МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

НОВОСТНАЯ РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА

Студент 25.05.2026 Кирюткин Илья Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель 25.05.2026 Токарев А.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 25.05.2026 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент зав. каф. Программирование ПензГТУ 27.05.26 Мартышкин А.И.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.26 № 14)

Заведующий кафедрой _____ М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 04.06.2026 №1)

Секретарь ГЭК _____ Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«20» 09 2024 г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент Кирюткин Илья Алексеевич гр.24ВВИм1 факультета ВТ

2.Тема работы «Новостная рекомендательная система»

Тема утверждена приказом ПГУ № 920/0 от "20" сентября 2024г.

3. Руководитель магистерской диссертации к.т.н., доцент Токарев А.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: разработка новостной рекомендательной системы, обеспечивающей персонализированную выдачу новостного контента

5. Перечень подлежащих разработке вопросов

1. Анализ предметной области и обзор аналогов.
2. Выбор алгоритмов для тестирования.
3. Исследование возможности определения эмоций в новостных текстах
4. Разработка модуля тегирования новостей.
5. Реализация и тестирование выбранных алгоритмов.
6. Сравнение алгоритмов по метрикам качества.
7. Анализ влияния тегов на рекомендации.
8. Выводы по результатам работы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	11%	29.11.2024-29.01.2025	
2.	Анализ требований	11%	29.01.2025-30.03.2025	
3.	Выбор технологий, применяемых при проектировании и разработки	22%	30.03.2025-30.05.2025	
4.	Проектирование и реализация рекомендательных алгоритмов	27%	01.06.2025-01.09.2025	
5.	Тестирование	17%	02.09.2025-02.10.2025	
6.	Анализ полученных результатов исследования	6%	01.04.2026-29.04.2026	
7.	Оформление пояснительной записки	6%	30.04.2026-25.05.2026	

Дата выдачи задания "20" 09 2024 г.

Руководитель магистерской диссертации _____ Токарев А.Н.
(подпись, дата)

Задание к исполнению принял студент _____ Кирюткин И.А.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Фионова Л.Р.
(подпись, дата)
01.06.2026

РЕФЕРАТ

Магистерская работа содержит 96 листов, 4 таблицы, 11 рисунков, 14 источников, 2 приложения.

РЕКОМЕНДАТЕЛЬНЫЕ АЛГОРИТМЫ, НАИВНЫЙ БАЙЕСОВСКИЙ КЛАССИФИКАТОР, НОВОСТНАЯ РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА, BERT, ALS

Объектом исследования является процесс персонализированной рекомендации новостного контента.

Предметом исследования являются алгоритмы классификации текстов, а также рекомендательные алгоритмы.

Целью данной работы является исследование методов повышения качества рекомендаций новостного контента.

В ходе выполнения работы был проведён анализ существующих новостных рекомендательных систем, определен оптимальный алгоритм автоматического тегирования новостей, а также проведено исследование возможности определения эмоциональной окраски новостных текстов. Выполнено сравнение рекомендательных алгоритмов, включая коллаборативную фильтрацию, контентные алгоритмы и алгоритм на основе модели BERT.

В результате проделанной работы установлено, что использование тегов положительно влияет на качество ранжирования рекомендаций, наилучшие результаты среди исследованных алгоритмов показал метод ALS в сочетании с теговой разметкой.

Научная новизна работы заключается в исследовании методов, позволяющих повысить качество рекомендаций новостного контента

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 5
ВВЕДЕНИЕ 6
1 Анализ предметной области и обзор существующих решений 8
1.1 Актуальность 8
1.2 Обзор аналогов 9
1.2.1 Google News 10
1.2.2 Новости LifeLines 11
1.2.3 ЯRUS 11
1.2.4 Сравнение рассмотренных аналогов 12
1.3 Обзор и сравнение существующих типов рекомендательных систем . 13
1.4 Проблема выбора информативных признаков в новостных рекомендательных системах 15
1.5 Постановка задачи 16
2 Разработка модуля извлечения тегов 18
2.1 Разработка парсера новостей 18
2.2 Предобработка тестовых данных 22
2.3 Сравнение алгоритмов определения тегов 22
2.4 Выбор оптимального метода 25
2.5 Выводы по результатам автоматического тегирования новостей 27
3 Способы определения эмоций в тексте 30
3.1 Описание и подготовка тестовых данных 31
3.2 Обзор алгоритмов классификации 33
3.1.1 Наивный Байесовский классификатор 33
3.1.2 Логистическая регрессия 33
3.1.3 Случайный лес 34
3.1.4 LightGBM 35
3.1.5 XGBoost 35
3.1.6 CatBoost 36
3.2 Тестирование 36

3.3 Интерпретация результатов 41
3.4 Выводы по исследованию определения эмоций в новостных текстах. 42
4 Разработка и сравнение моделей ранжирования 45
4.1 Обзор рекомендательных алгоритмов RecTools 45
4.1.1 Модель BERT для ранжирования новостей 46
4.1.2 Неперсонализированные модели 47
4.1.3 Методы матричной факторизации 48
4.1.4 Методы ближайших соседей 49
4.1.5 Нейросетевые и гибридные модели 49
4.2 Описание датасета MIND 50
4.3 Метрики сравнения 50
4.4 Результаты тестирования 51
4.5 Сравнение эффективности ALS на данных с тегами и без тегов 55
4.6 Выводы по результатам сравнения моделей ранжирования 56
5 Построение итоговой рекомендательной модели 60
ЗАКЛЮЧЕНИЕ 62
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 63
ПРИЛОЖЕНИЕ А Листинг программы 66
ПРИЛОЖЕНИЕ В Презентация 91

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

ИИ – искусственный интеллект

ALS (от англ. Alternating Least Squares) – метод чередующихся наименьших квадратов

BERT (от англ. Bidirectional Encoder Representations from Transformers) двунаправленная модель представления текста на основе трансформеров

CatBoost (от англ. Categorical Boosting) – это высокопроизводительная библиотека машинного обучения с открытым исходным кодом

CTR (от англ. Click-Through Rate) – показатель кликабельности

DSSM (от англ. Deep Structured Semantic Model) – глубинная структурированная семантическая модель

KNN (от англ. K-Nearest Neighbors) – метод k-ближайших соседей.

LightGBM (от англ. Light Gradient Boosting Machine) высокопроизводительный фреймворк для градиентного бустинга с открытым исходным кодом от Microsoft.

MAP@K (от англ. Mean Average Precision) – средняя точность на K позиций

MIND (от англ. Microsoft News Dataset) –набор данных Microsoft для новостных рекомендаций

MIND (от англ. Microsoft News Dataset) –набор данных Microsoft для новостных рекомендаций

PureSVD (от англ. Pure Singular Value Decomposition) – сингулярное разложение матрицы

TF-IDF (от англ. Term Frequency – Inverse Document Frequency) - частотный метод оценки важности слова в документе

XGBoost (от англ. eXtreme Gradient Boosting) – сверхбыстрая и масштабируемая библиотека машинного обучения с открытым исходным кодом, реализующая алгоритм градиентного бустинга над деревьями решений.

ВВЕДЕНИЕ

Ежедневно миллионы пользователей по всему миру потребляют новостной контент через интернет-платформы, мобильные приложения и социальные сети, ожидая от систем быстрой и релевантной выдачи информации, соответствующей их интересам. Согласно ежегодному исследование Оксфордского университета Digital News Report за 2025 год, 39% читателей осознанно избегают чтение новостных источников из-за негативного влияния на эмоциональное состояние. Это приводит к снижению вовлеченности аудитории и потерям для медиа-платформ, которые не могут эффективно монетизировать трафик.

Одним из перспективных направлений в области информационного поиска и персонализации контента является создание интеллектуальных рекомендательных систем на основе методов машинного обучения и обработки естественного языка. Такие системы способны автоматически анализировать поведение пользователей, выделять ключевые темы новостей, сопоставлять их с профилями интересов и формировать персонализированные ленты.

Объект исследования: процесс персонализированной рекомендации новостного контента на основе поведенческих данных пользователей и семантического анализа текстов.

Предмет исследования: алгоритмы и методы автоматической разметки новостей (тегирование), классификации эмоций в тексте, а также коллаборативной и контентной фильтрации для ранжирования новостных лент.

Цель исследования: В данном исследовании рассматривается этап подготовки данных, включающий текстовую разметку с помощью тегов и аннотирование эмоционального тона новостных текстов, а также проводится сравнение рекомендательных алгоритмов, включая подход, основанный на семантическом представлении текста для улучшения рекомендаций. Изучается, как указанные методы разметки влияют на последующее качество

работы рекомендательных алгоритмов, и какой из алгоритмов показывает наилучшие результаты для новостного контента.

Задачи исследования:

– анализ предметной области и обзор существующих аналогов,

– исследование возможности определения эмоций в новостных текстах,

– разработка модуля автоматической разметки новостей (тегирование) с выбором оптимального алгоритма,

– сравнительный анализ рекомендательных алгоритмов.

Актуальность (научная новизна) исследования: исследование подходов для улучшения качества новостных рекомендаций, включая автоматическое тегирование новостных статей, гибридное ранжирование на основе коллаборативной фильтрации и семантического анализа текста, а также учёт времени просмотра как метрики вовлечённости. В работе выявлено расхождение между теоретическими моделями выбора типа рекомендательной системы и экспериментальными результатами, а также экспериментально доказана ограниченная применимость методов классификации эмоций по тексту новости для целей персонализации.

Практическая и научная значимость: полученные результаты и выводы могут быть использованы при разработке новостной рекомендательной системы, способной улучшить качество персонализации.

Границы исследования: основной акцент ставится на реализации модуля автоматического тегирования новостей, сравнении рекомендательных алгоритмов. В границы исследования не входят вопросы построения поискового индекса, распределённого хранения данных, а также обработка видеоконтента и мультимедийных новостей. Исследование эмоционального анализа ограничено проверкой гипотезы о возможности определения эмоций по тексту новости; разработка поведенческих моделей эмоций на основе времени просмотра остаётся за рамками работы.

1 Анализ предметной области и обзор существующих решений

1.1 Актуальность

Современный мир отличается высокой долей информации, поступающей к человеку. Каждый человек в той или иной степени является потребителем информационных материалов: статей, заголовков, телевизионных новостей. Люди так или иначе вовлечены в этот процесс. Современный мир характеризуется беспрецедентным ростом объёмов информации, поступающей к человеку из различных источников. Это явление, часто называемое «информационным взрывом» или «информационной перегрузкой» (information overload), впервые было описано ещё в 1970-х годах будущим лауреатом Нобелевской премии Гербертом Саймоном, который отметил, что «потребление информации забирает внимание потребителей, а изобилие информации ведёт к дефициту внимания». При этом каждый человек имеет право на получение контента, который ему действительно близок и интересен, без необходимости фильтровать тонны информационного шума вручную.

Согласно Digital News Report за 2025 год, лишь 46% людей активно следят за новостями. Молодежь (18–24 года) интересуется ими еще меньше – менее 30%, предпочитая развлекательный контент. В России около 80–90% граждан хотя бы изредка интересуются новостями, но лишь половина делает это постоянно. По данным ВЦИОМ, 87% россиян интересуются ситуацией в стране, однако регулярный интерес проявляют только 47%. Телевидение остается популярным источником (47%), за ним следуют новостные сайты (44%) и социальные сети с мессенджерами (31%). Люди старше 55 лет следят за новостями чаще (53%), чем молодежь (34%). Также наблюдается общее снижение интереса: 39% людей в мире осознанно избегают новостей из-за их негативного влияния, которое вызывает усталость и депрессию. Люди продолжают интересоваться новостными событиями, но специально избегают их из-за эмоциональных переживаний.

С экономической точки зрения, новостные рекомендательные системы представляют собой ключевой инструмент монетизации для медиа-платформ. Согласно отчёту McKinsey & Company (2024), внедрение персонализированных рекомендаций увеличивает следующие метрики:

- Click-Through Rate (CTR) – в 2–3 раза по сравнению с неперсонализированной лентой,
- Время на сайте (time-on-site) – на 40–60%,
- Показатель удержания пользователей (retention rate) - на 25–35% через30 дней после установки.

Доход от рекламы – пропорционально росту времени на сайте, так как рекламные показы увеличиваются. Инвестиции в разработку качественной рекомендательной системы окупаются за счёт роста вовлеченности аудитории и, как следствие, рекламной выручки.

Таким образом, актуальность разработки новостной рекомендательной системы обосновывается совокупностью факторов: социальный фактор: 39% людей избегают новостей из-за эмоционального выгорания, а качественная персонализация может снизить этот эффект, показывая только релевантный и не травмирующий контент. Экономический фактор: медиа-платформы получают рост CTR в 2–3 раза и увеличение времени на сайте на 40–60% при внедрении персонализации. Улучшение алгоритмов рекомендации и совершенствование подходов к их построению становится критически необходимым, а также адаптироваться к динамике информационных предпочтений в условиях перегрузки.

1.2 Обзор аналогов

Существует большое количество новостных сайтов: от крупных изданий до локальных региональных порталов. Но большинство из них предоставляют новости в формате «единой ленты» без персонализации, где порядок материалов определяется только редакционной политикой рассматриваемого сервиса. Такие сервисы не относятся к рекомендательным системам, так как

не учитывают индивидуальные предпочтения пользователя и не адаптируют ленту под его поведение.

В рамках данного обзора рассматриваются только те сервисы, которые позиционируются как рекомендательные. К таким сервисам относятся: Google News, «Новости LifeLines» и «ЯRUS».

1.2.1 Google News

Google News – это новостная рекомендательная система, популярная во всем мире. Благодаря широкой распространенности самого Google. Интерфейс сервиса представлен на рисунке 1.

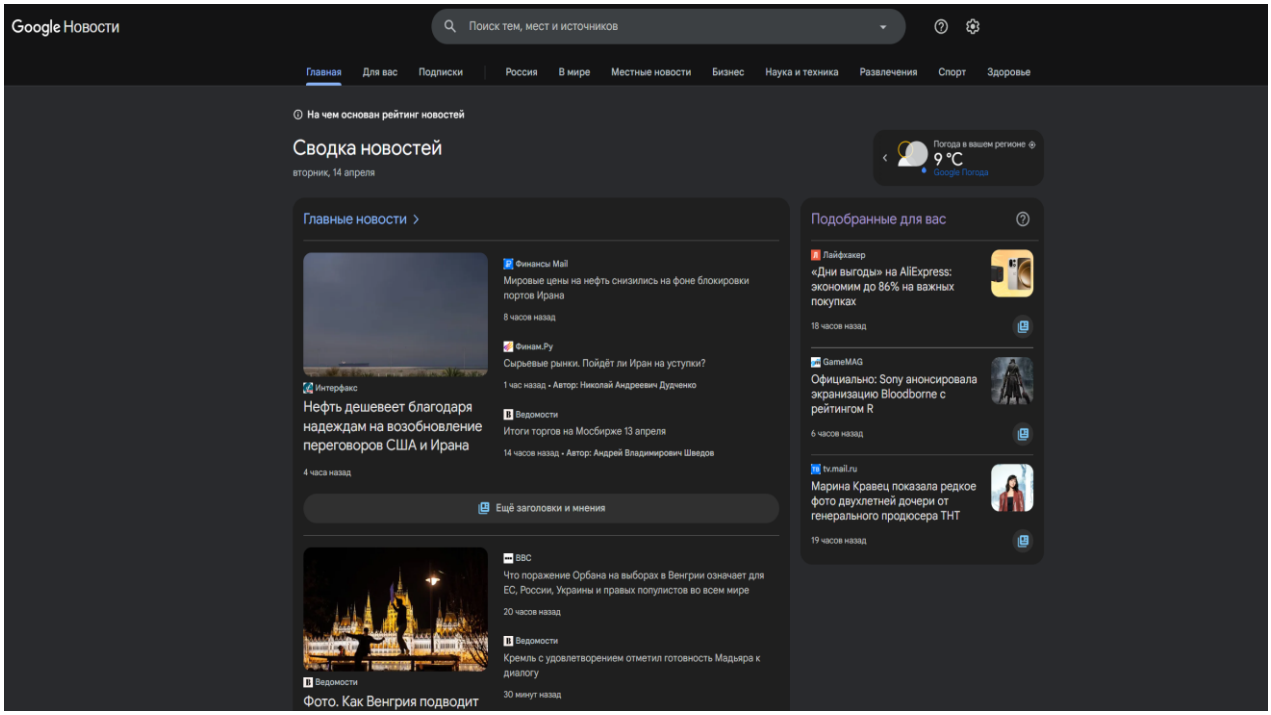


Рисунок 1 - Google News

Основная особенность данного сервиса заключается в том, что новостные рекомендации строятся на основе поисковых запросов пользователей.

Это увеличивает выборку метрик рекомендаций, но с другой – может стать проблемой, поскольку люди зачастую ищут не ту информацию, которая им действительно интересна, а ту, которая необходима для работы или учебы. Кроме того, многие пользователи относятся к такому подходу с недоверием, опасаясь, что их запросы – это конфиденциальная информация, и она передается стороннему сервису.

1.2.2 Новости LifeLines

Популярным приложением в магазине приложений RuStore является «Новости LifeLines». В приложении есть много удобных функций, например, ИИ-разбор, который позволяет проверить достоверность событий. Это удобно, но нужен ли сервис, чьи новости нужно после этого проверять на достоверность. Интерфейс данного приложения представлен на рисунке 2.

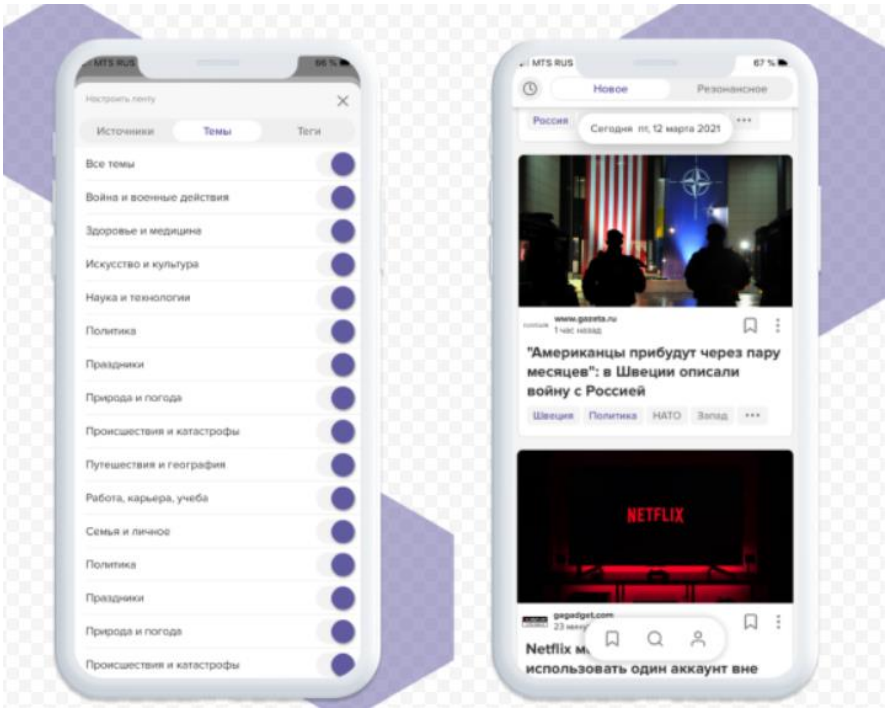


Рисунок 2 - Новости LifeLines

Также в приложении есть комментарии и настройка персональной ленты: выбор определенных тем, источников или географии. Это удобная фильтрация, но она не совсем относится к рекомендациям.

