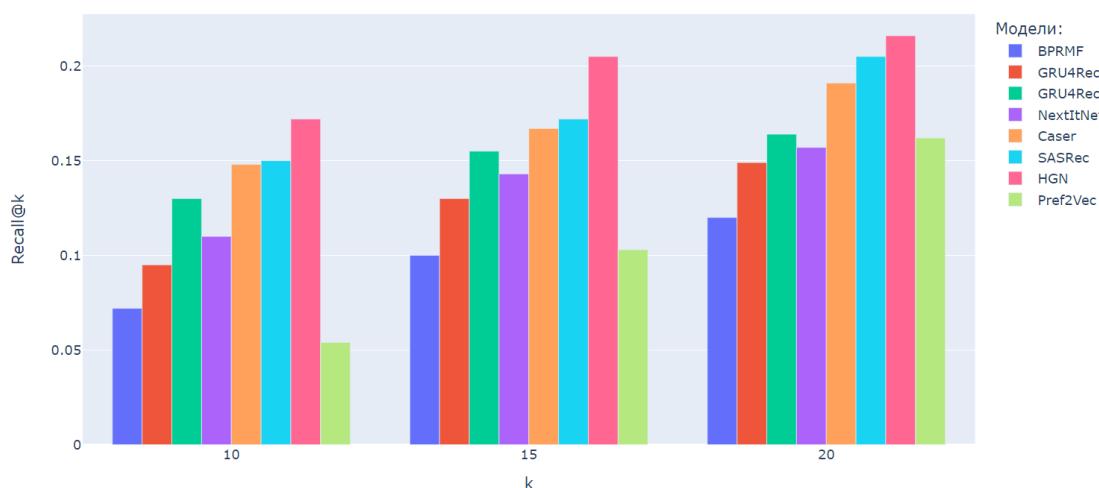


Рис. 4.4. Сравнение показателей метрики Recall@k различных РС на наборе данных «Comics & Graphic».

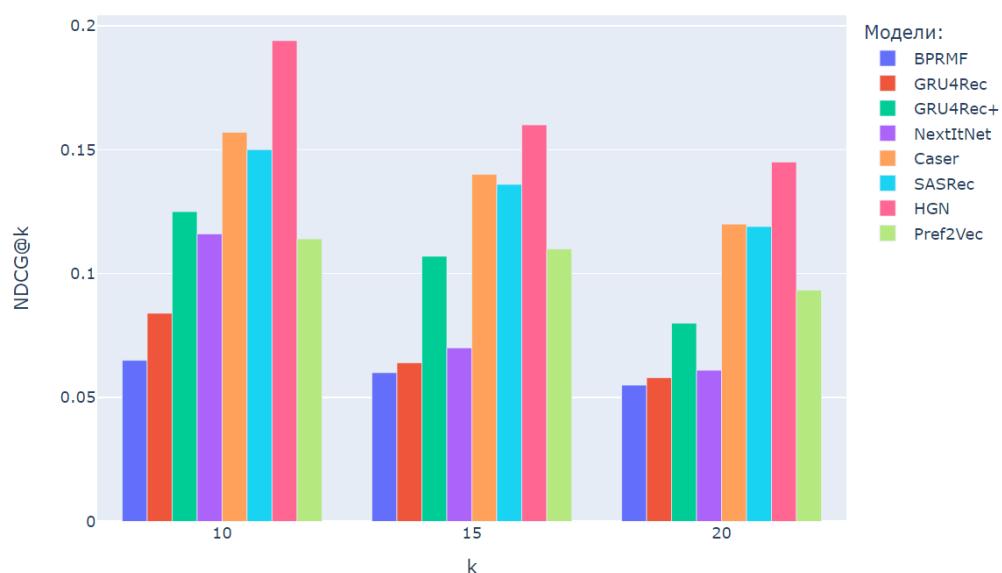


Рис. 4.5. Сравнение показателей метрики NCDG@k различных РС на наборе данных «Comics & Graphic».

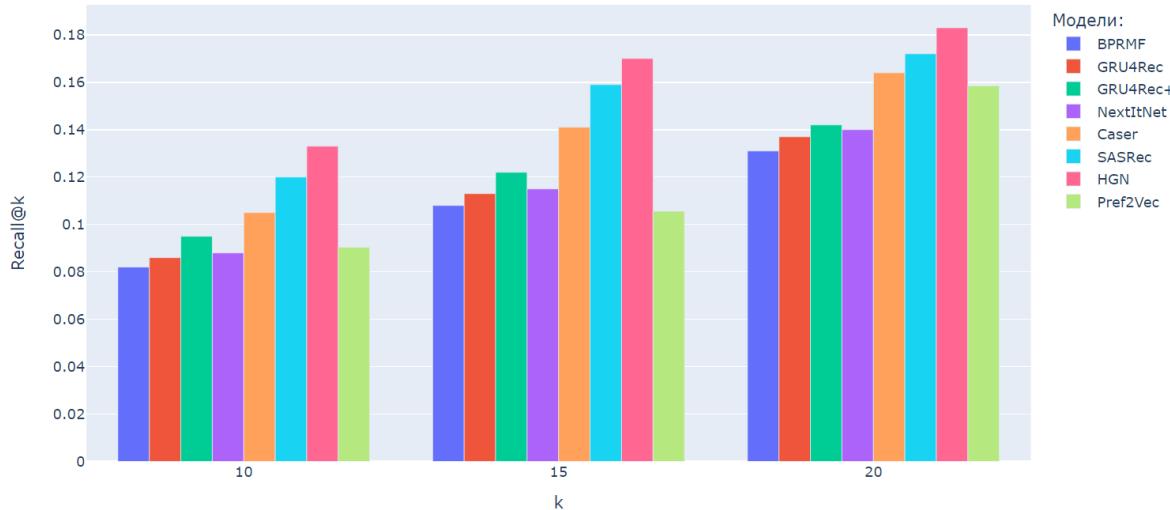


Рис. 4.6. Сравнение показателей метрики Recall@k различных РС на наборе данных «Children».

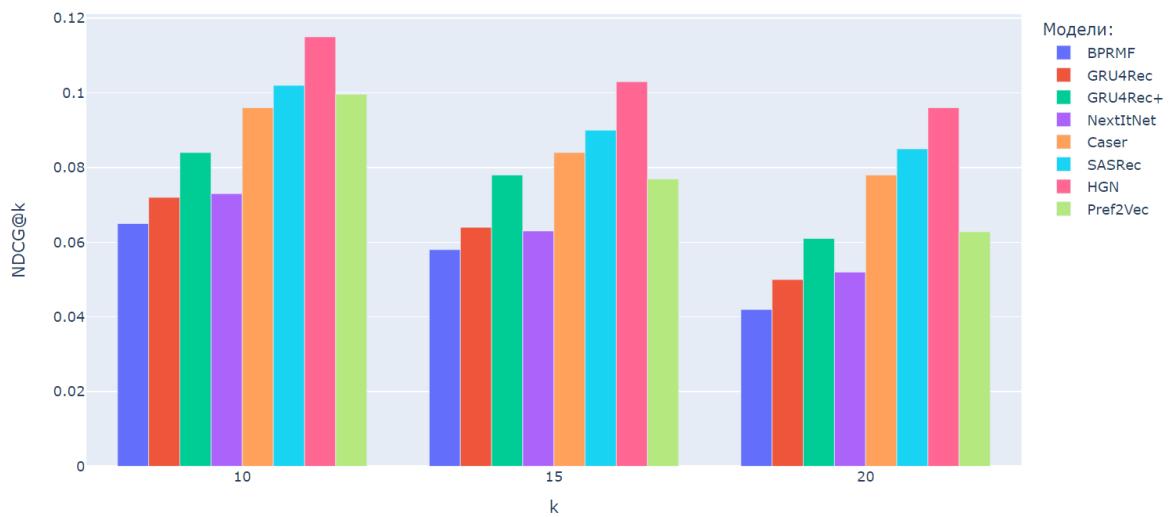


Рис. 4.7. Сравнение показателей метрики NDCG@k различных РС на наборе данных «Children».

Результаты тестирования сравниваются с моделями:

- *HGN (Hierarchical Gating Network)* – модель, применяющая иерархическую вентильную сеть для изучения представлений последовательности элементов на уровне группы;
- *BPRMF (Bayesian Personalized Ranking from Implicit Feedback)* – модель, которая использует матричную факторизацию на основе байесовского персонализированного ранжирования;
- *GRU4Rec (Gated Recurrent Unit for Recommender)* – модель, которая использует рекуррентные нейронные сети для моделирования

последовательностей взаимодействия пользователя с элементом для рекомендаций на основе сеанса;

- *GRU4Rec+* – улучшенная версия GRU4Rec, которая использует другую функцию потерь и стратегию выборки и демонстрирует значительный прирост производительности при рекомендации вида Top-N;
- *NextItNet* – РС, которая применяет расширенные сверточные нейронные сети для увеличения рецептивных полей, не полагаясь на операцию объединения;
- *Caser (Convolutional Sequence Embedding Recommendation Model)* – модель вложения сверточной последовательности, которая захватывает цепи Маркова высокого порядка путем применения операций свертки;
- *SASRec (Self-Attentive Sequential Recommendation)* – модель, которая использует механизм self-attention для идентификации соответствующих элементов для прогнозирования следующего элемента.

Приведём более детальное сравнение значений метрик NCDG@k и Recall@k для $k = 10$:

Таблица 4.2

Сравнение метрик NCDG@10 и Recall@10 различных РС

	BPRMF	GRU4Rec	GRU4Rec+	NextItRec	Caser	SASRec	HGN	Pref2Vec
	Recall@10							
<i>GoodReads-Children</i>	0,0814	0,0857	0,0978	0,0879	0,1060	<u>0,1165</u>	0,1263	0,0904
<i>GoodReads-Comics</i>	0,0788	0,0958	0,1288	0,1078	0,1473	<u>0,1494</u>	0,1743	0,0542
NDCG@10								
<i>GoodReads-Children</i>	0,0664	0,0715	0,0821	0,0720	0,0943	<u>0,1007</u>	0,1130	0,0996
<i>GoodReads-Comics</i>	0,0713	0,0912	0,1328	0,1171	<u>0,1629</u>	0,1592	0,1927	0,1140

Результаты тестирования модели Pref2Vec (табл. 4.1–4.2 и рис. 4.4–4.7) показали, что значения метрики Recall@k оказались не столь высокими относительно показателей одних из последних моделей РС (HGN, SASRec и Caser), но придерживаются примерно того же уровня, что другие модели,

представленные в работе «Hierarchical Gating Networks for Sequential Recommendation» [18] (BPRMF, GRU4Rec и NextItRec). Однако, значения метрик NCDG@k, напротив, оказались относительно высокими, что говорит о довольно хорошей способности ранжирования рекомендаций моделью Pref2Vec. Принимая к сведению введение нового подхода в обработке пользовательских данных и факт наличия временных ограничений при разработке и тестировании модели, можно сказать, что полученные результаты вполне удовлетворяют ожиданиям.

4.4. Оценка перспективности разработанной модели

Подытоживая анализ результатов тестирования модели векторизации пользовательских предпочтений, можно заявить, что были получены значения, которые стоят примерно на одном уровне с одними из наиболее актуальных разработок в сфере проектирования РС. Это говорит о том, что разработанная модель Pref2Vec является достаточно перспективной, но, в то же время, возможны некоторые улучшения. Рассмотрим некоторые из них.

Показатели оценки качества работы спроектированного и реализованного ЯРС могут оказаться выше при большем корпусе модели Pref2Vec (из-за временных ограничений был использован корпус из данных о 100 000 пользовательских пространств эмбеддингов, хотя в наборе данных жанра «Children» представлены данные 544 000 пользователей и в наборе «Comics & Graphic» – данные 310 000 пользователей [31]. Также возможен рост показателей при проведении тестирования на большем количестве пользователей или (и) при варьировании гиперпараметра *depth* модели Pref2Vec.

Кроме того, возможно увеличение показателей при варьировании значений гиперпараметров модели Doc2Vec, при помощи которой происходит формирование исходных эмбеддингов. Качество формирования векторных

представлений объектов прямо влияет на качество построения рекомендаций моделью Pref2Vec.

Также на качество построения рекомендаций оказывает непосредственное влияние метод поиска центров кластеров взаимодействий. Вероятно, использование таких методов, как методов K-Means [16], распространения близости (Affinity Propagation) [1] или, например, иерархической кластеризации (Agglomerative Clustering) [2], позволит увеличить значения показателей.

К тому же возможно увеличение показателей модели Pref2Vec с помощью внедрения методов пространственных вычислений библиотеки SciPy, отличных от косинусного сходства: например, вычисление Манхэттенского расстояния [6], Евклидова расстояния [7] или расстояния Махalanобиса [19].

4.5. Выводы по четвёртой главе

В качестве метрик оценки работоспособности модели были использованы метрики Recall@k и NCDG@k, которые были выбраны в целях достижения сопоставимости результатов оценки с показателями наиболее актуальных моделей в сфере разработки РС, представленных в научной работе «Hierarchical Gating Networks for Sequential Recommendation» [18].

В процессе оценки качества работы ЯРС был разработан класс Evaluator, при помощи которого производилась подготовка тестовых данных (разбиение пользовательских данных на выборки X и Y и ранжирование выборки Y) и собственно вычисление значений метрик Recall@k и NCDG@k. Так как процесс построения рекомендаций модели Pref2Vec для одного пользователя занимает достаточно много времени, было произведено тестирование для 200 пользователей. Корпус модели Pref2Vec содержал данные о 100 000 пользовательских пространств. Результаты оценки качества работы реализованной системы представлены в табл. 4.1.

Вместе с тем был произведён сравнительный анализ полученных результатов с показателями некоторых из моделей РС. Анализ показал, что полученные результаты сравнимы с показателями моделей BPRMF, GRU4Rec и NextItRec, что говорит о конкурентоспособности разработанного ЯРС. Значения метрик NCDG@k оказались относительно высокими, что говорит о довольно хорошей способности ранжирования рекомендаций моделью Pref2Vec. Также следует отметить, что была произведена оценка конкурентоспособности спроектированной и реализованной модели, и были приведены возможные пути её усовершенствования.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы был осуществлён обзор предметной области разработки РС, в процессе чего были поставлены цель работы и задачи, которые подлежали решению при разработке собственного ЯРС.

Получены следующие результаты работы:

- 1) Предложена модель векторизации пользовательских предпочтений Pref2Vec в качестве основы гибридной РС. Модель является новым подходом к построению РС, при котором решается проблема внедрения контекстной зависимости в ЯРС посредством перехода к более высокому уровню абстракции – формирование рекомендаций производится в зависимости от предпочтений конкретного пользователя. Также модель позволяет придерживаться принципов гибридной рекомендации, что является наиболее актуальным подходом в разработке РС;
- 2) Предложен класс пользовательского пространства эмбеддингов UES, который позволяет векторизовать предпочтения конечного пользователя и который используется в построении модели Pref2Vec. Объект класса UES содержит набор расширенных эмбеддингов, которые отображают семантику векторизуемого объекта и семантику взаимодействия пользователя с объектом;
- 3) Предложен новый способ применения задачи кластеризации в решении задачи построения рекомендаций, который позволил решить следующие проблемы: проблема разреженности данных, проблема масштабируемости разрабатываемой системы, и проблема «серых овец»;
- 4) Реализованы модель Pref2Vec и класс UES на языке программирования Python в качестве расширения функциональности библиотеки Gensim;
- 5) Были осуществлены выбор и подготовка набора данных, который использовался, как и при разработке ЯРС, так и при его тестировании.