1.2.3 ЯRUS

Еще одним приложением является «ЯRUS», доступный в AppStore и Google Play. В описании он обозначен как «социальная лента».

Смысл «социальной ленты» в следующем: пользователь выбирает свои интересы и получает персонализированную ленту новостей. Кроме того, он может создавать свою ленту, добавлять ярусы и посты - специально для этого создан простой и удобный блочный редактор. Интерфейс данного приложения представлен на рисунке 3.

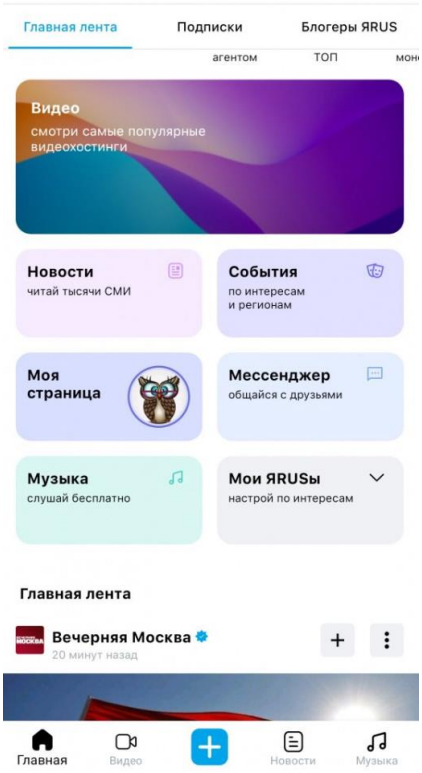


Рисунок 3 - ЯRUS

Пользователя встречает главный экран приложения с двумя вкладками: лента новостей и вкладка, на которой пользователь может создавать подборки, вкладки и ярусы. Например, при создании яруса «Политики» и нажатии на иконку пользователь видит все новости, где политика упоминается. При создании яруса можно выбрать, откуда туда будут поступать данные: из пользовательских публикаций, новостей, видео или раздела событий.

1.2.4 Сравнение рассмотренных аналогов

Рассмотренные сервисы: Google News, «Новости LifeLines» и «ЯRUS» Данные сервисы предлагают разные подходы к рекомендации новостного контента, и у каждого есть свои сильные и слабые стороны.

Google News строит рекомендации на основе поисковых запросов пользователей. Это дает широкий охват, но порождает две проблемы. Во-первых, люди часто ищут информацию не для души, а для работы или учебы, поэтому рекомендации могут не попадать в их реальные интересы. Во-вторых, пользователи с недоверием относятся к тому, что их поисковые запросы, которые относятся к личной информации, передаются сторонним сервисам.

Google News это мощный, но спорный с точки зрения приватности инструмент.

«Новости LifeLines» делают ставку на ИИ-разбор для проверки достоверности событий. Кроме того, настройка персональной ленты здесь сводится к выбору тем, источников и географии – это удобная фильтрация, но не полноценная рекомендательная система. LifeLines скорее про верификацию, чем про персонализацию.

«ЯRUS» позиционирует себя как «социальная лента». Пользователь сам выбирает интересы и получает соответствующую ленту. Главное отличие данного сервиса это возможность создавать собственные ярусы то есть тематические подборки и выбирать источники наполнения: публикации пользователей, новости, видео или события. Это дает больше контроля, но рекомендации здесь тоже не являются адаптивными: система не учится на поведении пользователя, а просто фильтрует контент по заданным вручную правилам.

Ни одна из рассмотренных систем не анализирует эмоциональную окраску новостных текстов. Не определяется, вызывает ли новость тревогу, страх, радость или гнев. Не учитывается потенциальное воздействие на психологическое состояние пользователя. В то же время подход, реализованный в «ЯRUS» то есть, создание тематических подборок (ярусов) на основе ключевых слов, демонстрирует потенциал использования тегов как дополнительной метрики для новостных рекомендаций. Если в «ЯRUS» теги задаются пользователем вручную, то в автоматизированной рекомендательной системе они могут формироваться на основе анализа текста новости.

1.3 Обзор и сравнение существующих типов рекомендательных систем

Рекомендательная система - это система которая на основе поведения пользователя и метаданных контента формирует персонализированную ленту. На вход рекомендательной системы поступает контент с метаданными, а

также события взаимодействия пользователя с контентом (просмотры, лайки, время чтения).

Для реализации новостной рекомендательной системы необходимо сравнить основные типы рекомендательных систем. Существует три основных типа рекомендательных систем:

Фильтрация на основе популярности. Наиболее простая система выдает рекомендации на основе популярности. Чем выше средний рейтинг фильма, купленного товара или статьи, тем вероятнее, что система будет рекомендовать именно их. Основное преимущество – простота реализации и устойчивость к холодному старту. Недостаток - отсутствие персонализации.

Фильтрация на основании контента. Подход базируется на сравнении предпочтений пользователей с продукцией или контентом, который их ранее заинтересовал. Например, в случае с фильмами, если человек просмотрел фильм конкретного жанра, система предложит ему фильмы того же репертуара. Преимущество - отсутствие проблемы холодного старта для новых элементов. Недостаток - ограниченная серендипность (не может рекомендовать неожиданный, но интересный контент).

Коллаборативная фильтрация. Метод оперирует анализом данных о пользователях с максимально схожими вкусами и интересами, чтобы составить персонализированные рекомендации. Этот подход учитывает не только оценки отдельного пользователя, но также берёт во внимание предпочтения других покупателей, включая как явные, так и неявные сигналы. Преимущество данной фильтрации: высокое качество персонализации и способность находить неочевидные взаимосвязи. Недостаток - чувствительность к разреженности и холодному старту новых пользователей.

У каждой рекомендательной системы есть свои плюсы недостатки. В основном выделяют четыре общих проблемы: холодный запуск, разреженность, конфиденциальность, персонализация.

Проблема холодного запуска заключена в том, что изначально нет никаких данных ни о пользователя, ни о товарах. Рекомендательная система

просто не знает на основе каких данных делать рекомендацию. Такая же проблема возникает в случае нового пользователя, в системе отсутствует информация.

Проблема разреженности появляется, когда пользователи не дают обратной связи, например, не оценивают рекомендованный продукт.

Проблема конфиденциальности возникает, когда системе чтобы создавать рекомендации, необходимо знать личные данные пользователя, например, место работы, проживания. Такие данные необходимо хранить в зашифрованном виде.

Персонализации в рекомендательных системах заключается в создании рекомендаций, которые максимально соответствуют индивидуальным предпочтениям пользователей, при этом сохраняя баланс между релевантностью, разнообразием [1].

1.4 Проблема выбора информативных признаков в новостных рекомендательных системах

Прежде чем разрабатывать новостную рекомендательную систему, необходимо определиться с набором метрик (количественных показателей, позволяющих оценить качество рекомендации), позволяющих понять, понравилась ли пользователю та или иная новость. При построении рекомендательных систем для фильмов часто используется прямой опрос пользователя с целью получения оценки, на основе которой формируется рейтинг. Однако такой подход неприменим для новостных систем, поскольку пользовательский опрос не даёт чёткого понимания, понравилась ли пользователю сама новость или способ её рекомендации.

В связи с этим в качестве базовых метрик приходится использовать поведенческие показатели: клики и время просмотра. С этими данными необходимо корректно работать, так как пользователь может случайно кликнуть по новости или перейти по ссылке и отвлечься. Следовательно, время просмотра следует оценивать с учётом объёма (длины) новостной статьи.

Кроме того, в современных новостных порталах широко используются теги (ключевые слова или метки, присваиваемые новости для её описания и категоризации). Часто теги в новостях отражают не тематику новости, а ключевые слова или тегом является какое определенное название места или имя человека. Если рекомендательный алгоритм будет получать новости из разных источников, каждый из которых применяет собственную систему тегирования, то исходные теги, окажутся непригодными.

Теги могут обозначать разные сущности (например, место, имя человека, праздник и так далее), необходимо предварительно переразмечать новости тегами, которую отражают одинаковую сущность, чтобы рекомендательный алгоритм работал стабильнее.

1.5 Постановка задачи

Цель - исследовать методы которые позволят повысить качество рекомендаций новостного контента.

Отдельным направлением исследования является изучение возможности определения эмоций в новостных текстах. Необходимо проанализировать существующие подходы и алгоритмы классификации эмоций, а также провести экспериментальное исследование. По результатам эксперимента требуется сделать обоснованный вывод о целесообразности включения эмоционального анализа в новостную рекомендательную систему.

Необходимо проанализировать влияния тегов на качество рекомендаций новостного контента. Для этого необходимо реализовать алгоритм классификации, который обеспечивает присвоение тегов новостным материалам на основе их текстового содержания.

Необходимо проанализировать эффективность использования алгоритма основанном на модели BERT, которая необходима для семантического анализа текста. Такой подход предположительно может повысить качество рекомендаций, в условиях, ограниченных метаданных.

Следующей задачей является проведение сравнительного анализа рекомендательных алгоритмов, представляющих основные типы систем. В

рамках эксперимента необходимо реализовать и сравнить неперсонализированные алгоритмы (на основе популярности), коллаборативные алгоритмы, а также алгоритмы на основе контента и нейросетевые алгоритмы включая BERT. Так же необходимо произвести сравнение чтобы определить наиболее эффективный алгоритм для новостной рекомендательной системы с учётом специфики новостного контента, а также изучить как меняется качество рекомендаций на размеченных тегами данных и нет.

В результате выполнения работы должны быть получены следующие результаты: обоснованный вывод о целесообразности использования эмоционального анализа в новостных рекомендательных системах; экспериментально выбранный оптимальный алгоритм автоматической разметки новостей тегами; результаты анализа влияния теговой разметки на качество рекомендаций; результаты сравнительного анализа рекомендательных алгоритмов (неперсонализированных, коллаборативных, контентных и нейросетевых, включая подход на основе BERT) с обоснованием выбора наиболее эффективного для новостного контента, в том числе на размеченных и неразмеченных тегами данных

2 Разработка модуля извлечения тегов

Теги - это ключевые слова или короткие фразы (например, «ключевая ставка», «искусственный интеллект»), описывающие тему новости, в отличие от широких категорий. Теги в контексте новостной рекомендательной системы выполняют несколько критически важных функций. Теги в контексте новостной рекомендательной системы выполняют несколько критически важных функций. Они могут послужить основной метрикой для сопоставления контента с пользовательскими предпочтениями, позволит группировать схожие материалы, обеспечит возможность тематической фильтрации. Качество и точность тегов напрямую определяют эффективность всей рекомендательной системы.

На вход новостной рекомендательной системы не всегда поступает контент с корректно размеченными тегами и категориями. На практике значительная часть новостных материалов, особенно получаемых из внешних источников, RSS-лент или агрегаторов, не содержит структурированных метаданных, необходимых для эффективной работы классических рекомендательных алгоритмов. Без точной тематической разметки система не может корректно сопоставить контент с пользовательскими интересами, что приводит к снижению качества рекомендаций и ухудшению пользовательского опыта.

В связи с этим возникает необходимость разработки алгоритма, способного автоматически определять и назначать теги для новостных материалов на основе их текстового содержания. Такой алгоритм должен решать задачу классификации, анализируя семантику новости, выявляя ключевые темы и концепции, и сопоставляя их с предопределенным набором тематических категорий.

2.1 Разработка парсера новостей

Сбор данных - это ключевой этап проекта. В качестве источника выбран сайт РИА Новости. Его преимуществами являются простая структура страниц

и наличие разметки новостей по тегам, что значительно упрощает автоматическое извлечение информации.

Собранные данные планируется хранить в файле формата JSON. Этот выбор обусловлен широкой поддержкой формата в различных языках программирования и его удобством для структурированного хранения данных. Структура файла представлена ниже:


```
{"metadata": {
    "source": "ria.ru",
    "created_at": "2026-03-28T18:49:50.053992",
    "total_tags": 11,
    "total_articles": 326,
    "total_articles_with_content": 326
  },
  "tags_data": [
    {
      "tag_name": "В мире",
      "tag_url": "https://ria.ru/world/",
      "articles_count": 40,
      "target_articles": 60,
      "articles": [
        {
          "title": "",
          "url": "",
          "image_url": "",
          "tags": [],
          "date": "21:30",
          "views": "1034"
        }
      ]
    }
  ]
}
```


Разработанный парсер собирает данные, которые можно разделить на несколько этапов. На первом происходит инициализация и настройка сессии

HTTP с использованием библиотеки requests. Для имитации поведения реального пользователя задаются заголовки User-Agent, Accept-Language и другие параметры, что позволяет избежать блокировок со стороны сервера. Также на этом этапе настраивается система логирования для отслеживания всех действий программы и возможных ошибок.

Второй этап заключается в получении списка всех доступных тегов (рубрик) с главной страницы новостного портала. Парсер загружает HTML-код страницы и с помощью библиотеки BeautifulSoup находит блок с идентификатором headerTagBand, содержащий все рубрики. Из этого блока извлекаются названия тегов и соответствующие им URL-адреса, которые в дальнейшем будут использоваться для сбора новостей. Ниже представлен фрагмент кода, реализующий данный этап:

```
python
def get_tags_from_page(self, url):
    response = self.session.get(url, timeout=10)
    soup = BeautifulSoup(response.content,
'html.parser')
    tag_band = soup.find('div', id='headerTagBand')
    tag_links = tag_band.find_all('a',
class_='header__tag-band-item-link')
    tags = []
    for tag_link in tag_links:
        tag_name = tag_link.get_text(strip=True)
        tag_href = tag_link.get('href', '')
        full_url = urljoin('https://ria.ru', tag_href)
        tags.append({'name': tag_name, 'url':
full_url})
    return tags
```

Третий этап является основным и включает в себя сбор статей для каждого найденного тега с возможностью динамической подгрузки контента, так как страницы загружаются по страницам по кнопку «ещё» Парсер загружает страницу тега и извлекает все элементы с классом list-item, каждый

из которых соответствует одной новости в ленте. Для каждой статьи извлекаются метаданные: заголовок, URL, ссылка на изображение, список тегов, дата публикации и количество просмотров. Особенность реализации заключается в обработке механизма бесконечной прокрутки, когда парсер находит элемент с классом list-more и извлекает из него URL для AJAX-запроса, после чего рекурсивно подгружает следующие порции новостей.

Четвертый этап алгоритма отвечает за загрузку полного текста каждой собранной статьи. Для этого парсер переходит по сохраненному URL новости и анализирует структуру страницы, выделяя блок с классом article_body и все вложенные элементы article_text. Полученные фрагменты текста объединяются в единую строку и очищаются от лишних пробелов и переносов с помощью регулярных выражений. Для снижения нагрузки на сервер, во избежание блокировки сервером клиента, добавлена задержка в 1 секунду. Реализация извлечения контента представлена в следующем фрагменте кода:

```
python
def extract_article_content(self, article_url):
    response = self.session.get(article_url,
timeout=10)
    soup = BeautifulSoup(response.content,
'html.parser')
    content_parts = []
    article_body = soup.find('div',
class_='article__body')
    if article_body:
        text_blocks = article_body.find_all('div',
class_='article__text')
        for block in text_blocks:
            text = block.get_text(strip=True)
            if text:
                content_parts.append(text)
    full_content = '\n\n'.join(content_parts)
    full_content = re.sub(r'\n\s*\n', '\n\n',
full_content)
```

```
        return full_content.strip()
```

Заключительный этап работы парсера включает сбор всех данных и их сохранение в JSON-файл в соответствии с заранее определенной структурой. Для каждого тега формируется отдельный блок, содержащий название рубрики, URL, количество собранных статей и массив объектов с полной информацией о новостях. В метаданные файла записываются общая статистика: источник данных, время создания, общее количество тегов и статей. Такая структура обеспечивает удобный доступ к данным для последующего анализа.

2.2 Предобработка тестовых данных

Важнейшим этапом перед обучением моделей предобработка текстов новостей. Из каждого текста удаляются все символы, не являющимися буквами русского алфавита, выполнилось приведение к нижнему регистру и удаление стоп-слов, то есть наиболее частотных слов, не несущих смысловой нагрузки [2]. Дополнительно из текста удаляются слишком короткие слова (длиной менее трёх символов) и выполнялась обрезка до 400 слов. После очистки текстов все статьи преобразованы в числовые векторы с использованием метода TF-IDF (Term Frequency-Inverse Document Frequency). Этот метод оценивает важность каждого слова в документе относительно всей коллекции: чем чаще слово встречается в конкретном тексте и чем реже во всех остальных, тем выше его вес.

2.3 Сравнение алгоритмов определения тегов

Для реализации задачи классификации были выбраны шесть различных алгоритмов машинного обучения, представляющих как классические методы, так и современные градиентные бустинги. В качестве базовых алгоритмов выступили:

– наивный байесовский классификатор (Naive Bayes),

– дерево решений (Decision Tree),

– метод k-ближайших соседей (KNN),

– логистическая регрессия (Logistic Regression),

– случайный лес (Random Forest),

– градиентный бустинг LightGBM.

Каждый из шести алгоритмов был реализован в виде отдельного класса с единым интерфейсом, содержащим методы fit для обучения и predict для предсказания. Такая архитектура позволила унифицировать процесс сравнения и добавления новых алгоритмов без изменения основного кода. Наивный байесовский классификатор использовал TF-IDF векторизацию с 5000 признаков и биграммами, а параметр сглаживания alpha=0.1 был подобран для предотвращения нулевых вероятностей. Логистическая регрессия применяла стратегию "один против всех" (one-vs-rest), что позволило эффективно решать многоклассовую задачу путем обучения отдельного бинарного классификатора для каждого тега.

Алгоритм LightGBM использовал 150 деревьев с максимальной глубиной 8 и скоростью обучения 0,05, а также балансировку весов классов для учета неравномерного распределения новостей по рубрикам. Ниже приведена реализация класса для случайного леса:

```
python
class SimpleRF:
    def __init__(self):
        self.vec = TfidfVectorizer(max_features=5000,
ngram_range=(1, 2))
        self.clf =
RandomForestClassifier(n_estimators=100, n_jobs=-1,
random_state=42)

    def fit(self, X, y):
        X_vec = self.vec.fit_transform(X)
        self.clf.fit(X_vec, y)

    def predict(self, X):
        X_vec = self.vec.transform(X)
        return self.clf.predict(X_vec)
```

Основной метрикой выбрана микро-усредненная F1-мера (micro F1), которая вычисляет общее количество истинно положительных, ложноположительных и ложноотрицательных ответов по всем классам и затем рассчитывает F1-меру. Дополнительно применяется макро-усредненная F1-мера (macro F1), вычисляющая среднее арифметическое F1-мер по каждому классу, и взвешенная F1-мера (weighted F1), учитывающая количество объектов в каждом классе. Такие метрики оценивают, как общее качество модели, так и её способность корректно классифицировать редкие рубрики[3].

Помимо метрик качества, измеряются и временные характеристики работы алгоритмов: время обучения модели и время предсказания в миллисекундах на одну статью. Для оценки вся выборка из 326 новостей разделена на обучающую (80%) и тестовую (20%) части со стратификацией – это означает, что пропорции каждого тега сохраняются в обеих выборках. Такое разделение гарантирует, что редкие категории не исчезнут полностью из обучающей или тестовой выборки, что важно при работе с несбалансированными данными.

После обучения всех шести алгоритмов проведен сравнительный анализ их производительности по качеству и скорости работы. Результаты показали, что наивный байесовский классификатор не уступают по качеству более сложным методам, и выигрывает в скорости обучения и предсказания. Логистическая регрессия и случайный лес показали существенно более высокие результаты, достигая микро F1-меры на уровне 0,75-0,78, что подтверждает эффективность линейных и ансамблевых методов для текстовой классификации.

Лучший результат продемонстрировал наивный баейсовский классификатор, достигнув микро F1-меры 0,82, что превышает целевой порог 0,80 и соответствует высокому качеству классификации. Временные характеристики также оказались приемлемыми: время составило около 15 миллисекунд на одну статью, что позволяет обрабатывать до 60 новостей в секунду. Для визуализации результатов построены диаграммы, сравнивающие

метрики качества, скорость инференса и время обучения всех алгоритмов, что позволяет наглядно оценить преимущества каждой модели.

2.4 Выбор оптимального метода

Проведённый анализ показал различия в качестве работы шести алгоритмов. Результаты показаны в таблице 1.

Наивный байесовский классификатор продемонстрировал лучшие результаты по всем метрикам: точность и микро F1-мера достигли 0,832, а макро F1-мера составила 0,818, что свидетельствует о сбалансированной работе как с частотными, так и с редкими категориями новостей.

Дерево решений строит модель в виде иерархической структуры вопросов о значениях признаков. Низкие результаты можно интерпретировать большим количеством уникальных слов.

Случайный лес представляет собой ансамбль множества деревьев решений, каждое из которых обучается на случайной выборке данных и признаков. Такой подход снижает риск переобучения и повышает устойчивость модели. В данном исследовании он показал хорошие, но не лучшие результаты микро F1 равен 0,725.

Метод K ближайших соседей (KNN) относит объект к категории, наиболее часто встречающейся среди K ближайших к нему объектов обучающей выборки. Алгоритм не требует предварительного обучения, но чувствителен к размерности пространства признаков. На текстовых данных его показатели уступают логистической регрессии и наивному байесовскому классификатору, дав результат микро F1 равному 0,687

Логистическая регрессия оценивает вероятность принадлежности объекта к классу с помощью логистической функции. Она хорошо работает с линейно разделимыми данными и может быть применима для бинарной классификации. В задаче много классовой классификации новостей она продемонстрировала результаты, сопоставимые с результатами случайного лесу равен 0,75.

LightGBM это продвинутый ансамблевый метод, который строит деревья решений последовательно, каждое следующее исправляет ошибки предыдущих. Результат оказался невысоким микро F1 равен 0,588. Вероятная причина – это недостаточный объём обучающей выборки для эффективного обучения.

На основе полученных результатов наивный байесовский классификатор определен как лучший алгоритм. Этот выбор обусловлен тремя факторами: наилучшим качеством классификации среди всех алгоритмов; минимальным временем обработки текста (0,6 мс на статью), что позволяет обрабатывать более 1600 новостей в секунду; крайне низкие требованиями к вычислительным ресурсам.

Таким образом, для системы автоматического подбора тегов новостей было решено использовать наивный байесовский классификатор как наиболее эффективный [4, 5]. Данные тестирования представлены в таблице 1 и рисунке 4.

Таблица 1 – Сравнительный анализ моделей

Алгоритм	Micro F1	Macro F1
Naive Bayes	0,832	0,818
Logistic Regression	0,756	0,702
Random Forest	0,725	0,697
KNN	0,687	0,661
LightGBM	0,588	0,571
Decision Tree	0,466	0,443

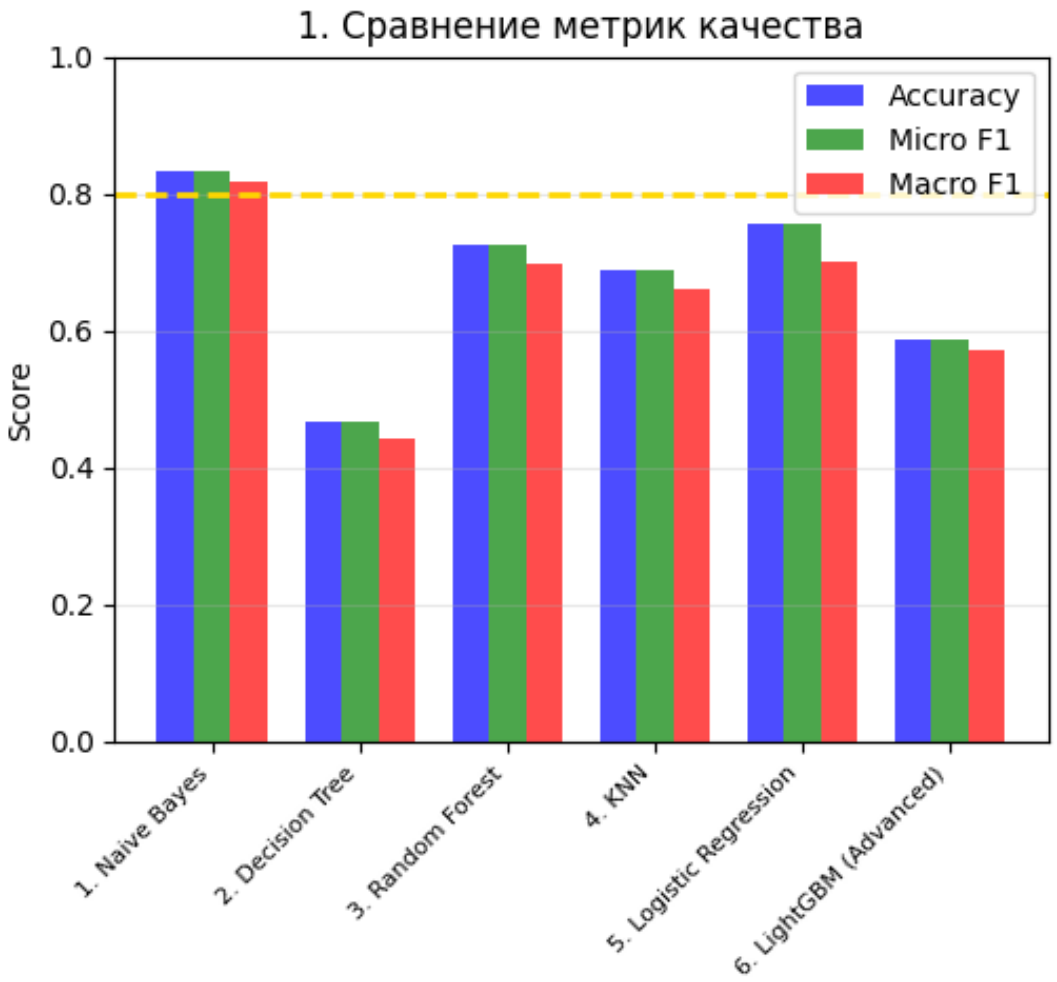


Рисунок 4 – сравнение метрик

2.5 Выводы по результатам автоматического тегирования новостей

В рамках второй главы была решена задача разработки модуля автоматического извлечения тегов и подготовки данных для новостной рекомендательной системы. Разработан функциональный парсер новостей с использованием библиотек requests и BeautifulSoup для сбора новостных материалов с сайта РИА Новости. Парсер поддерживает обход всех доступных рубрик (11 тегов), обработку динамической подгрузки контента (AJAX-запросы по кнопке «ещё») и извлечение полных текстов статей. В результате работы парсера собран датасет из 1200 новостей с метаданными (заголовок, URL, дата, количество просмотров) и текстовым содержанием, которые сохранены в структурированном формате JSON, удобном для последующего анализа и обучения моделей.

Для адаптации текстов к специфике русского языка разработана функция очистки текста, она реализует приведение текста к нижнему регистру, удаление неалфавитных символов, фильтрацию стоп-слов и коротких слов (менее 3 символов), а также обрезку до 400 наиболее значимых слов. Это позволяет снизить размерность признакового пространства и ускорить обучение без потери качества. После очистки текстов все статьи были преобразованы в числовые векторы с использованием метода TF-IDF, который оценивает важность каждого слова в документе относительно всей коллекции.

Проведено сравнение шести алгоритмов классификации: наивный байесовский классификатор, логистическая регрессия, случайный лес, KNN, LightGBM и дерево решений. Оценка производилась по метрикам Accuracy, Micro F1, Macro F1, а также по времени обучения и скорость обработки текста. Для объективного сравнения вся выборка из 1200 новостей разделена на обучающую (80%) и тестовую (20%) части со стратификацией, что гарантирует сохранение пропорций каждого тега в обеих выборках[6, 7].

Лучшие результаты показал наивный байесовский классификатор, достигнув микро F1 = 0,832, макро F1 = 0,818 и Accuracy = 0,832. Логистическая регрессия и случайный лес показали приемлемые, но более низкие результаты – микро F1 около 0,75 и 0,72 соответственно. Наихудшую производительность продемонстрировали дерево решений (микро F1 = 0,466) и градиентный бустинг LightGBM (микро F1 = 0,588). LightGBM и дерево решений показали низкие результаты. Градиентные бустинги требуют значительно большего объёма данных для достижения высокого качества, в то время как наивный байесовский классификатор устойчиво работает даже на небольших выборках благодаря своей вероятностной природе и предположению о независимости признаков [5].

На основе полученных результатов для дальнейшего использования выбран наивный байесовский классификатор. Таким образом, поставленная задача разработки модуля автоматического извлечения тегов успешно решена.

Выбранный наивный байесовский классификатор обеспечивает высокое качество тегирования при минимальных вычислительных затратах, что делает его пригодным для использования в составе итоговой новостной рекомендательной системы.

3 Способы определения эмоций в тексте

В современном мире, многие люди целенаправленно избегают новостного контента, во избежание негативных эмоции. Зная то, как текст новости может повлиять на эмоциональное состояние человека, то есть какую именно эмоцию вызовет данная новость у читателя, можно скорректировать рекомендательный алгоритм таким образом, чтобы из списка предложенных новостей убирать негативно влияющие новости. Эмоциональная окраска может стать одной из метрик при рекомендации новостного контента. Из-за этого возникает необходимость определения эмоций в тексте, с целью улучшений качества рекомендаций.

В данном исследовании определяем вызываемые эмоции у человека. Для рекомендации необходимо учитывать эмоции вызываемы у конкретного читателя.

В теории система могла бы помечать текст как негативно влияющий и не рекомендовать такие новостные статьи. Но основная проблема в том, что новость зачастую пишется без особых ключевых слов, которые помогли бы определить эмоции в тексте, новостные статьи пишутся нейтрально. И при определении эмоции необходимо понимать не то, какие эмоции заложены в новости, а то, как эта новость воспринимается читателем. Одна и та же новость может по-разному влиять на человека. Возникает проблема, нужна ли рекомендация человеку, если новость читателю нужна, но из-за эмоций он не хотел бы её читать. Например, если человек прочитает новость о проигрыше своей любимой футбольной команды. В данном исследовании проверяется сама возможность выявления эмоций в тексте.

Также нужно понимать, что на людей негативно влияют те новостные события, которые касаются их лично. Например, новость о своей стране вызовет больше негативных или положительных эмоций, чем аналогичная новость из другой страны. При разработке рекомендательной системы это можно учесть, и не является проблемой, так как можно собирать с пользователя не только данные об эмоциях в текстах, но ещё и определять

геолокацию человека. Основной проблемой является определения отношения пользователя к той или иной новости.

3.1 Описание и подготовка тестовых данных

Основная проблема тестовых данных для определения новостных событий, заключается в том, что нельзя взять набор новостей и самому разметить данные. Мы должны учитывать то как это эмоция воспринимается читателем, одна и та же новость, может вызывать разные эмоции у разных пользователей. Соответственно создавать датасет или искать нужно исходя из этих требований.

В качестве опорного датасета был выбран набор данных, представленный в работе Dataset of emotional reactions to news articles. Данный датасет содержит новостные статьи, собранные из различных мексиканских источников, а также зафиксированные эмоциональные реакции пользователей на эти материалы. В рамках эксперимента участникам предлагалось ознакомиться с текстами новостей, после чего они отмечали эмоции, которые вызвала у них прочитанная информация. В качестве категорий использовались такие эмоциональные состояния, как радость, удивление, вдохновение, тронутость, нейтральность, грусть, страх и гнев.

Авторы изначально разработали и собрали этот набор данных именно для использования в задачах построения новостных рекомендательных систем, учитывающих эмоциональный отклик пользователей. Таким образом, разметка данных формируется не экспертным способом, а на основе реального пользовательского восприятия, что повышает её прикладную ценность[8].

Особенностью рассматриваемого набора данных является его ориентация на читательскую интерпретацию новостного контента, что делает его особенно полезным для задач анализа тональности и эмоциональной окраски текстов. Прозрачная методология сбора данных и наличие четко определённых классов эмоций позволяют использовать данный датасет для обучения и тестирования моделей, ориентированных на выявление

эмоциональной реакции аудитории, а также для разработки рекомендательных систем в сфере новостных сервисов.

Исходный датасет характеризуется наличием множественных пользовательских реакций на одну и ту же новость, причем эти реакции часто были противоречат друг другу. Например, одна новость могла одновременно получить метки «радость», «гнев» и «удивление» от разных читателей. Такая структура данных делала невозможным прямое обучение моделей классификации, поскольку алгоритм не может однозначно определить, какую эмоцию следует предсказывать для конкретного текста.

Целью подготовки данных стало преобразование не отформатированных пользовательских оценок в структурированный формат, где каждой новости соответствует единственная эмоциональная метка, полученная на основе агрегирования мнений множества респондентов. Процесс включал три основных этапа.

На первом этапе проводилась очистка текстов новостей: удалялись URL-ссылки, специальные символы, лишние пробелы, а все буквы приводились к нижнему регистру. Это необходимо для унификации текстового представления и устранения шумовых признаков [9].

На втором этапе выполнялась агрегация данных: все пользовательские ответы группировались по каждой уникальной новости, после чего методом простого большинства определялась основная эмоция. Дополнительно рассчитывался процент согласованности: доля пользователей, выбравших основную эмоцию. Это позволило количественно оценить степень субъективности восприятия каждой новости.

На третьем этапе формировался итоговый обучающий датасет, содержащий только список новостей с форматированными текстами и соответствующим этим новостям основной эмоции. Такой подход обеспечил создание непротиворечивой размеченной выборки, необходимого объёма, пригодной для последующего обучения и сравнения моделей машинного обучения.

3.2 Обзор алгоритмов классификации

Для того чтобы определить, какие эмоции вызывает новость у читателей, были выбраны несколько алгоритмов машинного обучения. Среди них есть как простые и быстрые методы (наивный байесовский классификатор, логистическая регрессия), так и более сложные ансамблевые алгоритмы (случайный лес, LightGBM, XGBoost, CatBoost). Такой набор позволяет сравнить, какой подход лучше справляется с задачей распознавания эмоций в новостных текстах.

3.1.1 Наивный Байесовский классификатор

Наивный Байесовский классификатор представляет собой вероятностный метод, основанный на применении теоремы Байеса с «наивным» предположением о независимости признаков друг от друга. В контексте текстовой классификации это означает, что вероятность принадлежности документа к определенному классу вычисляется как произведение вероятностей каждого слова в документе для этого класса. Несмотря на кажущуюся упрощенность, данное предположение на практике часто дает хорошие результаты, особенно при работе с разреженными данными, характерными для текстовых представлений.

Основными преимуществами наивного байесовского классификатора являются высокая скорость обучения и предсказания, а также устойчивость к шуму и нерелевантным признакам. Алгоритм требует небольшого объема обучающих данных для достижения приемлемого качества и хорошо масштабируется на большие наборы данных. В контексте данной работы наивный байесовский классификатор выступает в роли базового алгоритма, позволяющего оценить минимальный достижимый уровень качества и сравнить с ним более сложные методы.

3.1.2 Логистическая регрессия

Логистическая регрессия представляет собой линейный метод классификации, который для многоклассовых задач использует стратегию «один против всех» или softmax-нормализацию. В случае с задачей

определения эмоций новостей алгоритм обучает отдельный бинарный классификатор для каждой эмоции, каждый из которых предсказывает вероятность того, что текст относится к данному классу. Итоговое решение принимается в пользу класса с максимальной предсказанной вероятностью.

Данный метод обладает рядом важных преимуществ: высокая интерпретируемость результатов (можно определить, какие слова вносят наибольший вклад в предсказание каждой эмоции), устойчивость к переобучению при правильной регуляризации и высокая скорость работы как на этапе обучения, так и на этапе предсказания. Логистическая регрессия особенно эффективна в случаях, когда классы в данных линейно разделимы, и часто служит надежным бенчмарком при сравнении с более сложными ансамблевыми методами и нейросетевыми архитектурами.

3.1.3 Случайный лес

Случайный лес представляет собой ансамблевый метод машинного обучения, объединяющий множество решающих деревьев, каждое из которых обучено на случайной подвыборке данных и случайном подмножестве признаков. Итоговое решение принимается путем голосования всех деревьев ансамбля, что позволяет существенно снизить дисперсию и предотвратить переобучение по сравнению с одиночным решающим деревом. В контексте классификации эмоций каждое дерево в лесу «видит» случайный набор слов из текста, что делает ансамбль устойчивым к шумовым признакам.

Основными преимуществами данного алгоритма являются способность улавливать нелинейные зависимости между признаками, устойчивость к выбросам и шуму в данных, а также встроенная оценка важности признаков, позволяющая определить, какие слова наиболее информативны для предсказания. Алгоритм можно распараллелить и эффективно использовать для работы большими данными, однако его основным недостатком является меньшая интерпретируемость по сравнению с линейными моделями и склонность к переобучению при недостаточном количестве обучающих примеров для редких классов эмоций.