Следовательно, удалось обойти проблемы доступности датасетов и оценки работы системы;

- 6) Произведено тестирование предложенной модели Pref2Vec при помощи разработанного класса Evaluator, который позволяет произвести подготовку тестовых данных и вычисление значений метрик Recall@k и NCDG@k;
- 7) Произведен сравнительный анализ результатов тестирования с показателями наиболее актуальных моделей в сфере разработки РС. Анализ показал, что полученные результаты сравнимы с показателями моделей BPRMF, GRU4Rec и NextItRec и что показатели метрики Recall@k модели Pref2Vec обладают большей скоростью роста при росте параметра k, чем большинство представленных моделей. Также в ходе анализа было выяснено, что модель Pref2Vec обладает довольно высоким качеством ранжирования рекомендаций;
- 8) Произведена оценка перспективности предложенного решения и рассмотрены пути улучшения показателей модели Pref2Vec.

Таким образом при выполнении выпускной квалификационной работы цель была достигнута, и все поставленные задачи были решены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. AffinityPropagation – Scikit-learn Version: 1.5.0 // Scikit-learn [Электронный ресурс]. URL: <https://clck.ru/3BBWJG> (дата обращения: 24.05.2024);
2. AgglomerativeClustering – Scikit-learn Version: 1.5.0 // Scikit-learn [Электронный ресурс]. URL: <https://clck.ru/3BBWkK> (дата обращения: 24.05.2024);
3. Barkan O., Koenigstein N. ITEM2VEC: Neural item embedding for collaborative filtering // Proceedings of the IEEE 26th International Workshop on Machine Learning for Signal Processing. – MLSP, 2016. – C. 1-6;
4. Bird S., Loper E. NLTK: The Natural Language Toolkit // Proceedings of the ACL Interactive Poster and Demonstration Sessions. – ACL, 2004. – C. 214-217;
5. Christopher D., Surdeanu M., Bauer J., Finkel J., Bethard S. J., McClosky D. The Stanford CoreNLP Natural Language Processing Toolkit // Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. – ACL, 2014. – C. 55-60;
6. Cityblock distance – Scipy Version: 1.13.1 // Scipy [Электронный ресурс]. URL: <https://clck.ru/3BBWur> (дата обращения: 29.05.2024);
7. Euclidean distance – Scipy Version: 1.13.1 // Scipy [Электронный ресурс]. URL: <https://clck.ru/3BBWwQ> (дата обращения: 29.05.2024);
8. He X., Liao L., Zhang H., Nie L., Hu X., Chua T. Neural Collaborative Filtering // Proceedings of the International World Wide Web Conference Committee. – WWW, 2017. – C. 173-182;
9. Hillier W. What's the Best Language for Machine Learning? // The CareerFoundry Blog [Электронный ресурс]. URL: <https://clck.ru/3BBWxz> (дата обращения: 10.05.2024);

10. Hu Y., Koren Y., Volinsky C. Collaborative Filtering for Implicit Feedback Datasets // Proceedings of the IEEE International Conference on Data Mining, ICDM. – ICDM, 2008. – C. 263-272;
11. Jannach D., Zanker M., Felfernig, A., Friedrich, G. Recommender Systems: An Introduction: методическое пособие. – Cambridge University Press, 2010. – 352 с.;
12. Jannach D., Zanker M., Friedrich G. Tutorial: Recommender Systems // Proceedings of the International Joint Conference on Artificial Intelligence. – IJCAI, 2013 221 – C. 221-236;
13. Jurafsky D., Martin J. H. Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Second Edition: методическое пособие. – PEARSON INDIA, 2009. – T. 2. – 1044 с.;
14. Karatzoglou A., Hidasi B. Deep Learning for Recommender Systems // RecSys '17: Proceedings of the Eleventh ACM Conference on Recommender Systems. – RecSys, 2017. – C. 396-397;
15. Khusro S., Ali Z., Ullah I. Recommender Systems: Issues, Challenges, and Research Opportunities // Proceedings of the Information Science and Applications. – ICISA, 2016. – C. 1179-1189;
16. K-Means – Scikit-learn Version: 1.5.0 // Scikit-learn [Электронный ресурс]. URL: <https://clck.ru/3BBX2E> (дата обращения: 15.04.2024);
17. Le Q. V., Mikolov T. Distributed Representations of Sentences and Documents // Proceedings of the 31st International Conference on Machine Learning. – PMLR 32(2), 2014. – C. 1188-1196;
18. Ma C., Kang P., Liu X. Hierarchical Gating Networks for Sequential Recommendation // Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining – KDD, 2019. – C. 825-833;
19. Mahalanobis distance – Scipy Version: 1.13.1 // Scipy [Электронный ресурс]. URL: <https://clck.ru/3BBXkN> (дата обращения: 29.05.2024);

20. Manning C. D., Raghavan P., Schütze H. An Introduction to Information Retrieval. Online edition: методическое пособие. – Cambridge University Press, 2009. – 544 с.;
21. McGregor M. 8 Clustering Algorithms in Machine Learning that All Data Scientists Should Know // FreeCodeCamp [Электронный ресурс]. URL: <https://clck.ru/3BBXnX> (дата обращения: 05.04.2024);
22. Michiels L., Verachtert R., Goethals B. RecPack: An(other) Experimentation Toolkit for Top-N Recommendation using Implicit Feedback Data // Proceedings of the 16th ACM Conference on Recommender Systems. – RecSys, 2022. – С. 648-651;
23. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space // Proceedings of Workshop at the International Conference on Learning Representations. – ICLR, 2013. – С. 1-12;
24. NumPy Version: 2.1.dev0 // NumPy [Электронный ресурс]. URL: <https://clck.ru/3BBXqT> (дата обращения: 03.04.2024);
25. Pandas Version: 2.2.2 // Pandas [Электронный ресурс]. URL: <https://clck.ru/3BBXsd> (дата обращения: 03.04.2024);
26. Pedregosa F., Varoquaux G., Gramfort A., Michel V. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. – 2011. – Т. 12. – С. 2825-2830;
27. Plotly package // Graphing Libraries: Plotly [Электронный ресурс]. URL: <https://clck.ru/3BBXuH> (дата обращения: 08.04.2024);
28. Schröder G., Thiele M., Lehner W. Setting goals and choosing metrics for recommender system evaluations. // Proceedings of the UCERSTI2 Workshop at the 5th ACM Conference on Recommender Systems. – UCERSTI2, 2011 – С. 23-53;
29. Schubert E., Rousseeuw P. J. Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms // Information Systems. – 2021. – Т. 101. – С. 101-804;

30. Virtanen P., Gommers R., Oliphant T. E., Haberland M. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python // Nature Methods. – 2020. – Т. 17. – С. 261-272;
31. Wan M., McAuley J. Item Recommendation on Monotonic Behavior Chains // Proceedings of the 12th ACM Conference on Recommender Systems. – RecSys, 2018. – С. 86-94;
32. White B. Information Retrieval metrics // Github [Электронный ресурс]. URL: <https://clck.ru/3BBXx7> (дата обращения: 03.05.2024);
33. WordCloud for Python Version: 1.8.1 // WordCloud [Электронный ресурс]. URL: <https://clck.ru/3BBXyL> (дата обращения: 15.04.2024);
34. Yan A., Cheng S., Kang W., Wan M., McAuley J. CosRec: 2D Convolutional Neural Networks for Sequential Recommendation // Proceedings of the 28th ACM International Conference on Information and Knowledge Management. – CIKM, 2019. – С. 2173-2176;
35. Yellowbrick: Clustering Visualizers – Elbow Method // Yellowbrick [Электронный ресурс]. URL: <https://clck.ru/3BBY35> (дата обращения: 05.04.2024);
36. Řehůřek R. Software Framework for Topic Modelling with Large Corpora // Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. – LREC, 2010. – С. 45-50.