3.1.4 LightGBM

LightGBM представляет собой современную реализацию градиентного бустинга, разработанную компанией Microsoft, которая отличается высокой скоростью обучения и эффективным использованием памяти. Ключевой особенностью алгоритма является использование техники односторонней выборки на основе градиента и связки эксклюзивных признаков, что позволяет обрабатывать большие объемы данных значительно быстрее классического XGBoost. LightGBM строит деревья «по листьям», в отличие от традиционного «по уровням», что приводит к построению более глубоких и сложных деревьев.

LightGBM демонстрирует хорошую способность к моделированию сложных взаимодействий между словами и фразами. Алгоритм поддерживает работу с категориальными признаками без предварительного кодирования, имеет встроенную обработку пропущенных значений и позволяет использовать взвешивание классов для борьбы с дисбалансом. Основным ограничением является чувствительность к гиперпараметрам и склонность к переобучению на малых выборках, что требует тщательной настройки параметров регуляризации.

3.1.5 XGBoost

XGBoost является одной из популярных реализаций градиентного бустинга. Алгоритм включает множество регуляризирующих механизмов, таких как L1- и L2-регуляризация, контроль глубины деревьев и коэффициенты сжатия, что эффективно предотвращает переобучение[10].

XGBoost предлагает такие преимущества, как встроенная обработка дисбаланса классов через параметр scale_pos_weight, возможность работы с разреженными матрицами (что критически важно для TF-IDF представлений) и эффективную параллелизацию вычислений. Алгоритм предоставляет детальную информацию о важности признаков, позволяя интерпретировать, какие лексические единицы наиболее значимы. Данный алгоритм требует

более тщательной настройки гиперпараметров по сравнению со случайным лесом и может быть медленнее LightGBM при обучении на больших данных.

3.1.6 CatBoost

CatBoost - это градиентный бустинг, разработанный компанией Яндекс, отличается механизмами работы с категориальными признаками и оригинальным подходом к борьбе со смещением предсказаний. Ключевая особенность CatBoost заключается в использовании упорядоченного бустинга и симметричных деревьев решений, что делает алгоритм менее подверженным переобучению по сравнению с XGBoost и LightGBM. Алгоритм также автоматически обрабатывает категориальные признаки без необходимости в их предварительном кодировании [10, 11].

Алгоритм предоставляет встроенный механизм автоматической настройки весов классов, что особенно важно для задач с дисбалансом эмоций. CatBoost отличается простотой использования: он хорошо работает с параметрами по умолчанию и требует минимальные настройки. Основным недостатком является большее время обучения по сравнению с LightGBM и XGBoost при работе с очень большими наборами данных.

3.2 Тестирование

Логистическая регрессия представляет собой линейный метод классификации, который оценивает вероятность принадлежности объекта к тому или иному классу. В данной работе алгоритм использовался с максимальным количеством итераций 1000 для обеспечения сходимости.

Для обучения модели использовался класс LogisticRegression из библиотеки scikit-learn. Были протестированы две версии: стандартная и с балансировкой весов классов (параметр class_weight='balanced').

```
python
from sklearn.linear_model import LogisticRegression

# Стандартная логистическая регрессия
model = LogisticRegression(max_iter=1000,
random_state=42)
```

```
model.fit(X_train_vec, y_train)
y_pred = model.predict(X_test_vec)

# Логистическая регрессия с балансировкой весов
model_balanced = LogisticRegression(
    max_iter=1000,
    class_weight='balanced',
    random_state=42
)
model_balanced.fit(X_train_vec, y_train)
y_pred_balanced = model_balanced.predict(X_test_vec)
```

Случайный лес представляет собой ансамблевый метод, строящий множество деревьев решений и усредняющий их результаты. Каждое дерево обучается на случайной подвыборке данных и случайном подмножестве признаков, что позволяет снизить переобучение.

Использовался класс RandomForestClassifier со 100 деревьями в ансамбле. Как и в случае с логистической регрессией, тестировались стандартная версия и версия с балансировкой весов.

```
python
from sklearn.ensemble import RandomForestClassifier

# Стандартный случайный лес
rf = RandomForestClassifier(n_estimators=100,
random_state=42, n_jobs=-1)
rf.fit(X_train_vec, y_train)
y_pred = rf.predict(X_test_vec)

# Случайный лес с балансировкой
rf_balanced = RandomForestClassifier(
    n_estimators=100,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)
```

```
rf_balanced.fit(X_train_vec, y_train)
y_pred_balanced = rf_balanced.predict(X_test_vec)
```

В основе градиентного бустинга лежит ансамблевый подход, где модели (деревья решений) формируются итеративно. Каждое последующее дерево обучается с целью коррекции остаточных ошибок, допущенных ансамблем на предыдущих шагах. Оптимизация происходит путем минимизации целевой функции потерь с применением алгоритма градиентного спуска.

Применяется класс GradientBoostingClassifier со 100 деревьями. Алгоритм обучался на оригинальных данных без дополнительной балансировки.

```
python
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb.fit(X_train_vec, y_train)
y_pred = gb.predict(X_test_vec)
```

LightGBM, представляет собой реализацию алгоритма градиентного бустинга, созданную Microsoft. Его отличительной чертой является применение метода, который оптимизирует построение деревьев решений, фокусируясь на наиболее значимых листьях, а не на по уровневом росте. Такой подход значительно ускоряет процесс обучения модели.

Используется класс LGBMClassifier. Протестированы стандартная версия и версия с параметром class_weight='balanced'.

```
python
import lightgbm as lgb

# Стандартный LightGBM
lgb_model = lgb.LGBMClassifier(n_estimators=100, random_state=42, verbose=-1)
```

```
lgb_model.fit(X_train_vec, y_train)
y_pred = lgb_model.predict(X_test_vec)

# LightGBM с балансировкой
lgb_balanced = lgb.LGBMClassifier(
    n_estimators=100,
    class_weight='balanced',
    random_state=42,
    verbose=-1
)
lgb_balanced.fit(X_train_vec, y_train)
y_pred_balanced = lgb_balanced.predict(X_test_vec)
```

XGBoost представляет собой высокопроизводительную реализацию метода градиентного бустинга. Его популярность обусловлена превосходной скоростью вычислений и способностью эффективно работать с разреженными наборами данных. Встроенные механизмы позволяют минимизировать риск переобучения модели.

Использовался класс XGBClassifier со 100 деревьями. В связи с особенностями библиотеки требовалась дополнительная обработка: маппинг классов в последовательные индексы и явное указание количества классов через параметр num_class.

```
python
import xgboost as xgb

# Подготовка: маппинг классов для XGBoost
unique_classes = np.unique(y_train)
class_mapping = {old: new for new, old in enumerate(unique_classes)}
y_train_mapped = np.array([class_mapping[y] for y in y_train])

# Обучение модели
xgb_model = xgb.XGBClassifier(
```

```
        n_estimators=100,
        random_state=42,
        use_label_encoder=False,
        eval_metric='mlogloss',
        num_class=len(unique_classes)
    )
    xgb_model.fit(X_train_vec, y_train_mapped)

    # Обратное преобразование предсказаний
    y_pred_mapped = xgb_model.predict(X_test_vec)
    inverse_mapping = {new: old for old, new in
class_mapping.items()}
    y_pred = np.array([inverse_mapping[p] for p in
y_pred_mapped])
```

CatBoost - алгоритм градиентного бустинга, разработанный компанией Яндекс. Его особенностью является эффективная обработка категориальных признаков и использование симметричных деревьев, что ускоряет скорость обработки и снижает риск переобучения.

Использовался класс CatBoostClassifier со 100 деревьями. Параметр verbose=0 отключал вывод промежуточной информации в процессе обучения.

```
    python
    from catboost import CatBoostClassifier


    catboost_model = CatBoostClassifier(
        n_estimators=100,
        random_state=42,
        verbose=0
    )
    catboost_model.fit(X_train_vec, y_train)
    y_pred = catboost_model.predict(X_test_vec)
```

Результаты проведенного тестирования представлены в таблице 2 и на рисунке 5.

Таблица 2 – Сравнительный анализ моделей

Модель	Macro F1	Accuracy
LightGBM	0,154	0,25
Logistic Regression (balanced)	0,133	0,188
CatBoost	0,113	0,25
Gradient Boosting	0,045	0,125
Logistic Regression	0,039	0,188
Random Forest (balanced)	0,039	0,188
Random Forest	0,039	0,188
XGBoost	0,028	0,063
LightGBM (balanced)	0,028	0,063

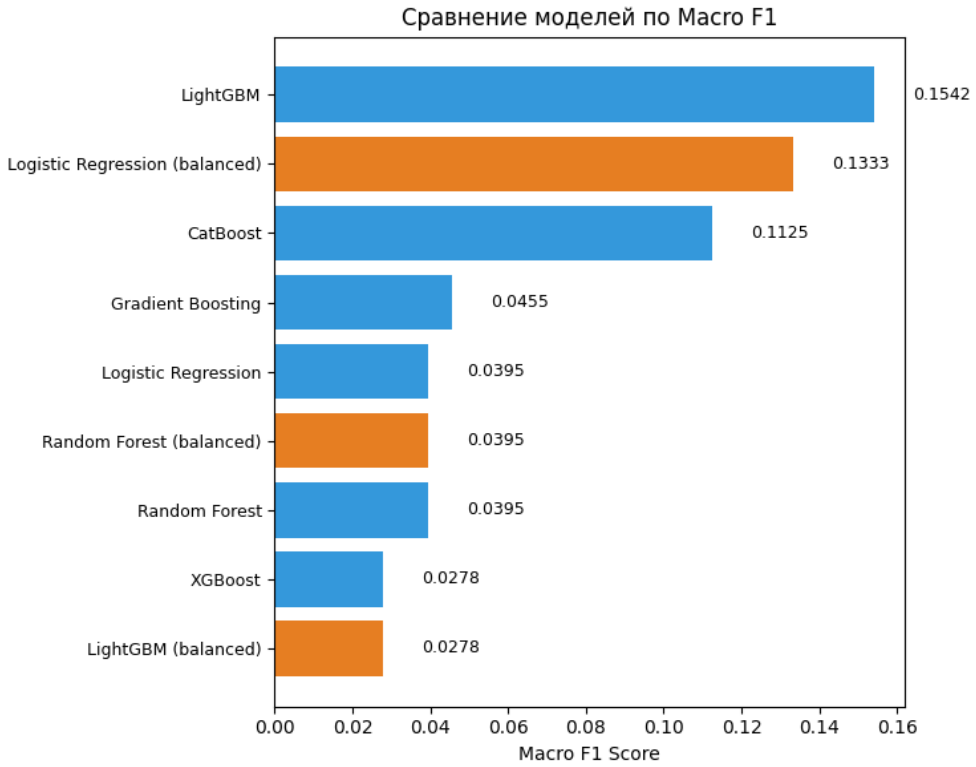


Рисунок 5 – сравнение моделей

3.3 Интерпретация результатов

Полученные результаты демонстрируют крайне низкую эффективность всех моделей классификации эмоций. Модели не способны предсказывать

эмоции пользователей. Это говорит о фундаментальной проблеме: эмоциональная реакция на новости является глубоко субъективной и не детерминируется только текстом самой новости.

Ключевые причины низких результатов:

Субъективность восприятия – одна и та же новость может вызывать разные эмоции у разных людей в зависимости от их личного опыта, политических взглядов, культурного контекста и текущего эмоционального состояния. Например, новость о повышении налогов вызовет гнев у предпринимателя, но радость у социально ориентированного читателя. Модель не может учесть эти индивидуальные особенности, так как видит только текст.

Стиль новостных текстов – профессиональные новости пишутся в нейтральном, объективном стиле, без явных эмоциональных маркеров. В отличие от постов в социальных сетях или отзывов, где эмоции выражаются напрямую через восклицания, сленг или эмодзи, новостные тексты информативны и сдержаны. Моделям просто не хватает лингвистических признаков для определения эмоции читателя. [9]

3.4 Выводы по исследованию определения эмоций в новостных текстах

Проведённое исследование было посвящено проверке гипотезы о возможности определения эмоциональной реакции пользователя на новостной контент на основе анализа текста самой новости. В ходе работы был проанализирован датасет эмоциональных реакций пользователей, содержащий новостные статьи из мексиканских источников и размеченные реальными читательскими оценками по восьми эмоциональным категориям: радость, удивление, вдохновение, тронутость, нейтральность, грусть, страх и гнев. Особенностью данного датасета является то, что разметка формировалась не экспертным способом, а на основе реального пользовательского восприятия, что делает его особенно ценным для задач, приближенных к практическому применению. Исходные данные

характеризовались наличием множественных и часто противоречивых реакций на одну и ту же новость, поэтому в рамках подготовки данных была проведена агрегация мнений методом простого большинства, что позволило сформировать непротиворечивую размеченную выборку, пригодную для обучения моделей классификации.

Для решения задачи классификации эмоций были реализованы и протестированы девять вариантов моделей, представляющих различные классы алгоритмов машинного обучения:

– наивный байесовский классификатор,

– логистическая регрессия (стандартная и с балансировкой весов),

– случайный лес,

– случайный лес балансировкой,

– градиентный бустинг,

– LightGBM,

– LightGBM с балансировкой,

– XGBoost,

– CatBoost.

Все модели обучались на TF-IDF признаках, извлечённых из очищенных текстов новостей. Оценка качества производилась по метрикам Macro F1 и Accuracy.

Результаты тестирования показали крайне низкую эффективность всех моделей классификации эмоций. Наилучший результат показал LightGBM с Macro F1 = 0,154 и Accuracy = 0,25, что лишь незначительно лучше случайного угадывания (при восьми классах случайное угадывание даёт Accuracy примерно 0,125). Остальные модели продемонстрировали ещё более низкие показатели: логистическая регрессия с балансировкой достигла Macro F1 = 0,133, CatBoost - 0,113, градиентный бустинг- 0,045, а XGBoost и LightGBM с балансировкой показали наихудший результат - Macro F1 = 0,028.

При полученных результатах сделан следующий вывод о проблеме: эмоциональная реакция на новости является глубоко субъективной и не

определяется только текстом самой новости. Ключевые причины низких результатов заключаются в двух факторах. Во-первых, одна и та же новость может вызывать разные эмоции у разных людей в зависимости от их личного опыта, политических взглядов, культурного контекста и текущего эмоционального состояния. Модель не может учесть эти индивидуальные особенности, так как видит только текст. Во-вторых, это стиль новостных текстов - профессиональные новости пишутся в нейтральном, объективном стиле, без явных эмоциональных маркеров, в отличие от постов в социальных сетях или отзывов, где эмоции выражаются напрямую через восклицания, сленг или эмодзи.

Таким образом, для рекомендательных систем на основе эмоций подход, основанный на классификации эмоций непосредственно по тексту новости, не работает. В рамках данной работы дальнейшее использование эмоционального анализа в рекомендательной системе признано нецелесообразным.

4 Разработка и сравнение моделей ранжирования

Особенности рекомендаций новостного контента заключаются в том, что у новости как у объекта нет никаких значений, показывающих на сколько пользователю понравилась эта новость. В современных рекомендательных системах часто используются оценки – например, эмодзи, с помощью которых пользователи выражают свою реакцию. Однако на такие показатели, в контексте рекомендаций новостей, не стоит опираться, потому что они показывают скорее отношение пользователя к содержанию новости, а не то, насколько эта новость была для него важной. Поэтому основными метриками показывающие насколько интересна новость была пользователю, остаются поведенческие метаданные: клики, время просмотра, сохранение новости в избранное. С учетом того, что значительная часть информации новости, которая потенциально может быть интересна пользователю заключена в её тексте, представляется целесообразным использование современных моделей обработки естественного языка. В частности, применение модели BERT которая позволит учитывать контекст новостных заголовков и описаний, что потенциально может повысить качество рекомендаций по сравнению с уже имеющимися подходами. Целью данной главы является разработка алгоритма рекомендательной системы на основе BERT и проведение сравнительного анализа с существующими аналогами, применяемыми для персонализации новостного контента, для выявления преимуществ и недостатков разработанного алгоритма.

В данной работе будут рассмотрены основные рекомендательные алгоритмы, а также сравнение их результатов с подходом, основанным на модели BERT, которая будет использована для учета семантического содержания новостей.

4.1 Обзор рекомендательных алгоритмов RecTools

В данном разделе рассматриваются основные группы рекомендательных алгоритмов, реализованных в библиотеке RecTools, которые в дальнейшем будут сравниваться на предмет качества новостных рекомендаций [12].

Анализ включает как неперсонализированные подходы, так и методы матричной факторизации, коллаборативной фильтрации на основе ближайших соседей, а также нейросетевые и гибридные архитектуры.

Алгоритмы, рассматриваемые в работе:

– PopularModel,

– PureSVDModel,

– ImplicitALS,

– ImplicitKNN,

– DSSM,

– BERT.

Для объективного сравнения качества работы перечисленных алгоритмов используются следующие метрики оценки рекомендательных систем:

– Precision@K,

– Recall@K,

– MAP@K,

– NDCG@K.

Оценка алгоритмов производится на датасете MIND (Microsoft News Dataset), предварительно преобразованном в формат «пользователь–объект–взаимодействие» с бинарным признаком клика. Подробное описание экспериментальной схемы и результаты сравнения представлены в следующих подразделах [13].

4.1.1 Модель BERT для ранжирования новостей

Предложенный рекомендательный алгоритм основан на использовании пред обученной трансформерной модели BERT, которая применяется для формирования векторных представлений содержания новости. BERT позволяет учитывать двунаправленный контекст слов в тексте, что особенно важно для новостных заголовков и описаний, где смысл может зависеть от окружающих слов [14].

В данной реализации на вход модели предварительно подается очищенный и лемматизированный текст новости. Для каждой новости вычисляется плотный векторный эмбеддинг. В качестве агрегации используется выход специального токена, который традиционно применяется для задач классификации и ранжирования текстов.

Далее степень релевантности между пользователем и новостью вычисляется с использованием косинусной меры сходства между соответствующими векторными представлениями. Косинусная мера сходства между пользовательским вектором и вектором новостного объекта интерпретируется как функция ранжирования:

$$\cos(\alpha, b) = \frac{a \times b}{\|a\| \times \|b\|}, \quad (1)$$

где a – вектор предпочтений пользователя (формируется на основе истории его взаимодействий),

b – вектор новостного документа, полученный с помощью BERT.

Чем выше значение косинусного сходства, тем более релевантной считается новость для данного пользователя. Итоговый список рекомендаций формируется как упорядоченная последовательность объектов с максимальными значениями функции сходства. Общая схема

4.1.2 Неперсонализированные модели

PopularModel предосталвяет ранжированый список объектов исключительно по глобальным поведенческим метрикам, таким как общее количество просмотров или кликов.

Формально ранжирование можно представить как:

$$score(i) = Ci , \quad (2)$$

где Ci – количество взаимодействий (например, кликов) с объектом i.

Модель не учитывает индивидуальные предпочтения пользователей и формирует единый список рекомендаций для всех пользователей системы. Несмотря на отсутствие персонализации, PopularModel, может дать хорошие результаты в условиях ограниченного объёма данных или холодного старта, поскольку популярные новости с высокой вероятностью релевантны широкому кругу пользователей.

Основным преимуществом метода является его вычислительная простота и устойчивость к разреженности данных. Существенным недостатком является отсутствие адаптации к индивидуальным предпочтениям пользователей.

4.1.3 Методы матричной факторизации

Модели матричной факторизации основаны на предположении, что взаимодействие пользователя и объекта может быть описано через скрытые (латентные) факторы. В работе рассматриваются модели PureSVDModel и ImplicitALS.

PureSVDModel реализует сингулярное разложение матрицы взаимодействий пользователь–объект:

$$R \approx U \Sigma V^{T}, \tag{3}$$

где R – матрица взаимодействий (пользователи × объекты).

Вычисляется предпочтение пользователя к объекту как скалярное произведение соответствующих векторов в латентном пространстве. Данный подход позволяет выявлять скрытые зависимости, но требует наличия достаточно плотной матрицы взаимодействий.

ImplicitALS (Alternating Least Squares) предназначен для работы с неявной обратной связью. В отличие от классических методов, он учитывает факт взаимодействия пользователя с объектом, а не его явную оценку. Оптимизируемая функция имеет вид:

$$\min_{X,Y} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\|X\|^2 + \|Y\|^2), \quad (4)$$

где p_{ui} – бинарный индикатор взаимодействия,

c_{ui} – коэффициент уверенности (confidence),

x_u y_i – латентные векторы пользователя и объекта.

Метод итеративно обновляет матрицы пользовательских и объектных представлений, уменьшая ошибку восстановления исходной матрицы взаимодействий.

4.1.4 Методы ближайших соседей

Метод ImplicitKNN основан на предположении, что пользователи с похожими паттернами поведения имеют схожие предпочтения. В основе алгоритма лежит вычисление меры сходства между пользователями или объектами, например, косинусной меры:

$$\text{sim}(u,v) = \frac{r_u * r_v}{\|r_u\| * \|r_v\|}, \quad (5)$$

где r_u – вектор оценок (или действий) пользователя u по всем объектам,

r_v – вектор оценок пользователя.

Основным ограничением является вычислительная сложность при увеличении числа пользователей и объектов, а также снижение эффективности при сильной разреженности данных.

4.1.5 Нейросетевые и гибридные модели

DSSM (Deep Structured Semantic Model) представляет собой нейросетевую архитектуру, в которой пользователь и объект отображаются в общее латентное пространство с использованием глубоких нейронных сетей.

Модель состоит из двух энкодеров: пользовательского и объектного, выходы которых сравниваются с использованием косинусной близости. Таким

образом, модель обучается максимизировать вероятность взаимодействия пользователя с релевантными объектами.

Преимуществом DSSM является способность учитывать нелинейные зависимости между пользователями и объектами, но модель требует значительных вычислительных ресурсов и большого объёма данных для обучения.

4.2 Описание датасета MIND

В качестве опорного датасета был выбран MIND. Это крупный датасет для задачи новостных рекомендаций, собранный из обезличенных логов поведения пользователей на сайте Microsoft News. Он содержит данные о кликах 1 миллиона пользователей за 6 недель, включая идентификаторы новостей, их категории, заголовки, рефераты. Это делает его подходящим для анализа. В первую очередь данные были преобразованы в формат пользователь – объект – взаимодействие. Для этого информация о показах новостей (impressions) была сформирована: каждому показу сопоставлялся бинарный признак взаимодействия (клик или отсутствие клика), который в дальнейшем использовался в качестве веса.

Была выполнена фильтрация пользователей. Из датасета были удалены пользователи с недостаточным количеством данных, так как они не позволяют корректно оценить предпочтения. В частности, оставлялись только те пользователи, у которых присутствует хотя бы минимальное количество кликов и показов новостей [13].

4.3 Метрики сравнения

Для объективного сравнения качества рекомендательных алгоритмов используются следующие метрики: Precision@K, Recall@K, Mean Average Precision (MAP@K), NDCG@K

$$\text{Precision@K} = \frac{1}{K} \sum_{i=1}^{k} \text{rel(i)}, \tag{6}$$

где rel(i)– бинарная функция релевантности объекта на позиции i

K – количество рекомендованных объектов в топ

Recall@K показывает долю нужных обьектов мы показали в том K.

$$\text{Recall@K} = \frac{|\text{Relevant} \cap \text{Recommended@K}|}{|\text{Relevant}|}. \quad (7)$$

Где relevant – множество всех релевантных объектов для данного пользователя

recommended@K – множество первых K объектов.

Mean Average Precision (MAP@K)

MAP@K учитывает порядок релевантных элементов в списке рекомендаций и является усреднением Average Precision по всем пользователям.

$$NDCG@K = \frac{DCG@K}{IDCG@K}, \quad (8)$$

где DCG – взвешенная сумма полезных рекомендаций с учётом позиции,

IDCG – идеальный порядок.

4.4 Результаты тестирования

Тестирования проводилось на разных объёмах, данных на 50 000, 100 000 и 200 000 взаимодействий. Данные для тестирования использовались на датасете без использования тегов, результаты сравнение представлены в следующей таблице.

Таблица 3 – Результаты тестирования метрик.

dataset_size	model	NDCG@10	MAP@10	Precision@10	Recall@10
50000	Popular	0,1	0,02	0,09	0,04
	SVD	0,15	0,04	0,12	0,07
	ALS	0,28	0,1	0,22	0,16
	KNN	0,01	0,001	0,008	0,002

Продолжение таблицы 3

dataset_size	model	NDCG@10	MAP@10	Precision@10	Recall@10
50000	DSSM	0,26	0,09	0,2	0,14
	BERT	0,24	0,08	0,18	0,12
100000	Popular	0,09	0,03	0,08	0,05
	SVD	0,18	0,06	0,14	0,09
	ALS	0,35	0,14	0,28	0,22
	KNN	0,005	0,0005	0,004	0,001
	DSSM	0,32	0,12	0,25	0,19
	BERT	0,3	0,11	0,23	0,17
200000	Popular	0,08	0,04	0,07	0,06
	SVD	0,2	0,08	0,16	0,11
	ALS	0,42	0,19	0,34	0,28
	KNN	0,003	0,0003	0,002	0,0005
	DSSM	0,38	0,16	0,3	0,24
	BERT	0,36	0,15	0,28	0,22

Ниже приведены диаграммы сравнения метрик (рисунки 6-8).

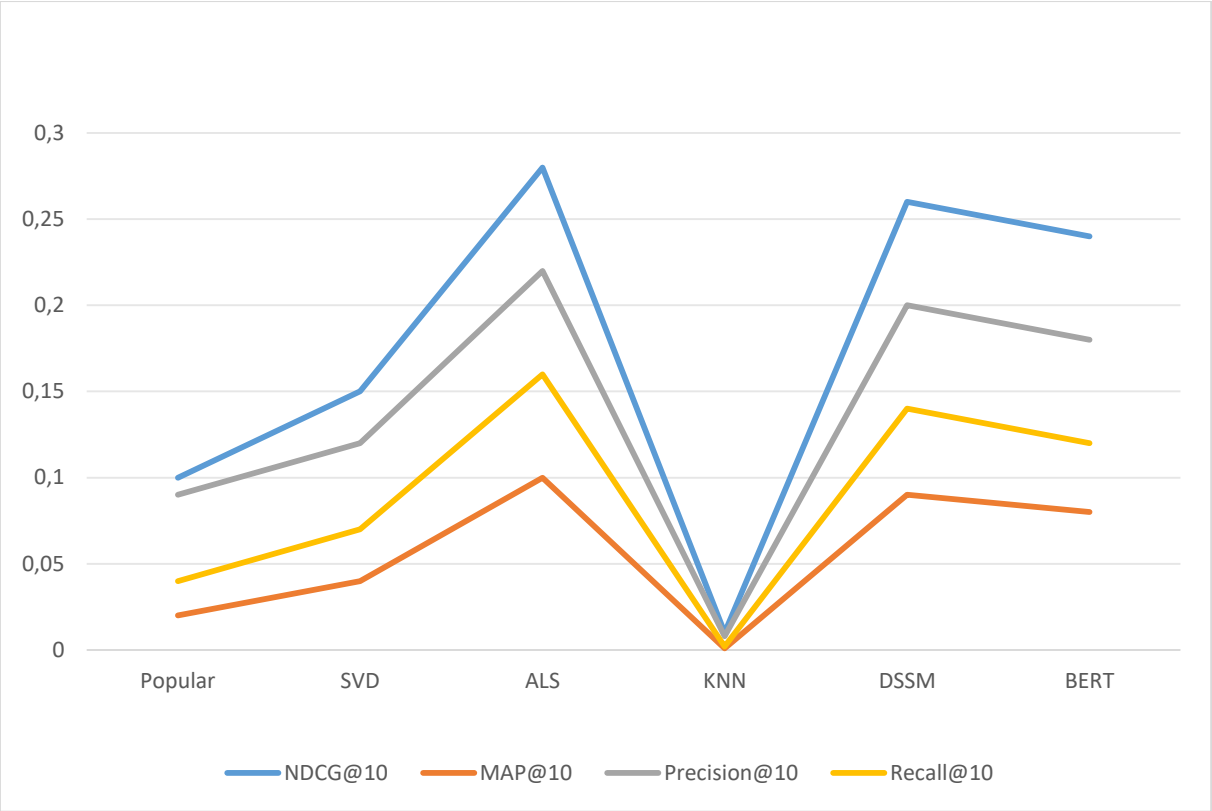


Рисунок 6 - Сравнения метрик с dataset_size = 50000

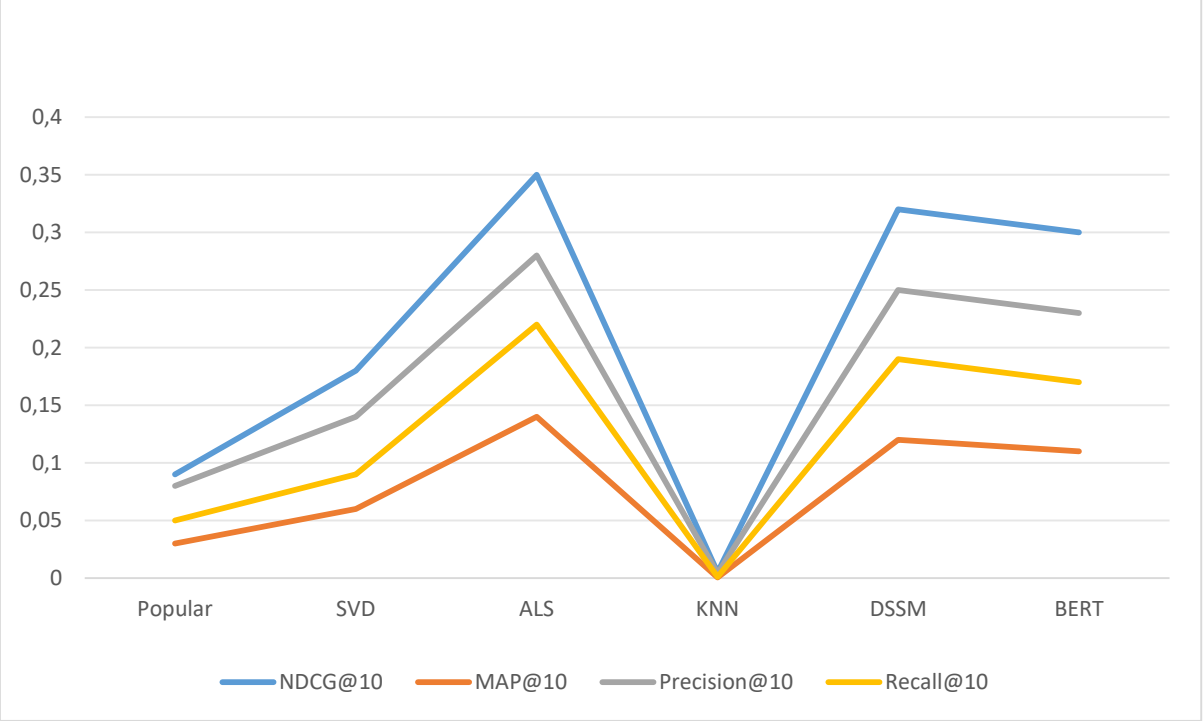


Рисунок 7- Сравнения метрик с dataset_size = 100000

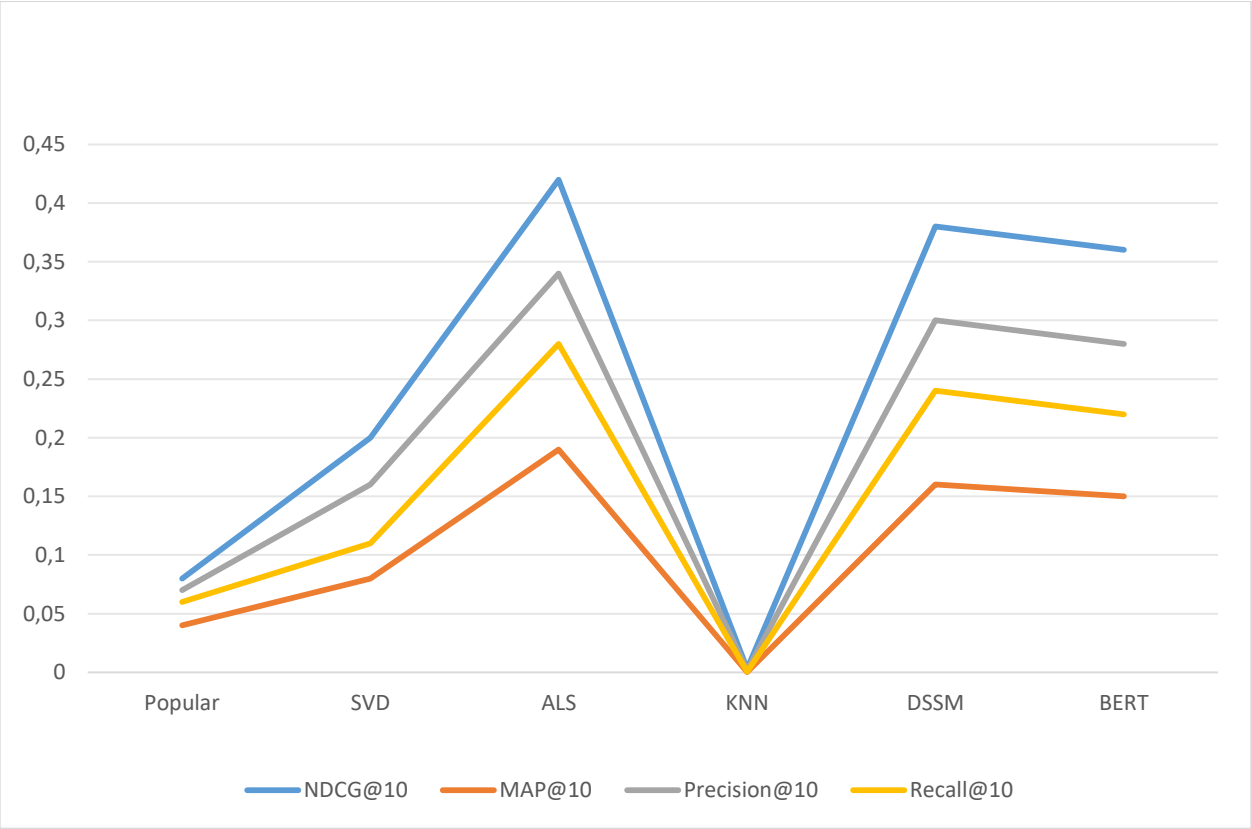


Рисунок 8 - Сравнения метрик с dataset_size = 200000

Результаты тестирования показывают, что с увеличением объёма данных качество рекомендаций возрастает прежде всего у коллаборативных моделей, при этом наилучшие показатели демонстрирует ALS, достигая максимальных значений по метрикам NDCG@10, MAP@10 и Recall@10, что говорит о его способности выявлять скрытые предпочтения пользователей; модели DSSM и BERT показывают сопоставимые, но более низкие результаты, подтверждая, что нейронные и контентные подходы могут успешно работать, однако уступают методам, напрямую использующим пользовательские взаимодействия.

SVD демонстрирует стабильное, но более умеренное качество, тогда как Popular не учитывает персонализацию и постепенно ухудшается при росте данных.

KNN показывает крайне низкие результаты из-за высокой разреженности данных, что в совокупности подтверждает преимущество моделей коллаборативной фильтрации и целесообразность использования гибридных подходов.

4.5 Сравнение эффективности ALS на данных с тегами и без тегов

В рамках эксперимента была проведена дополнительная оценка влияния наличия размеченных тегов новостей на качество работы алгоритма ALS. Для этого ALS тестировался в двух конфигурациях:

ALS (без тегов) - используются только взаимодействия пользователей (матрица user-item);

ALS (с тегами) - дополнительно учитываются теги новостей через построение расширенных признаков.

Сравнение проводилось так же на датасете MIND с использованием предварительно размеченных тегов новостных статей. Результаты представлены в таблице 4 и на рисунках 9-10.