ПРИЛОЖЕНИЕ 1. КОД КЛАССА ПОЛЬЗОВАТЕЛЬСКОГО ПРОСТРАНСТВА ЭМБЕДДИНГОВ

```

from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import pandas as pd
import re

from gensim.parsing.preprocessing import preprocess_string
from gensim.models.doc2vec import Doc2Vec

from sklearn.manifold import TSNE
from sklearn_extra.cluster import KMedoids

import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from wordcloud import WordCloud


READ      = 'read'
SHELVED   = 'shelved'
RATING_0  = 'rating_0'
RATING_1  = 'rating_1'
RATING_2  = 'rating_2'
RATING_3  = 'rating_3'
RATING_4  = 'rating_4'
RATING_5  = 'rating_5'

interactions = [SHELVED, READ, RATING_0, RATING_1, RATING_2, RATING_3, RATING_4,
RATING_5]

left_boundary = -1
right_boundary = 1
step = (right_boundary - left_boundary) / (len(interactions) - 1)

VALUE_READ      = left_boundary + 0 * step
VALUE_SHELVED   = left_boundary + 1 * step
VALUE_RATING_0  = left_boundary + 2 * step
VALUE_RATING_1  = left_boundary + 3 * step
VALUE_RATING_2  = left_boundary + 4 * step
VALUE_RATING_3  = left_boundary + 5 * step
VALUE_RATING_4  = left_boundary + 6 * step
VALUE_RATING_5  = left_boundary + 7 * step
VALUE_DEFAULT   = np.nan

df_user_data = pd.read_csv(
    filepath_or_buffer='/content/drive/MyDrive/BKP/children/children_user_data.csv',
    index_col='user_id'
)

df_corpus_annotations = pd.read_csv(
    filepath_or_buffer='/content/drive/MyDrive/BKP/children/children_corpus_annotation
s.csv',
    index_col='book_id'
)

```

```

)

filename_d2v = '/content/drive/MyDrive/BKP/children/children_model_d2v.d2v'
model_d2v = Doc2Vec.load(filename_d2v)
VECTOR_SIZE = model_d2v.vector_size

class UES(object):
    """
    USER EMBEDDING SPACE

    Класс, который описывает пространство эмбеддингов конкретного пользователя.
    Экземпляр содержит в себе набор эмбеддингов, учитывающих, как и объект,
    так и вид взаимодействия с ним, и набор средств анализа рассчитанного
    пространства.
    """

    def __init__(self, user_data):
        """
        Метод инициализации класса, в котором производится обращение к методам,
        формирующими собственно пространство эмбеддингов.

        Аргументы:
        - user_data (`pandas.Series`) - серия библиотеки Pandas, в которой
          содержится препроцессированная информация о взаимодействиях
          пользователя с книгами в виде категоризированного набора
          идентификаторов книг.
        """

        self.user_id = user_data.name
        # Датасет, в котором хранится все рассчитанные данные
        self.df = pd.DataFrame(columns=['id', 'embedding', 'interaction', 'type',
        'tokens'])
        # Формирование пространства эмбеддингов
        self.__construct_embedding_space(user_data)
        self.__compute_centers()

    def __construct_embedding_space(self, user_data):
        """
        Метод, при помощи которого происходит расчёт эмбеддингов: формирование
        эмбеддинга с помощью модели Doc2Vec и его последующая корректировка.

        Аргументы:
        - user_data (`pandas.Series`) - серия библиотеки Pandas, в которой
          содержится препроцессированная информация о взаимодействиях
          пользователя с книгами в виде категоризированного набора
          идентификаторов книг.
        """

        for interaction in interactions:
            # Идентификаторы книг текущего вида взаимодействия
            ids = re.findall('[0-9]+', user_data[interaction])
            for id in ids:
                # Если имеется аннотация рассматриваемой, происходит формирование
                # эмбеддинга
                try:
                    # Формирование изначального эмбеддинга

```

```

        text_tokens =
preprocess_string(df_corpus_annotations.loc[int(id)]['annotation'])
        text_embedding = model_d2v.infer_vector(text_tokens)
        # Добавление измерения, характеризующего вид взаимодействия
        interaction_embedding = np.append(text_embedding,
self.__get_interaction_value(interaction))
        # Запись полученного эмбеддинга в датасет пространства
        new_row = {'id': f'{interaction}_{id}',
        'embedding': interaction_embedding,
        'interaction': interaction,
        'type': 'object',
        'tokens': ' '.join(text_tokens)}
        self.df.loc[len(self.df)] = new_row
    except:
        continue
    self.df = self.df.set_index('id')

def __get_interaction_value(self, interaction):
    """
    Метод, который предоставляет значение дополнительной компоненты
    эмбеддинга в зависимости от вида взаимодействия пользователя с книгой.

    Аргументы:
    - interaction ('str') - вид взаимодействия пользователя с книгой.
    """

    match interaction:
        case 'shelved':
            return VALUE_SHELFED
        case 'read':
            return VALUE_READ
        case 'rating_0':
            return VALUE_RATING_0
        case 'rating_1':
            return VALUE_RATING_1
        case 'rating_2':
            return VALUE_RATING_2
        case 'rating_3':
            return VALUE_RATING_3
        case 'rating_4':
            return VALUE_RATING_4
        case 'rating_5':
            return VALUE_RATING_5
        case _:
            return VALUE_DEFAULT

def __compute_centers(self):
    """
    Метод, позволяющий получить центры (медоиды) кластеров взаимодействий
    при помощи алгоритма K-Medoids.

    Все вычисленные центры кластеров помечаются в датасете `self.df`.
    """

    for interaction in interactions:
        # Элементы кластера текущего вида взаимодействия
        interaction_data = self.df[self.df['interaction'] == interaction]