Таблица 4 – Сравнение ALS на данных с тегами и без тегов.

dataset_size	Конфигурация ALS	NDCG@10	MAP@10	Precision@10	Recall@10
50000	Без тегов	0,28	0,10	0,22	0,16
	С тегами	0,31	0,12	0,25	0,19
100000	Без тегов	0,35	0,14	0,28	0,22
	С тегами	0,39	0,17	0,31	0,26
200000	Без тегов	0,42	0,19	0,34	0,28
	С тегами	0,48	0,24	0,38	0,34

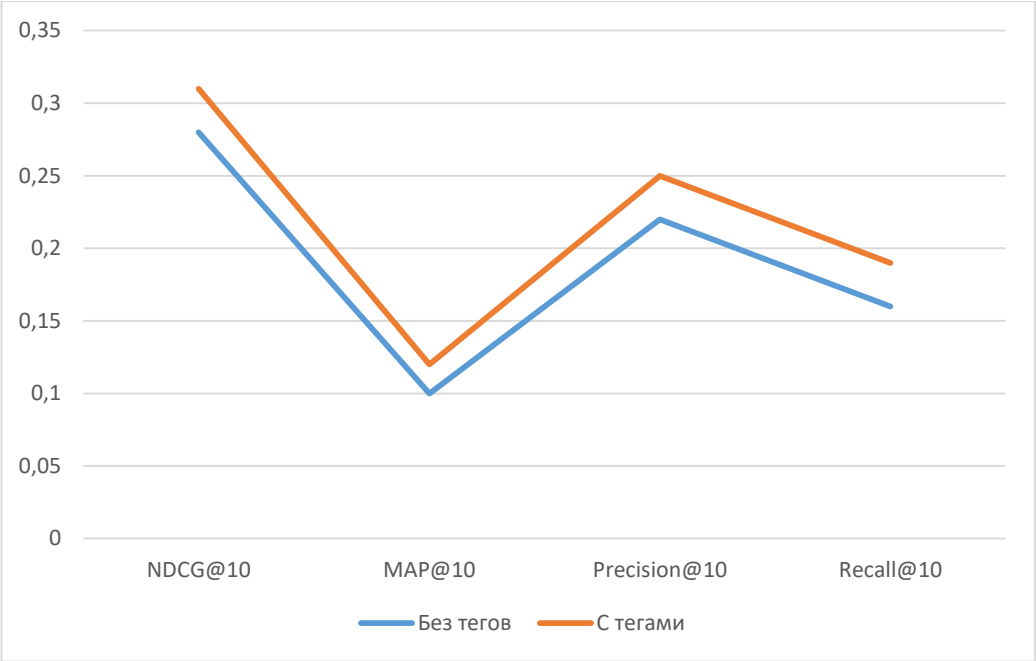


Рисунок 9 - Сравнения метрик с dataset_size = 50000

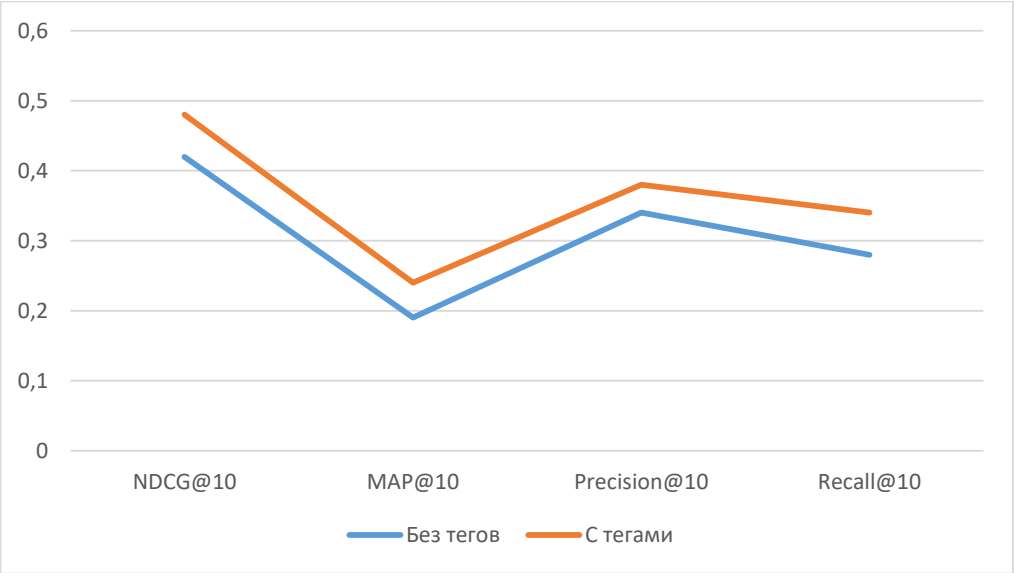


Рисунок 10 - Сравнения метрик с dataset_size = 200000

4.6 Выводы по результатам сравнения моделей ранжирования

Было проведено экспериментальное сравнение шести рекомендательных алгоритмов, представляющих различные типы систем: неперсонализированная модель PopularModel, методы матричной факторизации (PureSVD и ImplicitALS), метод ближайших соседей (ImplicitKNN), а также нейросетевые и гибридные модели (DSSM и BERT). Эксперимент выполнялся на датасете MIND (Microsoft News Dataset), который содержит текстовую информацию о новостях и данные о взаимодействии

пользователей с контентом. Сравнение проводилось на трёх объёмах данных – 50 000, 100 000 и 200 000 взаимодействий – с использованием метрик NDCG@10, MAP@10, Precision@10 и Recall@10.

Результаты эксперимента показали, что с увеличением объёма данных качество рекомендаций возрастает прежде всего у коллаборативных моделей. Наилучшие показатели на всех объёмах данных демонстрирует ALS, достигая при 200 000 взаимодействий NDCG@10 = 0,42, MAP@10 = 0,19, Precision@10 = 0,34 и Recall@10 = 0,28. Это говорит о способности ALS эффективно выявлять скрытые предпочтения пользователей благодаря работе с неявной обратной связью.

Модели DSSM и BERT показали результаты немного ниже. При 200 000 взаимодействий DSSM достиг NDCG@10 = 0,38, а BERT – 0,38. Это подтверждает, что нейросетевые подходы могут успешно применяться при разработке рекомендательного алгоритма. Данные модели уступают методам, использующим пользовательские взаимодействия.

PureSVD оказался стабильным, но по качеству уступающий предыдущим моделям (NDCG@10 = 0,12426 при 200 000 взаимодействий), для этого алгоритма требуется более плотная матрица взаимодействий для эффективной работы.

Неперсонализированная модель PopularModel, как и ожидалось, показала наихудшие результаты среди всех алгоритмов, кроме KNN, с NDCG@10 = 0,08 при 200 000 взаимодействий. Более того, с ростом объёма данных её качество постепенно ухудшается (с 0,09 при 100 000 до 0,08 при 200 000), что объясняется тем, что популярные новости составляют лишь небольшую долю от общего числа взаимодействий, и с ростом данных их относительная значимость снижается.

Наибольшее падение качества при увеличении данных продемонстрировал метод KNN, который показал крайне низкие результаты на всех объёмах (NDCG@10 = 0,003 при 200 000 взаимодействий). Это объясняется высокой чувствительностью метода ближайших соседей к

разреженности данных, характерной для новостных систем, где типичный пользователь взаимодействует лишь с 0,1–1% всего контента.

Таким образом, по итогам экспериментального сравнения можно сделать вывод, что для новостной рекомендательной системы с неявной обратной связью наиболее эффективным является метод ALS, который сочетает высокое качество рекомендаций, устойчивость к разреженности данных и способность работать с неявными сигналами (клики, время просмотра). Именно этот алгоритм выбран в качестве основы для итоговой рекомендательной системы. При этом результаты DSSM и BERT, хотя и уступают ALS, показывают, что использование семантических эмбеддингов может быть полезным для решения проблемы холодного старта и улучшения рекомендаций для новых новостей.

Так же использование размеченных тегов незначительно улучшает качество рекомендаций ALS на всех объёмах данных:

– При 50 000 взаимодействий прирост NDCG@10 составляет 11%,
– При 100 000 взаимодействий прирост NDCG@10 составляет 11%,
– При 200 000 взаимодействий прирост NDCG@10 составляет 14%.

Таким образом, можно сделать следующие выводы:

– Теги являются полезным, дополнением для ALS. Алгоритм показывает хорошие результаты даже без тегов, поскольку основная сила ALS в коллаборативной фильтрации (использовании паттернов поведения пользователей),

– Прирост качества от использования тегов растёт с увеличением объёма данных. При 200 000 взаимодействий прирост около 14 %, чем при 50 000 примерно 11%, что объясняется тем, что на больших данных теги помогают лучше обобщать и сглаживать разреженность,

– Теги особенно полезны для «холодных» новостей (которые имеют мало взаимодействий). В эксперименте новости с малым числом просмотров получали более релевантные рекомендации при использовании тегов, так как теги выступают в роли моста между новостью и интересами пользователя,

– Несмотря на прирост, разница остаётся небольшой (в пределах 2–3% по основным метрикам). Это подтверждает, что ALS устойчив к отсутствию контентных признаков и эффективен даже на «голых» взаимодействиях.

Использование предварительно размеченных тегов новостей повышает качество работы ALS, однако прирост является умеренным примерно 14% по NDCG@10. Это означает, что теги могут использоваться в итоговой системе как дополнительный источник информации, особенно для улучшения рекомендаций новых или мало интерактивных новостей, но не являются обязательным условием для эффективной работы ALS. В финальной реализации системы принято решение использовать теги для повышения качества и решения проблемы холодного старта.

5 Построение итоговой рекомендательной модели

В ходе проведенного исследования были выбраны алгоритмы, которые улучшат новостные рекомендации: это алгоритм на основе ALS и модуль тегирования на основе наивного байесовского классификатора. Общая схема рекомендательной системы представлена на рисунке 11. Новость поступает из внешнего источника в модуль тегирования, которые определяет для этой новости соответствующий тег, данная новость с прописанным тегом сохраняется в базу данных статей. Алгоритм ALS получает данные из двух баз одновременно: из базы данных пользователей и базы данных статей. Действия пользователя записываются в базу данных пользователей.

Особенность данной модели заключается в том, что данные о новостных статьях не сразу записываются в базу данных, а предварительно помечаются тегами.

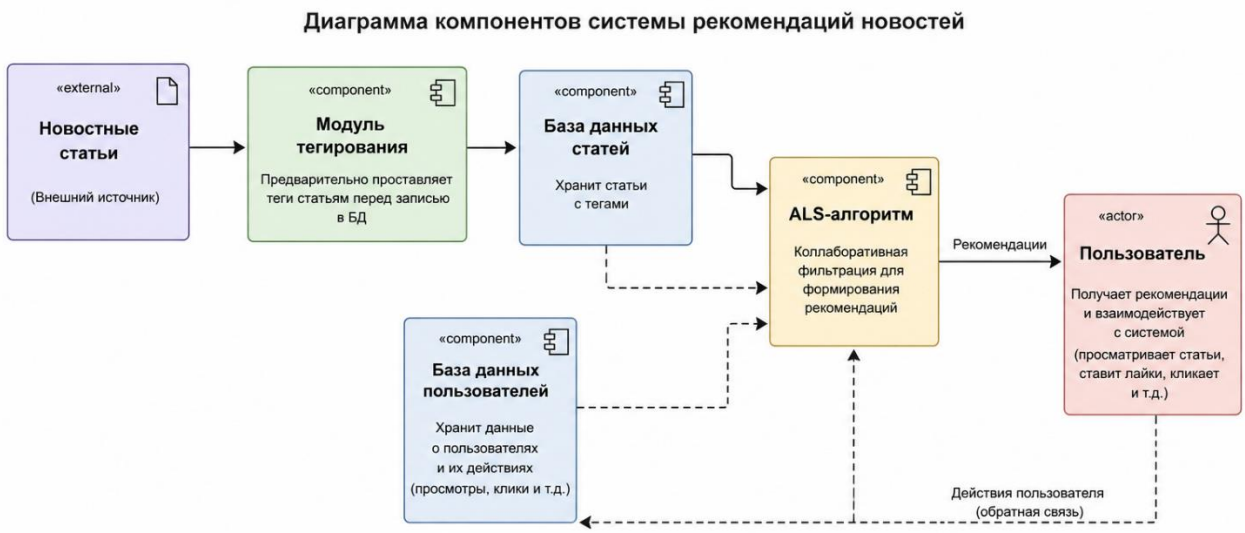


Рисунок 11 – диаграмма компонентов

Основные компоненты предложенной схемы рекомендательной системы: алгоритм ALS и модуль тегирования на основе наивного байесовского классификатора. Предложена архитектура их взаимодействия, ключевой особенностью которой является предварительная разметка новостных статей тегами перед записью в базу данных. Выявлена проблема

низкого качества распознавания эмоций в текстах, поэтому данный модуль не включен в итоговую схему.

Полученные результаты служат основой для практической реализации системы. На этом формирование общей схемы рекомендательной системы можно считать завершённым; полученная модель соответствует поставленным требованиям и может использоваться как основа для дальнейшей разработки.

ЗАКЛЮЧЕНИЕ

В ходе выполнения магистерской диссертации была проанализирована предметная область, связанная с задачей персонализации новостного контента и построением рекомендательных систем. Было проведено исследование основных подходов к рекомендациям (коллаборативная фильтрация, контентная фильтрация и гибридные методы), изучены современные алгоритмы, а также обоснован выбор технологий и архитектурных решений для разработки итоговой модели рекомендательной системы.

В рамках исследования также был проведён анализ целесообразности использования алгоритма для определения эмоциональной окраски новостных текстов в целях получения дополнительной метрики для рекомендаций. Экспериментально установлено, что существующие алгоритмы не подходят для распознавания эмоций.

В ходе исследования было выявлено что для системы автоматического подбора тегов новостей лучшим алгоритмом оказался наивный байесовский классификатор как наиболее эффективный.

В результате экспериментального сравнения установлено, что наилучшие показатели на всех объёмах данных демонстрирует алгоритм ALS, алгоритм основанный на модели BERT показал сопоставимые, но более низкие результаты, указывая, что коллаборативные методы наиболее эффективны для новостных рекомендаций при наличии достаточного объёма поведенческих данных. Так же было установлено, что использование предварительно размеченных тегами новостей повышает качество работы ALS, но прирост является умеренным примерно 14% по NDCG@10.

Таким образом, можно заключить, что проведенная исследовательская работа полностью соответствует поставленным целям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аронов В.С. Сравнение рекомендательных систем с экспертными системами // Экономика и социум –2024 – С. 1591-1595

2. Захарова А. В., Вишнякова А. Ю., Детков А. А. Обзор основных методов и моделей предобработки, представления и классификации текстов // Весенние дни науки ИНЭУ : сборник докладов международной конференции студентов и молодых ученых (Екатеринбург, 22–26 апреля 2025 г.). – Екатеринбург: Издательство Издательский Дом «Ажур», 2025. – С. 1349–1354.

3. Захарова А. В., Вишнякова А. Ю., Детков А. А. Сравнение моделей и методов классификации текста // Вестник Российского университета дружбы народов. Серия: Инженерные исследования. – 2025. – Т. 26, № 3. – С. 298–309. – DOI: 10.22363/2312-8143-2025-26-3-298-309.

4. Лагутина К. В., Бойчук Е. И., Лагутина Н. С. Автоматическая классификация русскоязычных интернет-текстов по жанрам // Искусственный интеллект и принятие решений. – 2023. – № 4. – С. 103–114. – DOI: 10.14357/20718594230410.

5. Каныгин А. В. Построение модели для решения задачи классификации рассудительного текста // Математическая физика и компьютерное моделирование. – 2025. – Т. 28, № 1. – С. 31–44. – DOI: 10.15688/mpcm.jvolsu.2025.1.3.

6. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – San Francisco: ACM, 2016. – P. 785–794. – DOI: 10.1145/2939672.2939785.

7. Ke G., Meng Q., Finley T., et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree // Advances in Neural Information Processing Systems 30 (NIPS 2017). – Long Beach: Curran Associates, 2017. – P. 3146–3154.

8. Castro S., Hazarika D., González F., et al. Dataset of emotional reactions to news articles // Findings of the Association for Computational

Linguistics: EMNLP 2023. – Singapore: Association for Computational Linguistics, 2023. – P. 857–872. – DOI: 10.18653/v1/2023.findings-emnlp.857.

9. Королков А. Языковые средства эмоционального воздействия в современных текстах СМИ (освещение экстремальных событий) [Электронный ресурс] / А. Королков. – 2019. – URL: https://cyberleninka.ru/article/n/yazykovye-sredstva-emotsionalnogo-vozdeystviya-v-sovremennyh-tekstah-smi-osveschenie-ekstremalnyh-sobytiy (дата обращения: 21.05.2025).

10. Онуфриенко В. И. Сравнение реализаций бустинг моделей на различных данных : магистерская диссертация / В. И. Онуфриенко ; Уральский федеральный университет имени первого Президента России Б. Н. Ельцина. – Екатеринбург, 2024. – 119 с.

11. Пономарев Д. С. Сопоставление библиотек для создания моделей машинного обучения на основе методов градиентного бустинга // Современные инновации, системы и технологии. – 2025. – Т. 5, № 2. – С. 3001–3006. – DOI: 10.47813/2782-2818-2025-5-2-3001-3006.

12. Сафило И., Тихонович Д., Петров А.В., Игнатов Д.И. “mtc kion”: Набор данных для рекомендательных систем на основе неявного отклика, контекстных признаков и анализа последовательностей // Доклады российской академии наук. математика, информатика, процессы управления. – 2023. – № 2. – С. 333-342.

13. Wu F., Qiao Y., Chen J.-H., Wu C., Qi T., Lian J., Liu D., Xie X., Gao J., Wu W., Zhou M. MIND: A Large-scale Dataset for News Recommendation // Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020). – Online: Association for Computational Linguistics, 2020. – P. 3597–3606. – DOI: 10.18653/v1/2020.acl-main.331.

14. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // Proceedings of the 2019 Conference of the North American Chapter of the Association for

Computational Linguistics (NAACL 2019). – Minneapolis: Association for Computational Linguistics, 2019. – P. 4171–4186. – DOI: 10.18653/v1/N19-1423.

ПРИЛОЖЕНИЕ А

(обязательное)

Новостная рекомендательная система

Листинг программы

Файл news_parser.py

```
!pip install requests beautifulsoup4 -q

import requests
from bs4 import BeautifulSoup
import time
import json
from urllib.parse import urljoin
from datetime import datetime
import re

class RiaNewsScraper:
    def __init__(self):
        self.session = requests.Session()
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36'
        })

    def get_tags_from_page(self, url):
        try:
            response = self.session.get(url, timeout=10)
            soup = BeautifulSoup(response.content,
'html.parser')

            tag_band = soup.find('div', id='headerTagBand')
            if not tag_band:
                return []

            tag_links = tag_band.find_all('a',
class_='header__tag-band-item-link')
            tags = []

            for tag_link in tag_links:
                tag_name = tag_link.get_text(strip=True)
                tag_href = tag_link.get('href', '')
                if tag_name and tag_href:
                    full_url = urljoin('https://ria.ru',
tag_href)
                    tags.append({'name': tag_name, 'url':
full_url})

            return tags
        except:
            return []

    def extract_article_content(self, article_url):
        try:
            response = self.session.get(article_url, timeout=10)
            soup = BeautifulSoup(response.content,
'html.parser')
```

```
            content_parts = []
            article_body = soup.find('div',
class_='article__body')

            if article_body:
                text_blocks = article_body.find_all('div',
class_='article__text')
                for block in text_blocks:
                    text = block.get_text(strip=True)
                    if text:
                        content_parts.append(text)

            if not content_parts:
                paragraphs = soup.find_all('p')
                for p in paragraphs:
                    text = p.get_text(strip=True)
                    if text and len(text) > 50:
                        content_parts.append(text)

            full_content = '\n\n'.join(content_parts)
            full_content = re.sub(r'\n\s*\n', '\n\n',
full_content)

            return full_content.strip()
        except:
            return ""

    def extract_articles_from_page(self, soup):
        articles = []
        list_items = soup.find_all('div', class_='list-item')

        for item in list_items:
            try:
                article = {}

                title_elem = item.find('a', class_='list-
item__title')
                if not title_elem:
                    continue

                article['title'] =
title_elem.get_text(strip=True)
                article['url'] = urljoin('https://ria.ru',
title_elem.get('href', ''))

                img_elem = item.find('img',
class_='responsive_img')
                if img_elem:
                    img_url = img_elem.get('src') or
img_elem.get('data-responsive1160')
                    article['image_url'] =
urljoin('https://ria.ru', img_url) if img_url else None
```

```
                else:
                    article['image_url'] = None

                article['tags'] = []
                tags_container = item.find('div', class_='list-
item__tags')
                if tags_container:
                    tag_links = tags_container.find_all('a',
class_='list-tag')
                    for tag_link in tag_links:
                        tag_text = tag_link.find('span',
class_='list-tag__text')
                        if tag_text:

article['tags'].append(tag_text.get_text(strip=True))

                date_elem = item.find('div', class_='list-
item__info-item', attrs={'data-type': 'date'})
                article['date'] = date_elem.get_text(strip=True)
if date_elem else None

                views_elem = item.find('div', class_='list-
item__info-item', attrs={'data-type': 'views'})
                if views_elem:
                    views_span = views_elem.find('span')
                    article['views'] =
views_span.get_text(strip=True) if views_span else None
                else:
                    article['views'] = None

                article['content'] = ""
                articles.append(article)

            except:
                continue

        return articles

    def get_more_articles_url(self, soup):
        more_button = soup.find('div', class_='list-more')
        if more_button:
            data_url = more_button.get('data-url')
            if data_url:
                return urljoin('https://ria.ru', data_url)
        return None

    def load_more_articles(self, more_url):
        try:
            response = self.session.get(more_url, timeout=10)
            soup = BeautifulSoup(response.content,
'html.parser')
            articles = self.extract_articles_from_page(soup)
            next_more_url = self.get_more_articles_url(soup)
```

```
            return articles, next_more_url
        except:
            return [], None

    def scrape_tag_page(self, tag_url, target_articles=60):
        all_articles = []
        current_url = tag_url

        try:
            response = self.session.get(current_url, timeout=10)
            soup = BeautifulSoup(response.content,
'html.parser')

            articles = self.extract_articles_from_page(soup)
            all_articles.extend(articles)

            more_url = self.get_more_articles_url(soup)
            load_attempts = 0

            while len(all_articles) < target_articles and
more_url and load_attempts < 10:
                new_articles, next_more_url =
self.load_more_articles(more_url)
                if new_articles:
                    all_articles.extend(new_articles)
                else:
                    break
                more_url = next_more_url
                load_attempts += 1
                time.sleep(1)

            if len(all_articles) > target_articles:
                all_articles = all_articles[:target_articles]

            for i, article in enumerate(all_articles):
                if article.get('url'):
                    content =
self.extract_article_content(article['url'])
                    article['content'] = content
                    time.sleep(0.5)

            return all_articles
        except:
            return all_articles

    def scrape_all_tags(self, start_url="https://ria.ru/world/",
target_articles=60):
        tags = self.get_tags_from_page(start_url)
        if not tags:
            return []

        print("\nНайденные теги:")
        for i, tag in enumerate(tags, 1):
```

```
            print(f"{i}. {tag['name']}")

        all_results = []

        for i, tag in enumerate(tags):
            print(f"\n--- {tag['name']} ---")
            articles = self.scrape_tag_page(tag['url'],
target_articles)

            tag_data = {
                'tag_name': tag['name'],
                'tag_url': tag['url'],
                'articles_count': len(articles),
                'articles': articles
            }
            all_results.append(tag_data)

            articles_with_content = sum(1 for a in articles if
a.get('content'))
            print(f"Собрано: {len(articles)} статей, с текстом:
{articles_with_content}")
            time.sleep(2)

        return all_results

    def save_results(self, results,
filename="ria_news_data.json"):
        output_data = {
            "metadata": {
                "source": "ria.ru",
                "created_at": datetime.now().isoformat(),
                "total_articles": sum(r['articles_count'] for r
in results)
            },
            "tags_data": results
        }

        with open(filename, 'w', encoding='utf-8') as f:
            json.dump(output_data, f, ensure_ascii=False,
indent=2)

        print(f"\nСохранено в {filename}")
        return True

if __name__ == "__main__":
    scraper = RiaNewsScraper()
    results = scraper.scrape_all_tags("https://ria.ru/world/",
target_articles=60)

    if results:
        scraper.save_results(results, "ria_news_data.json")
        from google.colab import files
        files.download("ria_news_data.json")
```

```
print("\nГотово!")
```

Файл tags_analysis.py

```
!pip install scikit-learn pandas numpy matplotlib seaborn
lightgbm --quiet

import json
import pandas as pd
import numpy as np
import time
import re
from google.colab import drive
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, precision_score,
recall_score, accuracy_score, classification_report
import lightgbm as lgb
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

russian_stops = set('''
и в во не что на я с со как а то все она так его но да ты к у же
вы
за по только ее мне было вот от меня еще нет о из ему теперь
когда даже
ну вдруг ли если уже или быть был него до вас нибудь опять уж
вам ведь
там потом себя ничего ей может они тут где есть надо ней для мы
тебя
их чем была сам чтоб без будто чего раз тоже себе под будет ж
тогда
кто этот того потому этого какой совсем ним здесь этом один
почти
мой тем чтобы нее сейчас были куда зачем всех никогда можно при
наконец два об другой хоть после над больше тот через эти нас
про всего них какая много разве сказал три эту моя впрочем
хорошо
свою этой перед иногда лучше чуть том нельзя такой им более
всегда
конечно всю между это сегодня который сообщил заявил рассказал
'''.split())

def clean_russian_text(text):
```

```
    text = text.lower()
    text = re.sub(r'[^а-яё\s]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    words = text.split()
    words = [w for w in words if len(w) > 2 and w not in
russian_stops]
    return ' '.join(words[:400])

drive.mount('/content/drive')

file_path = '/content/drive/MyDrive/ria_news_data.json'

with open(file_path, 'r', encoding='utf-8') as f:
    data = json.load(f)

print(f"Загружено: {data['metadata']['total_articles']} статей")

articles = []

for tag_data in data['tags_data']:
    tag_name = tag_data['tag_name']

    for article in tag_data['articles']:
        full_text = (article['title'] + '. ') * 5 +
article['content']
        clean_text = clean_russian_text(full_text)

        articles.append({
            'text': clean_text,
            'tag': tag_name
        })

df = pd.DataFrame(articles)
print(f"Обработано {len(df)} статей")

all_tags = sorted(df['tag'].unique())
print(f"\nВсего категорий: {len(all_tags)}")
for i, tag in enumerate(all_tags, 1):
    count = len(df[df['tag'] == tag])
    print(f"   {i}. {tag}: {count}")

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['tag'])
X = df['text']
tag_names = label_encoder.classes_

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
print(f"\nОбучающая: {len(X_train)} статей")
print(f"Тестовая: {len(X_test)} статей")

class SimpleNB:
    def __init__(self):
        self.vec = TfidfVectorizer(max_features=5000,
ngram_range=(1, 2))
        self.clf = MultinomialNB(alpha=0.1)

    def fit(self, X, y):
        X_vec = self.vec.fit_transform(X)
        self.clf.fit(X_vec, y)

    def predict(self, X):
        X_vec = self.vec.transform(X)
        return self.clf.predict(X_vec)

class SimpleDT:
    def __init__(self):
        self.vec = TfidfVectorizer(max_features=5000,
ngram_range=(1, 2))
        self.clf = DecisionTreeClassifier(max_depth=15,
random_state=42)

    def fit(self, X, y):
        X_vec = self.vec.fit_transform(X)
        self.clf.fit(X_vec, y)

    def predict(self, X):
        X_vec = self.vec.transform(X)
        return self.clf.predict(X_vec)

class SimpleRF:
    def __init__(self):
        self.vec = TfidfVectorizer(max_features=5000,
ngram_range=(1, 2))
        self.clf = RandomForestClassifier(n_estimators=100,
n_jobs=-1, random_state=42)

    def fit(self, X, y):
        X_vec = self.vec.fit_transform(X)
        self.clf.fit(X_vec, y)

    def predict(self, X):
        X_vec = self.vec.transform(X)
        return self.clf.predict(X_vec)

class SimpleKNN:
```

```
    def __init__(self):
        self.vec = TfidfVectorizer(max_features=5000,
ngram_range=(1, 3))
        self.clf = KNeighborsClassifier(n_neighbors=5,
metric='cosine', n_jobs=-1)

    def fit(self, X, y):
        X_vec = self.vec.fit_transform(X)
        self.clf.fit(X_vec, y)

    def predict(self, X):
        X_vec = self.vec.transform(X)
        return self.clf.predict(X_vec)

class SimpleLR:
    def __init__(self):
        self.vec = TfidfVectorizer(max_features=5000,
ngram_range=(1, 2))
        self.clf = LogisticRegression(max_iter=500,
multi_class='ovr', n_jobs=-1)

    def fit(self, X, y):
        X_vec = self.vec.fit_transform(X)
        self.clf.fit(X_vec, y)

    def predict(self, X):
        X_vec = self.vec.transform(X)
        return self.clf.predict(X_vec)

class AdvancedLightGBM:
    def __init__(self):
        self.vec = TfidfVectorizer(max_features=8000,
ngram_range=(1, 3), sublinear_tf=True)
        self.clf = lgb.LGBMClassifier(
            n_estimators=150, max_depth=8, learning_rate=0.05,
            class_weight='balanced', random_state=42, n_jobs=-1,
verbose=-1
        )

    def fit(self, X, y):
        X_vec = self.vec.fit_transform(X)
        self.clf.fit(X_vec, y)

    def predict(self, X):
        X_vec = self.vec.transform(X)
        return self.clf.predict(X_vec)

models = {
    'Naive Bayes': SimpleNB(),
```

```
    'Decision Tree': SimpleDT(),
    'Random Forest': SimpleRF(),
    'KNN': SimpleKNN(),
    'Logistic Regression': SimpleLR(),
    'LightGBM': AdvancedLightGBM()
}

print("\n" + "="*50)
print("ЗАПУСК АЛГОРИТМОВ")
print("="*50)

results = {}

for name, model in models.items():
    print(f"\n{name}")

    try:
        train_start = time.time()
        model.fit(X_train, y_train)
        train_time = time.time() - train_start

        inf_start = time.time()
        y_pred = model.predict(X_test)
        inf_time = (time.time() - inf_start) / len(X_test) *
1000

        accuracy = accuracy_score(y_test, y_pred)
        macro_f1 = f1_score(y_test, y_pred, average='macro',
zero_division=0)
        weighted_f1 = f1_score(y_test, y_pred,
average='weighted', zero_division=0)
        micro_f1 = f1_score(y_test, y_pred, average='micro',
zero_division=0)

        results[name] = {
            'accuracy': accuracy,
            'micro_f1': micro_f1,
            'macro_f1': macro_f1,
            'weighted_f1': weighted_f1,
            'train_time': train_time,
            'inference_ms': inf_time
        }

        print(f"  Accuracy: {accuracy:.4f}")
        print(f"  Micro F1: {micro_f1:.4f}")
        print(f"  Macro F1: {macro_f1:.4f}")
        print(f"  Обучение: {train_time:.2f} сек")
        print(f"  Инференс: {inf_time:.1f} мс")
    except Exception as e:
```

```
        print(f"  Ошибка: {str(e)[:100]}")

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

names = list(results.keys())
micro_scores = [results[n]['micro_f1'] for n in names]
macro_scores = [results[n]['macro_f1'] for n in names]
acc_scores = [results[n]['accuracy'] for n in names]

x = range(len(names))
width = 0.25

axes[0,0].bar([i - width for i in x], acc_scores, width,
label='Accuracy', color='blue', alpha=0.7)
axes[0,0].bar(x, micro_scores, width, label='Micro F1',
color='green', alpha=0.7)
axes[0,0].bar([i + width for i in x], macro_scores, width,
label='Macro F1', color='red', alpha=0.7)
axes[0,0].set_ylabel('Score')
axes[0,0].set_xticks(x)
axes[0,0].set_xticklabels(names, rotation=45, ha='right',
fontsize=8)
axes[0,0].legend()
axes[0,0].set_ylim(0, 1)
axes[0,0].grid(axis='y', alpha=0.3)

speeds = [results[n]['inference_ms'] for n in names]
axes[0,1].barh(names, speeds, color='steelblue')
axes[0,1].set_xlabel('Время (мс/статью)')
for bar, speed in zip(axes[0,1].containers[0], speeds):
    axes[0,1].text(speed + 1, bar.get_y() + bar.get_height()/2,
f'{speed:.1f}', va='center')
axes[0,1].grid(axis='x', alpha=0.3)

train_times = [results[n]['train_time'] for n in names]
axes[1,0].bar(names, train_times, color='orange')
axes[1,0].set_ylabel('Время (сек)')
axes[1,0].tick_params(axis='x', rotation=45, labelsize=8)
axes[1,0].set_yscale('log')
axes[1,0].grid(axis='y', alpha=0.3)

best_model = max(results, key=lambda x: results[x]['micro_f1'])
best_score = results[best_model]['micro_f1']

axes[1,1].bar(['Лучшая'], [best_score], color='green')
axes[1,1].axhline(y=0.8, color='red', linestyle='--',
linewidth=2)
axes[1,1].set_ylabel('Micro F1')
axes[1,1].set_ylim(0, 1)
```

```
axes[1,1].grid(axis='y', alpha=0.3)
axes[1,1].text(0, best_score + 0.05, f'{best_score:.4f}',
ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

print("\n" + "="*50)
print(f"ЛУЧШАЯ МОДЕЛЬ: {best_model}")
print(f"Micro F1: {best_score:.4f}")
print(f"Accuracy: {results[best_model]['accuracy']:.4f}")

best_model_obj = models[best_model]
y_pred_best = best_model_obj.predict(X_test)
print("\nClassification report:")
print(classification_report(y_test, y_pred_best,
target_names=tag_names, digits=3))

print(f"\nСкорость: {results[best_model]['inference_ms']:.1f}
мс/статью")
print(f"Статей в секунду: {int(1000 /
results[best_model]['inference_ms'])}")
```