```

```

embeddings = interaction_data['embedding'].values.tolist()
# Если текущий кластер не является пустым, происходит поиск его центра
if not (len(embeddings) == 0):
    # Поиск центра кластера при помощи алгоритма K-Means
    kmedoids = KMedoids(n_clusters=1)
    kmedoids.fit(embeddings)
    center = kmedoids.cluster_centers_[0]
    # Отметка центра в датасете пространства
    for index, row in interaction_data.iterrows():
        if (np.array_equal(row['embedding'], center)):
            self.df.at[index, 'type'] = 'center'
            break

def retrieve_centers(self):
    """
    Метод, возвращающий раннее вычисленные центры кластеров взаимодействий
    в виде фрагмента датасета.

    Возвращает:
    - df (`pandas.DataFrame`) - датасет с центрами кластеров.
    """

    return self.df[self.df['type'] == 'center']

def show_tsne(self, is_text_embeddings=False):
    """
    Метод, при помощи которого происходит визуализация пространства
    эмбеддингов пользователя сниженной размерности при помощи
    алгоритма T-SNE.

    Аргументы:
    - is_text_embeddings (`bool`) - если аргумент равен `True`, то происходит
      визуализация исходного пространства эмбеддингов, иначе -
      пространства эмбеддингов с учётом вида взаимодействия.
    """

    # Размерность эмбеддингов определяет учёт взаимодействий
    embedding_size = VECTOR_SIZE if (is_text_embeddings) else VECTOR_SIZE + 1
    # Формирование итоговых эмбеддингов
    embeddings = np.zeros((len(self.df['embedding']), embedding_size))
    for i in range(len(self.df['embedding'])):
        embeddings[i,:] = np.array(
            self.df.iloc[i]['embedding'][:embedding_size]
        ).reshape((1, embedding_size))
    # Уменьшение размерности при помощи модели T-SNE
    perplexity = len(embeddings) - 2 if (len(embeddings) < 50) else 30
    model_tsne = TSNE(
        perplexity=perplexity, n_components=3, init='pca'
    )

    # Сохранение полученных значений в датасет пространства
    tsne_embeddings = model_tsne.fit_transform(embeddings)
    tsne_x = list(map(lambda x: x[0], tsne_embeddings))
    tsne_y = list(map(lambda x: x[1], tsne_embeddings))
    tsne_z = list(map(lambda x: x[2], tsne_embeddings))
    if (is_text_embeddings):
        self.df['tsne_x_text'] = tsne_x

```

```

        self.df['tsne_y_text'] = tsne_y
        self.df['tsne_z_text'] = tsne_z
    else:
        self.df['tsne_x_interaction'] = tsne_x
        self.df['tsne_y_interaction'] = tsne_y
        self.df['tsne_z_interaction'] = tsne_z

    # Визуализация полученных результатов
    if (is_text_embeddings):
        title = 'Визуализация UES при помощи метода T-SNE<br>(без учёта взаимодействий)'
        x = 'tsne_x_text'
        y = 'tsne_y_text'
        z = 'tsne_z_text'
    else:
        title = 'Визуализация UES при помощи метода T-SNE<br>(с учётом взаимодействий)'
        x = 'tsne_x_interaction'
        y = 'tsne_y_interaction'
        z = 'tsne_z_interaction'

    # Построение 3-ёх мерной диаграммы рассеяния
    fig_1 = px.scatter_3d(
        self.df, x=x, y=y, z=z, color='interaction', symbol='type',
        labels={'interaction': 'Вид взаимодействия', 'type': 'Тип объекта'}
    )
    fig_1.update_layout(
        height=470, width=800, title=dict(text=title, font=dict(size=18)),
        scene=dict(xaxis_title='X', yaxis_title='Y', zaxis_title='Z')
    )
    fig_1.show()

    # Построение матрицы рассеяния
    fig_2 = px.scatter_matrix(
        self.df, dimensions=[x, y, z], color='interaction',
        labels={'interaction': "Вид взаимодействия", x: 'X', y: 'Y', z: 'Z'}
    )
    fig_2.update_layout(
        height=470, width=800,
        title=dict(text='Матрица диаграмм рассеяния', font=dict(size=18))
    )
    fig_2.show()

def show_wordclouds(self):
    """
    Метод, визуализирующий кластеры взаимодействий посредством
    формирования облака слов.
    """

    fig = make_subplots(
        rows=3, cols=3,
        subplot_titles=('Shelved', 'Read', 'Rating - 0', 'Rating - 1',
                       'Rating - 2', 'Rating - 3', 'Rating - 4', 'Rating - 5')
    )
    index = 0
    for interaction in interactions:
        row_index = int(index / 3 + 1)
        col_index = int(index % 3 + 1)
        if not (len(self.df[self.df['interaction']] == interaction]) == 0:
            # Если текущий кластер не пустой, происходит построение облака слов

```

```
text = ' '.join(self.df[self.df['interaction'] == interaction]['tokens'])
wordcloud = WordCloud(
    width=500, height=500, min_font_size=10, background_color="white"
).generate(text)
fig.add_trace(px.imshow(wordcloud).data[0], row=row_index,
col=col_index)
fig.update_xaxes(visible=False, row=row_index, col=col_index)
fig.update_yaxes(visible=False, row=row_index, col=col_index)
else:
    # Если текущий кластер пустой, происходит вывод пустого облака слов
    wordcloud = WordCloud(
        width=500, height=500, min_font_size=10, background_color="white"
    ).generate('Empty')
    fig.add_trace(px.imshow(wordcloud).data[0], row=row_index,
col=col_index)
    fig.update_xaxes(visible=False, row=row_index, col=col_index)
    fig.update_yaxes(visible=False, row=row_index, col=col_index)
    index += 1
fig.update_layout(height=900, width=850,
                  title=dict(text='Облака слов в зависимости от реакции
пользователя',
font=dict(size=18)))
fig.show()
```

ПРИЛОЖЕНИЕ 2. КОД МОДЕЛИ ВЕКТОРИЗАЦИИ ПОЛЬЗОВАТЕЛЬСКИХ ПРЕДПОЧТЕНИЙ

```

from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import pandas as pd

import re
import collections
import datetime

from gensim.parsing.preprocessing import preprocess_string
from gensim.models.doc2vec import Doc2Vec

from sklearn.manifold import TSNE
from sklearn_extra.cluster import KMedoids
from sklearn.metrics.pairwise import cosine_similarity

import plotly.express as px
from plotly.subplots import make_subplots
from wordcloud import WordCloud

target_interactions = ['read', 'rating_4', 'rating_5']

df_user_data = pd.read_csv(
    filepath_or_buffer='/content/drive/MyDrive/BKP/children/children_user_data.csv',
    index_col='user_id'
)

df_corpus_annotations = pd.read_csv(
    filepath_or_buffer='/content/drive/MyDrive/BKP/children/children_corpus_annotations.csv',
    index_col='book_id'
)

filename_d2v = '/content/drive/MyDrive/BKP/children/children_model_d2v.d2v'
model_d2v = Doc2Vec.load(filename_d2v)
VECTOR_SIZE = model_d2v.vector_size

```

```

class Pref2Vec(object):
    """
PREF2VEC

Модель Pref2Vec - модель векторизации пользовательских предпочтений Pref2Vec.
Модель позволяет построить множество пользовательских пространств эмбеддингов
- корпус объектов UES, представленных в виде набора вычисленных центров
кластеров взаимодействий. На основе сформированного корпуса открывается
возможность исследования пользовательских пространств эмбеддингов, которое
позволяет строить рекомендации для конечного пользователя посредством поиска
схожих пользовательских предпочтений, то есть схожих пространств. Дальнейшая
рекомендация объектов происходит при помощи анализа расширенных эмбеддингов
наиболее подобных пользовательских пространств.

"""

def __init__(self, depth):
    """
Метод инициализации класса, в котором производится определение полей
`self.depth` и `self.corpus`.

Аргументы:
- depth (`int`) -
"""

    self.depth = depth
    self.corpus = pd.DataFrame(
        columns=['user_id', 'center_read', 'center_shelved',
                 'center_rating_0', 'center_rating_1', 'center_rating_2',
                 'center_rating_3', 'center_rating_4', 'center_rating_5']
    )

def build_corpus(self, filepath_or_buffer, users_data):
    """
Метод, при помощи которого производится построение корпуса
пользовательских предпочтений.

Для каждого пользователя рассчитывается пространство эмбеддингов (UES),
далее полученные центры кластеров взаимодействий сохраняются в корпус
модели.

Аргументы:
- users_data (`pandas.DataFrame`)
"""

```

```

counter = 0
for index, row in users_data.iterrows():
    # Расчёт проистранства эмбеддингов
    ues = UES(row)
    # Получение центров кластаров взаимодействий
    centers = ues.retrieve_centers()
    # Добавление данных в корпус
    new_row = {'user_id': ues.user_id}
    for interaction in interactions:
        new_row[f"center_{interaction}"] = self.__get_element_or_nan(
            array=centers['interaction']
            == interaction]['embedding'].values,
            index=0
        )
    self.corpus.loc[len(self.corpus)] = new_row

    counter += 1
    if (counter % 250 == 0):
        self.__save_corpus(filepath_or_buffer)

self.corpus = self.corpus.set_index('user_id')
self.__save_corpus(filepath_or_buffer)

def __save_corpus(self, filepath_or_buffer):
    """
    Метод, который сохраняет корпус модели в CSV-файл.

    Аргументы:
    - filepath_or_buffer (`str`) - путь файла, в который произойдёт
      сохранение файла.
    """

    self.corpus.to_csv(filepath_or_buffer)

def load_corpus(self, filepath_or_buffer):
    """
    Метод, позволяющий загрузить корпус модели из CSV-файла.

    Аргументы:
    - filepath_or_buffer (`str`) - путь к используемому CSV-файлу.
    """

    users_data = pd.read_csv(
        filepath_or_buffer=filepath_or_buffer,

```

```

        index_col='user_id'
    )
# Преобразование строковых значений датасета в значения с плавающей запятой
for user_id in users_data.index.values:
    for interaction in interactions:
        try:
            if not (pd.isna(users_data.loc[user_id][f"center_{interaction}"])):
                center = np.asarray(
                    np.array(re.findall('[-+0-9.e]+',
users_data.loc[user_id][f"center_{interaction}"]),
                           dtype=float
                )
                users_data.at[user_id, f"center_{interaction}"] = center
        except:
            continue
self.corpus = users_data

```

`def most_similar_users(self, target_user_data, topn=10, show_info=True):`

`'''`

Метод, при помощи которого происходит поиск ближайших `topn` пользователей к целевому поиску.

Поиск осуществляется рассчётом косинусных расстояний между центрами кластеров категорий искомого пользователя и соответствующими центрами пользователей корпуса модели `self.corpus`. Для каждой из категорий происходит рассчёт ближайших `self.depth` пользователей. Далее выводятся самые близкие пользователи к целевому.

Аргументы:

- `target_user_data` (`pandas.Series`) - серия библиотеки Pandas, в которой содержится препроцессированная информация о взаимодействиях целевого пользователя с книгами в виде категоризированного набора идентификаторов книг;
- `topn` (`int`) - количество ближайших пользователей;
- `show_info` (`bool`) - флаг вывода информации о работе метода.

Возвращает:

- `most_similar_users` (`list`) - список с ближайшими пользователями. Содержит набор объектов `dict`, в которых хранятся идентификатор пользователя, категория в которой пользователь оказался близок и косинусное расстояние с центром категории целевого пользователя.

`'''`

```
# Датасет поиска ближайших пользователей
self.df_most_similar_users = pd.DataFrame(
```

```

        columns=['user_id', 'interaction', 'interaction_index', 'similarity']
    )
    # Рассчёт целевого пользовательского пространства эмбеддингов
    target_ues = UES(target_user_data)
    target_user_id = target_ues.user_id
    target_centers = target_ues.retrieve_centers()

    # Поиск ближайших центров по видам взаимодействий
    for interaction in interactions:
        self.__most_similar_users_by_interaction(
            target_user_id=target_user_id,
            target_center=target_centers[target_centers['interaction'] == interaction]['embedding'].values,
            interaction=interaction
        )

    # Группировка и агрегирование полученный данных
    df_most_similar_users_aggregated =
        self.df_most_similar_users.groupby('user_id').agg({'interaction': 'count',
        'interaction_index': 'mean'})
    df_most_similar_user_top_n =
        df_most_similar_users_aggregated.sort_values(by=['interaction', 'interaction_index'],
        ascending=[False, True]).head(topn)
    most_similar_users = list()
    for user_id in df_most_similar_user_top_n.index.values:
        df_current_user =
            self.df_most_similar_users[self.df_most_similar_users['user_id'] == user_id]
        dict_current_user = {'user_id': user_id, 'appearances': []}
        for index, row in df_current_user.iterrows():
            dict_current_user['appearances'].append((row['interaction'],
            row['similarity']))
        most_similar_users.append(dict_current_user)

    # Вывод результатов
    if (show_info):
        print(f"Ближайшие пользователи к {target_user_id}:")
        for i in range(len(most_similar_users)):
            print(f"{i+1}) Пользователь №{most_similar_users[i]['user_id']}:")
            for appearance in most_similar_users[i]['appearances']:
                print(f"    {appearance[0]} - {appearance[1]}")

    return most_similar_users

def __most_similar_users_by_interaction(self, target_user_id, target_center,
interaction):
    """
    """

```

Метод, осуществляющий рассчёт косинусных расстояний между искомым центром категории и всеми остальными центрами категории корпуса `self.corpus`.

Расстояние вычисляется при помощи модуля `metrics.pairwise.cosine_similarity` библиотеки `sklearn`. Для увеличения скорости вычислений промежуточные результаты сохраняются и обрабатываются при помощи библиотеки `pandas`.

Аргументы:

- target_user_id (`str`) - идентификатор целевого пользователя;
 - target_center (`list`) - массив, содержащий целевой центр кластера текущего взаимодействия;
 - interaction ('str') - рассматриваемое взаимодействие.
- '''

```
# Если у пользователя нет эмбеддингов текущего взаимодействия, ближайших
пользователей нет
if (target_center.size == 0):
    return

# Инициализация датасета, в котором будут рассчитываться расстояния
df_similarity = pd.DataFrame(data={'user_id': self.corpus.index.values,
'center': self.corpus[f"center_{interaction}"]})

# Фильтрация элементов, в которых нет центра рассматриваемой категории
df_similarity = df_similarity[df_similarity['center'].notna()]

# Вычисление расстояний между искомым центром и всеми остальными
target_center = [target_center[0]] if (target_center.size > 1) else
list(target_center)

similarities = cosine_similarity(target_center, list(df_similarity['center']))
df_similarity['similarity'] = np.reshape(similarities,
df_similarity['center'].size)

# Обработка случая, в котором найденное расстояние - расстояние с искомым центром
if (df_similarity['similarity'].idxmax() == target_user_id):
    df_similarity = df_similarity.drop(df_similarity['similarity'].idxmax())

# Сохранение ближайших пользователей в датасет self.df_most_similar_users
df_most_similar_users = pd.DataFrame(
    data={
        'user_id':
df_similarity['similarity'].nlargest(self.depth).index.values,
        'interaction': [interaction] * self.depth,
        'interaction_index': [i+1 for i in range(self.depth)],
        'similarity': df_similarity['similarity'].nlargest(self.depth).values
    }
)
self.df_most_similar_users = pd.concat([self.df_most_similar_users,
df_most_similar_users])
```

```
def recommend_for_user(self, target_user_data, topn=50, show_info=True):
    """

```

Метод, при помощи которого происходит рассчёт `topn` рекомендаций для целевого пользователя.

Сначала рассчитываются 10 ближайших пользователей к целевому. Далее рассматриваются все объекты ближайший пользователей на предмет косинусного расстояния к объектам целевого пользоваля – происходит рассчёт расстояний между рассматриваемым объектом и всеми целевыми, наименьшее расстояние сохраняется. После происходит сортировка объектов по полученным расстояниям и вывод `topn` ближайших.

Аргументы:

- `target_user_data` (`pandas.Series`) – серия библиотеки Pandas, в которой содержится препроцессированная информация о взаимодействиях целевого пользователя с книгами в виде категоризированного набора идентификаторов книг;
- `topn` (`int`) – количество ближайших пользователей;
- `show_info` (`bool`) – флаг вывода информации о работе метода.

Возвращает:

- `recommendations` (`list`) – список с полученными рекомендациями.
Содержит пары с идентификатором рекомендуемого объекта и ближайшим расстоянием.

"""

```
# Рассчёт целевого пользовательского пространства эмбеддингов
ues_target = UES(target_user_data)
ues_target.df
ues_target.df[ues_target.df['interaction'].isin(target_interactions)]
# Датасет, в котором будут рассчитываться косинусные расстояния
self.df_embeoggings_target = pd.DataFrame(
    data={'embedding': ues_target.df['embedding'].values},
    index=[re.findall('^[0-9][0-9]+$', index)[0] for index in
           ues_target.df.index.values]
)

# Поиск ближайших пользователей
most_similar_users = self.most_similar_users(target_user_data, show_info=False)
most_similar_users_ids = [x['user_id'] for x in most_similar_users]
# Датасет с расстояниями всех объектов с искомыми
self.df_recommendations = pd.DataFrame(columns=['book_id', 'similarity'])
# Прогон объектов всех ближайших пользователей
for user_id in most_similar_users_ids:
```

```

ues_near = UES(df_user_data.loc[user_id])
ues_near.df
ues_near.df[ues_near.df['interaction'].isin(target_interactions)] =
    # Вычисление ближайшего расстояния для каждого объекта с целевыми объектами
for index, embedding in ues_near.df['embedding'].items():
    nearest_similarity = self.__get_nearest_similarity(
        embedding=embedding,
        embedding_id=re.findall('[0-9][0-9]+', index)[0]
    )
    self.df_recommendations.loc[len(self.df_recommendations.index)] = {
        'book_id': re.findall('[0-9][0-9]+', index)[0],
        'similarity': nearest_similarity
    }

# Сортировка объектов и вывод `topn` ближайших
self.df_recommendations = self.df_recommendations.drop_duplicates('book_id',
keep='first')
self.df_recommendations = self.df_recommendations.sort_values(by='similarity',
ascending=False).head(topn)
recommendations = []
for index, row in self.df_recommendations.iterrows():
    recommendations.append((row['book_id'], row['similarity']))

# Вывод результатов
if (show_info):
    print(f"Рекомендации для пользователя {ues_target.user_id}:")
    counter = 1
    for index, row in self.df_recommendations.iterrows():
        print(f"{counter}) Книга #{row['book_id']} - {row['similarity']}")
        counter += 1

return recommendations

def __get_nearest_similarity(self, embedding, embedding_id):
    """
    Метод, который позволяет найти ближайшее косинусное расстояние к одному
    из целевых объектов датасета `self.df_embeddings_target`.
    """

    Аргументы:
    - embedding (`list`) - эмбеддинг рассматриваемого объекта;
    - embedding_id (`str`) - идентификатор рассматриваемого объекта.
    """

    # Рассчитывает расстояние между рассматриваемым объектом и целевыми

```

```

similarities           = cosine_similarity([embedding],
list(self.df_embeddings_target['embedding']))
self.df_embeddings_target['similarity']      = np.reshape(similarities,
self.df_embeddings_target['embedding'].size)

# Обработка случая, в котором найденное расстояние - расстояние с искомым объектом
if (self.df_embeddings_target['similarity'].idxmax() == embedding_id):

self.df_embeddings_target.drop(self.df_embeddings_target['similarity'].idxmax())

return self.df_embeddings_target['similarity'].max()

def __get_element_or_nan(self, array, index):
    """
    Метод, который возвращает элемент массива, если его адрес находится
    в пределах массива. В ином случае возвращается элемент `np.nan`.

    Аргументы:
    - array (`array-like`) - массив, из которого необходимо получить элемент;
    - index (`int`) - индекс получаемого элемента.
    """

    try:
        return array[index]
    except IndexError:
        return np.nan

```

ПРИЛОЖЕНИЕ 3. КОД КЛАССА ОЦЕНКИ РАБОТЫ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ

```

from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import pandas as pd

import re
import collections
import datetime

from gensim.parsing.preprocessing import preprocess_string
from gensim.models.doc2vec import Doc2Vec

from sklearn.manifold import TSNE
from sklearn_extra.cluster import KMedoids
from sklearn.metrics.pairwise import cosine_similarity

import plotly.express as px
from plotly.subplots import make_subplots
from wordcloud import WordCloud

target_interactions = ['read', 'rating_4', 'rating_5']

df_user_data = pd.read_csv(
    filepath_or_buffer='/content/drive/MyDrive/BKP/children/children_user_data.csv',
    index_col='user_id'
)
df_corpus_annotations = pd.read_csv(
    filepath_or_buffer='/content/drive/MyDrive/BKP/children/children_corpus_annotations.csv',
    index_col='book_id'
)

filename_d2v = '/content/drive/MyDrive/BKP/children/children_model_d2v.d2v'
model_d2v = Doc2Vec.load(filename_d2v)
VECTOR_SIZE = model_d2v.vector_size

class Evaluator:

```

```
'''  
RECOMMENDER SYSTEM EVALUATOR
```

Класс Evaluator позволяет провести тестирование работы рекомендательной системы на предмет качества формирования списков рекомендаций. Также этот класс позволяет произвести подготовку тестовых данных (данных о взаимодействиях пользователей с различными объектами).

```
'''
```

```
def __init__(self, model_p2v, test_percent):
```

```
'''
```

Метод инициализации класса, в котором производится инициализация полей self.model_p2v (модель Pref2Vec) и self.test_percent (процент разбиения данных на выборки X и Y).

Аргументы:

- model_p2v ('Pref2Vec') - модель Pref2Vec, работа которой будет тестироваться;
- test_percent ('int') - процент разбиения данных на выборки X и Y.

```
'''
```

```
    self.model_p2v = model_p2v
```

```
    self.test_percent = test_percent
```

```
def test(self, users_data, filepath_or_buffer, k=20):
```

```
'''
```

Метод, который производит тестирование модели `model_p2v` на пользовательских данных `users_data`.

Используются метрики NDCD@k, Recall@k и Precision@k. Результаты тестирования сохраняются в поле `self.df_testing`. Кроме того, имеется возможность загрузки результатов в CSV-файл.

Аргументы:

- users_data ('pandas.DataFrame') - пользовательские данные, используемые при тестировании рекомендательной системы;
- filepath_or_buffer ('str') - путь к CSV-файлу, в который будут сохранены результаты тестирования;
- k ('int') - параметр 'k', используемый в метриках NDCD@k, Recall@k и Precision@k.

```
'''
```

```
    self.df_testing = pd.DataFrame(columns=['user_id', f"ndcg@{k}", f"recall@{k}",  
f"precision@{k}"])
```

```

self.k = k

counter = 1
for index, row in users_data.iterrows():
    try:
        # Разделение пользовательских данных
        x, y, y_true = self.__sample_preparation(index, row)
        # Получение рекомендаций для текущего пользователя
        recommendations
    model_p2v.recommend_for_user(target_user_data=x.iloc[0], topn=k, show_info=False)
        y_pred = [(lambda x: x[0])(x) for x in recommendations]
        # Тестирование и запись результатов в self.df_testing
        ndcg = self.ndcg_at_k(r=[x in y_true for x in y_pred])
        recall = self.recall_at_k(y_true, y_pred)
        precision = self.precision_at_k(y_true, y_pred)
        print(f"{counter}) User {index}:")
        print(f"    ndcg@{k}      = {ndcg}")
        print(f"    recall@{k}     = {recall}")
        print(f"    precision@{k} = {precision}")
        self.df_testing.loc[len(self.df_testing)] = {
            'user_id': index,
            f"ndcg@{k)": ndcg,
            f"recall@{k)": recall,
            f"precision@{k)": precision
        }
        # Периодическая запись CSV-файла с результатами тестирования
        counter += 1
        if (counter % 5 == 0):
            self.df_testing.to_csv(filepath_or_buffer)
    except:
        continue

# Запись CSV-файла с результатами тестирования
self.df_testing.to_csv(filepath_or_buffer)

```

```
def __sample_preparation(self, user_id, user_data):
    """

```

Метод разбиения тестовых данных на выборки X и Y и ранжирования выборки Y.

Аргументы:

- user_id ('str') - идентификатор пользователя, чьи данные будут подвергены разбиению;
- user_data ('pandas.Series') - собственно данные, которые будут подвергнуты разбиению.

```

Возвращаются:
- self.x ('list') - формируемая выборка X;
- self.y ('list') - формируемая выборка Y;
- self.y_true ('list') - ранжированная выборка Y;
"""

self.x = pd.DataFrame(columns=['user_id', 'read', 'shelved', 'rating_0',
'rating_1', 'rating_2', 'rating_3', 'rating_4', 'rating_5'])
self.y = pd.DataFrame(columns=['user_id', 'read', 'shelved', 'rating_0',
'rating_1', 'rating_2', 'rating_3', 'rating_4', 'rating_5'])
row_x = {'user_id': user_id}
row_y = {'user_id': user_id}

# Разделение пользовательских данных на выборки X и Y
for interaction in interactions:
    ids = re.findall('[0-9]+', user_data[interaction])
    x_size = int(len(ids) * (100 - self.test_percent) / 100)
    if (interaction in target_interactions):
        row_x[interaction] = str(ids[:x_size])
        row_y[interaction] = str(ids[x_size:])
    else:
        row_x[interaction] = str(ids)
        row_y[interaction] = str([])
self.x.loc[0] = row_x
self.y.loc[0] = row_y

# Ранжирование выборки Y
self.y_true = self.__get_y_true(self.x.iloc[0], self.y.iloc[0])

return (self.x, self.y, self.y_true)

```

```
def __get_y_true(self, x, y):
    """

```

Метод, осуществляющий ранжирование рассматриваемой выборки Y.

Вычисляется косинусное сходство каждого объекта выборки Y с каждым эмбеддингом выборки X, сохранив максимальные значения сходства. Далее элементы выборки Y ранжируются по вычисленным сходствам и формируется список `y_true`.

Аргументы:

- x ('pandas.Series') - выборка X;
- y ('pandas.Series') - выборка Y.

Возвращается:

```

- self.df_embeddings_y.index.values - ('list') - ранжированный список
данных выборки Y.
'''

# Формирование целевого пользовательского пространства X
ues_x = UES(x)
ues_x.df = ues_x.df[ues_x.df['interaction'].isin(target_interactions)]
self.df_embeddings_x = pd.DataFrame(
    data={'embedding': ues_x.df['embedding'].values},
    index=re.findall('[0-9][0-9]+', str(ues_x.df.index.values)))
)

# Формирование исследуемого пользовательского пространства Y
ues_y = UES(y)
ues_y.df = ues_y.df[ues_y.df['interaction'].isin(target_interactions)]
self.df_embeddings_y = pd.DataFrame(
    data={'embedding': ues_y.df['embedding'].values, 'distance': [0] * len(ues_y.df['embedding'].values)},
    index=re.findall('[0-9][0-9]+', str(ues_y.df.index.values)))
)

# Поиск наиболее схожих элементов в пространстве Y с элементами пространства X
for index, row in self.df_embeddings_y.iterrows():
    self.df_embeddings_y.loc[index, 'distance'] = self.__get_nearest_distance(
        embedding=row['embedding'],
        embedding_id=index
    )
self.df_embeddings_y = self.df_embeddings_y.sort_values(by='distance',
ascending=False).head(self.k)

return self.df_embeddings_y.index.values

```

```
def __get_nearest_distance(self, embedding, embedding_id):
'''
```

Метод, при помощи которого вычисляется косинусное сходство эмбеддинга объекта выборки Y с каждым эмбеддингом выборки X. Выводит максимальное значение сходства.

Аргументы:

- embedding ('list') - эмбеддинг рассматриваемого объекта;
- embedding_id ('str') - индентификатор рассматриваемого объекта.

Возвращается:

- self.df_embeddings_x['distance'].max() ('float') - максимальное значение сходства.

```
'''
```

```

# Рассчитывает расстояние между рассматриваемым объектом и целевыми
distances = cosine_similarity([embedding],
list(self.df_embeddings_x['embedding']))
self.df_embeddings_x['distance'] = np.reshape(distances,
self.df_embeddings_x['embedding'].size)

# Обработка случая, в котором найденное расстояние - расстояние с искомым объектом
if (self.df_embeddings_x['distance'].idxmax() == embedding_id):
    self.df_embeddings_x.drop(self.df_embeddings_x['distance'].idxmax())

return self.df_embeddings_x['distance'].max()

```

def precision_at_k(self, y_true, y_pred):

'''

Метод, позволяющий получить значение метрики Precision@k.

Аргументы:

- y_true ('list') - эталонный список рекомендаций;
- y_pred ('list') - список рекомендаций, формируемый рекомендательной системой;

Возвращает:

- precision ('float') - значение метрики Precision@k

'''

```

set_y_true = set(y_true)
set_y_pred = set(y_pred[:self.k])

```

```
precision = len(set_y_true & set_y_pred) / float(self.k)
```

```
return precision
```

def recall_at_k(self, y_true, y_pred):

'''

Метод, позволяющий получить значение метрики Recall@k.

Аргументы:

- y_true ('list') - эталонный список рекомендаций;
- y_pred ('list') - список рекомендаций, формируемый рекомендательной системой;

Возвращает:

- recall ('float') - значение метрики Recall@k

'''

```

    set_y_true = set(y_true)
    set_y_pred = set(y_pred[:self.k])

    recall = len(set_y_true & set_y_pred) / float(len(set_y_true))

    return recall

def dcg_at_k(self, r, method=0):
    """
    Метод, позволяющий получить значение метрики DCG@k.

    Аргументы:
    - y_true ('list') - эталонный список рекомендаций;
    - y_pred ('list') - список рекомендаций, формируемый
        рекомендательной системой;

    Возвращает:
    - dcg ('float') - значение метрики DCG@k
    """

    r = np.asarray(r)[:self.k]

    if r.size:
        if method == 0:
            return r[0] + np.sum(r[1:] / np.log2(np.arange(2, r.size + 1)))
        elif method == 1:
            return np.sum(r / np.log2(np.arange(2, r.size + 2)))
        else:
            raise ValueError('method must be 0 or 1.')
    else:
        return 0.

def ndcg_at_k(self, r, method=0):
    """
    Метод, позволяющий получить значение метрики NDCG@k.

    Аргументы:
    - y_true ('list') - эталонный список рекомендаций;
    - y_pred ('list') - список рекомендаций, формируемый
        рекомендательной системой;

    Возвращает:
    - ndcg ('float') - значение метрики NDCG@k
    """

```

```
'''  
  
    dcg_max = self.dcg_at_k(sorted(r, reverse=True), method)  
  
    if not dcg_max:  
        return 0.  
  
    return self.dcg_at_k(r, method) / dcg_max
```