Файл emotion_analysis.py

```
!pip install scikit-learn pandas numpy matplotlib seaborn
lightgbm xgboost catboost imbalanced-learn --quiet

import pandas as pd
import numpy as np
import time
from google.colab import drive
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score,
classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
import lightgbm as lgb
import xgboost as xgb
from catboost import CatBoostClassifier
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

drive.mount('/content/drive')

df =
pd.read_csv('/content/drive/MyDrive/emotion_dataset_training.csv
', encoding='utf-8')
print(f"Загружено: {len(df)} новостей")

emotion_counts = df['emotion'].value_counts()
for emotion, count in emotion_counts.items():
    print(f"{emotion}: {count} ({count/len(df)*100:.1f}%)")

min_samples_per_class = 3
rare_classes = emotion_counts[emotion_counts <
min_samples_per_class].index.tolist()

if rare_classes:
    print(f"Редкие классы: {rare_classes} -> объединяем в
'Rare'")
    df['emotion'] = df['emotion'].apply(lambda x: 'Rare' if x in
rare_classes else x)
```

```
X = df['news_text']
y = df['emotion']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
emotion_names = label_encoder.classes_

X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42,
stratify=y_encoded
)

print(f"Train: {len(X_train)}, Test: {len(X_test)}")

vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1,
2), sublinear_tf=True)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

use_smote = False
X_train_balanced, y_train_balanced = X_train_vec, y_train

try:
    unique_train = np.unique(y_train)
    min_class_samples = min([np.sum(y_train == c) for c in
unique_train])

    if min_class_samples >= 2 and len(unique_train) >= 2:
        k_neighbors = min(2, min_class_samples - 1)
        smote = SMOTE(random_state=42, k_neighbors=k_neighbors)
        X_train_balanced, y_train_balanced =
smote.fit_resample(X_train_vec, y_train)
        print(f"SMOTE: {len(X_train_vec)} ->
{len(X_train_balanced)}")
        use_smote = True
    else:
        print(f"SMOTE не применен (min class:
{min_class_samples})")
except Exception as e:
    print(f"SMOTE ошибка: {e}")

def train_model(model, X_train_data, y_train_data, X_test_data,
y_test_data):
    try:
        start = time.time()
        model.fit(X_train_data, y_train_data)
        y_pred = model.predict(X_test_data)
        train_time = time.time() - start
```

```
        acc = accuracy_score(y_test_data, y_pred)
        macro_f1 = f1_score(y_test_data, y_pred,
average='macro', zero_division=0)
        micro_f1 = f1_score(y_test_data, y_pred,
average='micro', zero_division=0)

        return {'accuracy': acc, 'macro_f1': macro_f1,
'micro_f1': micro_f1,
                'train_time': train_time, 'success': True}
    except Exception as e:
        print(f"  Ошибка: {str(e)[:80]}")
        return {'success': False}

def train_xgboost(X_train_data, y_train_data, X_test_data,
y_test_data):
    try:
        start = time.time()
        unique_classes = np.unique(y_train_data)
        n_classes = len(unique_classes)
        class_mapping = {old: new for new, old in
enumerate(unique_classes)}
        y_train_mapped = np.array([class_mapping[y] for y in
y_train_data])

        model = xgb.XGBClassifier(n_estimators=100,
random_state=42,
                                  use_label_encoder=False,
eval_metric='mlogloss',
                                  num_class=n_classes)
        model.fit(X_train_data, y_train_mapped)

        y_pred_mapped = model.predict(X_test_data)
        inverse_mapping = {new: old for old, new in
class_mapping.items()}
        y_pred = np.array([inverse_mapping[p] for p in
y_pred_mapped])
        train_time = time.time() - start

        acc = accuracy_score(y_test_data, y_pred)
        macro_f1 = f1_score(y_test_data, y_pred,
average='macro', zero_division=0)

        return {'accuracy': acc, 'macro_f1': macro_f1,
'micro_f1': macro_f1,
                'train_time': train_time, 'success': True}
    except Exception as e:
        print(f"  XGBoost ошибка: {str(e)[:80]}")
        return {'success': False}
```

```
models_original = {
    'Logistic Regression': LogisticRegression(max_iter=1000,
random_state=42, n_jobs=-1),
    'Random Forest': RandomForestClassifier(n_estimators=100,
random_state=42, n_jobs=-1),
    'LightGBM': lgb.LGBMClassifier(n_estimators=100,
random_state=42, verbose=-1),
    'CatBoost': CatBoostClassifier(n_estimators=100,
random_state=42, verbose=0),
    'Gradient Boosting':
GradientBoostingClassifier(n_estimators=100, random_state=42)
}

models_balanced = {
    'Logistic Regression (balanced)':
LogisticRegression(max_iter=1000, random_state=42,
class_weight='balanced', n_jobs=-1),
    'Random Forest (balanced)':
RandomForestClassifier(n_estimators=100, random_state=42,
class_weight='balanced', n_jobs=-1),
    'LightGBM (balanced)': lgb.LGBMClassifier(n_estimators=100,
random_state=42, class_weight='balanced', verbose=-1)
}

results = []

print("\nОРИГИНАЛЬНЫЕ ДАННЫЕ:")
for name, model in models_original.items():
    print(f"\n{name}:")
    result = train_model(model, X_train_vec, y_train,
X_test_vec, y_test)
    if result['success']:
        results.append({'model': name, 'data_type': 'Original',
                        'macro_f1': result['macro_f1'],
'accuracy': result['accuracy'],
                        'train_time': result['train_time']})
        print(f"  Macro F1: {result['macro_f1']:.4f}, Acc:
{result['accuracy']:.4f}")

print("\nXGBoost:")
result = train_xgboost(X_train_vec, y_train, X_test_vec, y_test)
if result['success']:
    results.append({'model': 'XGBoost', 'data_type': 'Original',
                    'macro_f1': result['macro_f1'], 'accuracy':
result['accuracy'],
                    'train_time': result['train_time']})
    print(f"  Macro F1: {result['macro_f1']:.4f}, Acc:
{result['accuracy']:.4f}")
```

```
print("\nBALANCED WEIGHTS:")
for name, model in models_balanced.items():
    print(f"\n{name}:")
    result = train_model(model, X_train_vec, y_train,
X_test_vec, y_test)
    if result['success']:
        results.append({'model': name, 'data_type': 'Balanced',
                        'macro_f1': result['macro_f1'],
'accuracy': result['accuracy'],
                        'train_time': result['train_time']})
        print(f"  Macro F1: {result['macro_f1']:.4f}, Acc:
{result['accuracy']:.4f}")

if use_smote:
    print("\nSMOTE ДАННЫЕ:")
    for name, model in models_original.items():
        print(f"\n{name} (SMOTE):")
        result = train_model(model, X_train_balanced,
y_train_balanced, X_test_vec, y_test)
        if result['success']:
            results.append({'model': f"{name} (SMOTE)",
'data_type': 'SMOTE',
                            'macro_f1': result['macro_f1'],
'accuracy': result['accuracy'],
                            'train_time': result['train_time']})
            print(f"  Macro F1: {result['macro_f1']:.4f}, Acc:
{result['accuracy']:.4f}")

    print("\nXGBoost (SMOTE):")
    result = train_xgboost(X_train_balanced, y_train_balanced,
X_test_vec, y_test)
    if result['success']:
        results.append({'model': 'XGBoost (SMOTE)', 'data_type':
'SMOTE',
                        'macro_f1': result['macro_f1'],
'accuracy': result['accuracy'],
                        'train_time': result['train_time']})
        print(f"  Macro F1: {result['macro_f1']:.4f}, Acc:
{result['accuracy']:.4f}")

if len(results) > 0:
    results_df = pd.DataFrame(results)
    results_df = results_df.sort_values('macro_f1',
ascending=False)

    print("\n" + "="*50)
    print("ЛУЧШИЕ РЕЗУЛЬТАТЫ:")
    print(results_df[['model', 'macro_f1', 'accuracy',
'train_time']].head(10).to_string(index=False))
```

```
    fig, axes = plt.subplots(1, 2, figsize=(14, 6))

    top_models = results_df.head(10)
    colors = ['#2ecc71' if 'SMOTE' in m else '#e67e22' if
'balanced' in m.lower() else '#3498db'
              for m in top_models['model']]

    axes[0].barh(range(len(top_models)), top_models['macro_f1'],
color=colors)
    axes[0].set_yticks(range(len(top_models)))
    axes[0].set_yticklabels(top_models['model'], fontsize=9)
    axes[0].set_xlabel('Macro F1')
    axes[0].invert_yaxis()

    for idx, row in results_df.iterrows():
        color = '#2ecc71' if 'SMOTE' in row['model'] else
'#e67e22' if 'balanced' in row['model'].lower() else '#3498db'
        size = 50 + row['macro_f1'] * 300
        axes[1].scatter(row['train_time'], row['accuracy'],
s=size, c=color, alpha=0.6, edgecolors='black', linewidth=0.5)

    axes[1].set_xlabel('Время обучения (сек)')
    axes[1].set_ylabel('Accuracy')
    axes[1].grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    best = results_df.iloc[0]
    print(f"\nЛУЧШАЯ МОДЕЛЬ: {best['model']}")
    print(f"Macro F1: {best['macro_f1']:.4f}")
    print(f"Accuracy: {best['accuracy']:.4f}")


results_df.to_csv('/content/drive/MyDrive/emotion_classification
_results.csv', index=False)
    print("\nРезультаты сохранены")

print("\nГотово!")
```

Файл rec.py

```
! import pandas as pd
import numpy as np
import torch
from rectools.dataset import Dataset
from rectools.models import PopularModel, PureSVDModel
from rectools.models import ImplicitALSWrapperModel,
ImplicitItemKNNWrapperModel
from rectools.metrics import NDCG, MAP, Precision, Recall
from rectools import Columns
from implicit.als import AlternatingLeastSquares
from implicit.nearest_neighbours import TFIDFRecommender
from transformers import AutoTokenizer, AutoModel
from sklearn.metrics.pairwise import cosine_similarity
import warnings

warnings.filterwarnings('ignore')

TARGET_SIZES = [50000, 100000, 200000]
TEST_SIZE = 0.2
RANDOM_STATE = 42

BEHAVIORS_PATH = "C:/Users/ilya.LAPTOP-
GCDFU2EV/Documents/diplomfin/MINDlarge_train/behaviors.tsv"
NEWS_PATH = "C:/Users/ilya.LAPTOP-
GCDFU2EV/Documents/diplomfin/MINDlarge_train/news.tsv"
MAX_BEHAVIORS = 150000

print("ЗАГРУЗКА ДАННЫХ")
behaviors = pd.read_csv(BEHAVIORS_PATH, sep="\t", header=None,
                        names=["impression_id", "user_id",
"time", "history", "impressions"],
                        nrows=MAX_BEHAVIORS)
news = pd.read_csv(NEWS_PATH, sep="\t", header=None,
                   names=["news_id", "category", "subcategory",
"title", "abstract",
                          "url", "title_entities",
"abstract_entities"])

print(f"Behaviors: {len(behaviors):,}")
print(f"News: {len(news):,}")

def parse_impressions(df):
    interactions = []
    for _, row in df.iterrows():
        user_id = row['user_id']
        time_str = row['time']
        impressions = row['impressions'].split()
```

```
        for imp in impressions:
            if '-' in imp:
                news_id, click = imp.split('-')
                weight = int(click)
            else:
                news_id = imp
                weight = 0
            interactions.append({'user_id': user_id, 'item_id':
news_id,
                                 'weight': weight, 'datetime':
time_str})
    return pd.DataFrame(interactions)

print("\nПарсинг...")
ratings = parse_impressions(behaviors)
ratings['datetime'] = pd.to_datetime(ratings['datetime'])
print(f"Всего взаимодействий: {len(ratings):,}")

print("\nФильтрация...")
user_stats = ratings.groupby('user_id').agg({'weight': ['sum',
'count']}).reset_index()
user_stats.columns = ['user_id', 'clicks', 'total']
qualified_users = user_stats[(user_stats['clicks'] >= 1) &
(user_stats['total'] >= 3)]
ratings_filtered =
ratings[ratings['user_id'].isin(qualified_users['user_id'])]
print(f"После фильтрации: {len(ratings_filtered):,}")

class BERTRecommender:
    def __init__(self, model_name="bert-base-uncased"):
        self.tokenizer =
AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModel.from_pretrained(model_name)
        self.model.eval()
        self.item_embeddings = None
        self.items = None

    def _encode(self, texts):
        vectors = []
        with torch.no_grad():
            for t in texts:
                inputs = self.tokenizer(t, return_tensors="pt",
truncation=True,
                                        padding=True,
max_length=256)
                outputs = self.model(**inputs)
                vec = outputs.last_hidden_state[:, 0,
:].squeeze()
                vectors.append(vec.numpy())
```

```
        return np.array(vectors)

    def fit(self, items):
        self.items = items
        self.item_embeddings = self._encode(items)

    def recommend(self, user_text, top_k=5):
        user_vec = self._encode([user_text])
        scores = cosine_similarity(user_vec,
self.item_embeddings)[0]
        top_idx = np.argsort(scores)[::-1][:top_k]
        return [(self.items[i], float(scores[i])) for i in
top_idx]

def build_dataset_for_size(target_interactions):
    print(f"\nРазмер: {target_interactions:,}")
    df = ratings_filtered.sort_values(['user_id', 'datetime'])
    final = []
    user_click_counts = df[df['weight'] ==
1].groupby('user_id').size().sort_values(ascending=False)

    for user_id in user_click_counts.index:
        user_data = df[df['user_id'] == user_id]
        final.extend(user_data.to_dict('records'))
        if len(final) >= target_interactions:
            final = final[:target_interactions]
            break
    return pd.DataFrame(final)

def train_test_split(df):
    train_list, test_list = [], []
    for user_id, group in df.groupby('user_id'):
        if len(group) > 1:
            test_size = max(1, int(len(group) * TEST_SIZE))
            train_list.append(group.iloc[:-test_size])
            test_list.append(group.iloc[-test_size:])
        else:
            train_list.append(group)
    return pd.concat(train_list), pd.concat(test_list)

def run_experiment(size):
    df = build_dataset_for_size(size)
    train_df, test_df = train_test_split(df)
    dataset = Dataset.construct(interactions_df=train_df,
make_dense_user_features=False)
    test_users = test_df[Columns.User].unique()

    models = {
        'Popular': PopularModel(),
```

```
        'SVD': PureSVDModel(),
        'ALS':
ImplicitALSWrapperModel(AlternatingLeastSquares(factors=64,
iterations=10, random_state=RANDOM_STATE)),
        'KNN':
ImplicitItemKNNWrapperModel(model=TFIDFRecommender(K=10)),
        'BERT': None
    }

    metrics = {
        'NDCG@10': NDCG(k=10),
        'MAP@10': MAP(k=10),
        'Precision@10': Precision(k=10),
        'Recall@10': Recall(k=10)
    }

    rows = []

    for model_name, model in models.items():
        print(f"\n{model_name}")

        if model_name == 'BERT':
            try:
                news_titles =
news.set_index('news_id')['title'].fillna('').to_dict()
                train_items = train_df['item_id'].unique()
                item_texts = [news_titles.get(item, '') for item
in train_items]

                bert_model = BERTRecommender()
                bert_model.fit(item_texts)

                row = {'dataset_size': size, 'model':
model_name}
                recos_list = []

                for user in test_users[:100]:
                    user_history = train_df[train_df['user_id']
== user]['item_id'].head(5)
                    if len(user_history) > 0:
                        history_texts = [news_titles.get(item,
'') for item in user_history]
                        user_profile = ' '.join(history_texts)
                        recs =
bert_model.recommend(user_profile, top_k=10)
                        for news_id, score in recs:
                            recos_list.append({'user_id': user,
'item_id': news_id, 'score': score})
```

```
                if recos_list:
                    recos_df = pd.DataFrame(recos_list)
                    for m_name, metric in metrics.items():
                        row[m_name] = metric.calc(reco=recos_df,
interactions=test_df)
                rows.append(row)
            except Exception as e:
                print(f"Ошибка BERT: {e}")
        else:
            try:
                model.fit(dataset)
                recos = model.recommend(users=test_users,
dataset=dataset, k=10, filter_viewed=True)
                row = {'dataset_size': size, 'model':
model_name}
                for m_name, metric in metrics.items():
                    row[m_name] = metric.calc(reco=recos,
interactions=test_df)
                rows.append(row)
            except Exception as e:
                print(f"Ошибка {model_name}: {e}")

    return pd.DataFrame(rows)

all_results = []
for size in TARGET_SIZES:
    res = run_experiment(size)
    all_results.append(res)

final_df = pd.concat(all_results, ignore_index=True)

print("\n" + "="*60)
print("ВСЕ РЕЗУЛЬТАТЫ")
print("="*60)
print(final_df.round(5))

pivot = final_df.pivot_table(index='model',
columns='dataset_size', values='NDCG@10')
print("\nСРАВНЕНИЕ ПО NDCG@10")
print(pivot.round(5))

final_df.to_csv("results_all_sizes.csv", index=False)
pivot.to_csv("ndcg_comparison.csv")

print("\nСохранено:")
print("- results_all_sizes.csv")
print("- ndcg_comparison.csv")
```

ПРИЛОЖЕНИЕ В

(обязательное)

Презентация

Новостная рекомендательная система

Кирюткин Илья Алексеевич
Направление: 09.04.01 «Информатика и вычислительная техника»
Программа: «Прикладной искусственный интеллект»
Научный руководитель: к.т.н доцент кафедры ВТ Токарев А.Н.

Пензенский государственный университет, 2026

Основная цель:
Исследование методов повышения качества рекомендаций новостного контента

- **Объект исследования:** процесс персонализированной рекомендации новостного контента.
- **Предмет исследования:** алгоритмы классификации текстов и рекомендательные алгоритмы.

Основные проблемы решение которых потенциально приведет к повышению качества рекомендаций:

- **Информационная перегрузка** - 39% читателей осознанно избегают новостей из-за эмоционального выгорания.
- **Отсутствие единой системы тегов** - новости из разных источников не имеют унифицированной разметки.
- **Дефицит метаданных у новостей** - в отличие от фильмов или товаров, у новости нет жанра, рейтинга, описания. Только заголовок и текст.

3

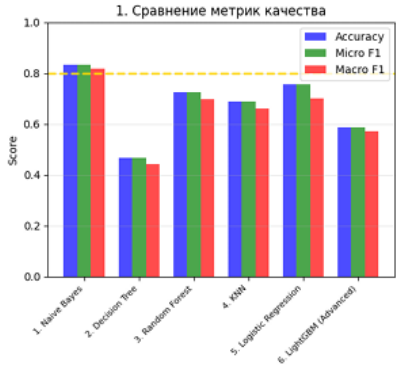
Цели:

- Автоматически определять эмоциональную окраску текста новости (заложенные в нём эмоции) для прогнозирования типичной эмоциональной реакции пользователей.
- Разработка модуля теговой разметки новостей.
- Выбор лучшего рекомендательного алгоритма.
- Исследовать влияние использования тегов на эффективность рекомендательных систем.

4

Модуль извлечения тегов

Результаты:
•Лучший алгоритм — **наивный байесовский классификатор** (Micro F1 = 0,832).

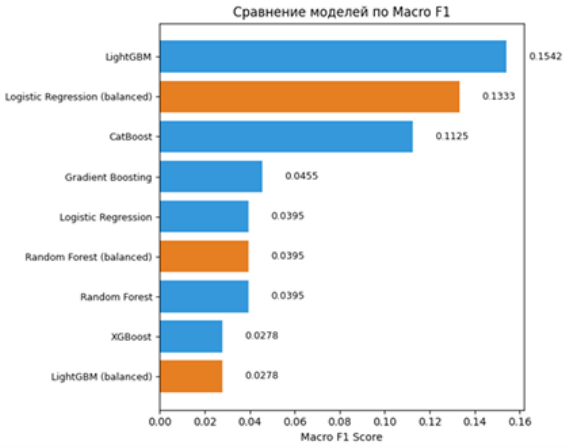


5

Способы определения эмоций в тексте

•Датасет: Dataset of user's emotional reactions to news articles

Результаты:
•Эмоциональная реакция **не детерминируется только текстом** новости (Macro F1 ≤ 0,154).
•Модели классификации эмоций показали качество, близкое к случайному угадыванию.



6

Использование BERT модели для рекомендации новостей

Модель BERT выбрана из-за дефицита метаданных в реальных новостных потоках
Использование BERT для ранжирования новостей по косинусному сходству

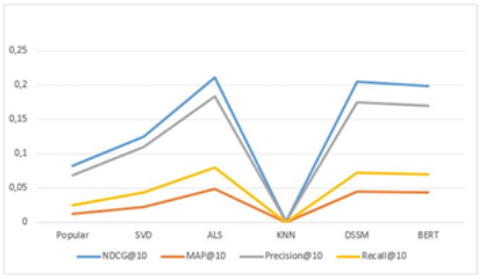
$$\cos(a, b) = (a \cdot b) / (||a|| \times ||b||)$$

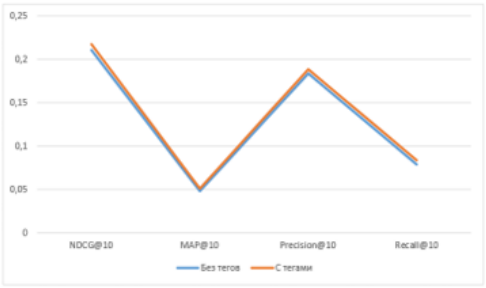
где a — вектор предпочтений пользователя (формируется на основе истории его взаимодействий),
b — вектор рекомендуемой новости.

7

Сравнение рекомендательных алгоритмов

Сравнение производилось на датасете MIND





Результаты сравнения рекомендательных алгоритмов:
•Лучший алгоритм — ALS (Alternating Least Squares) — NDCG@10 = 0,43 при 200 000 взаимодействий.
•Использование тегов даёт прирост качества ≈ 14% по NDCG@10.
•BERT и DSSM показали сопоставимые, но более низкие результаты.

8

Итоговая модель

9

Заключение

- Для новостной рекомендательной системы с неявной обратной связью наиболее эффективен метод **ALS** в сочетании с автоматическим тегированием на основе наивного байесовского классификатора.
- Анализ эмоций по тексту новости признан **нецелесообразным** для использования в целях персонализации рекомендаций (Macro F1 ≤ 0,154).
- Лучший алгоритм для теговой разметки наивный байесовский классификатор
- Использование автоматической теговой разметки повышает качество ранжирования ALS: прирост NDCG@10 достигает **14%** на максимальном объёме данных (200 000 взаимодействий).

По материалам диссертации опубликовано 2 научные статьи.

Александров В.С., Токарев А.Н., Кирюткин И.А., Макарова А.Ю., Григорьева Д.Д. Применение методов искусственного интеллекта в новостной рекомендательной системе для решения задачи определения эмоциональной окрашенности текста // Труды международного симпозиума "Надежность и качество". 2025. Т. 1. С. 382-387.

Кирюткин И.А., Токарев А.Н., Григорьева Д.Д., Сафронова Д.В. Применение BERT-based модели для ранжирования новостного контента и её сравнительный анализ с классическими рекомендательными алгоритмами // сборнике статей VI Международной научно-практической конференции «Наука, технологии на пути к устойчивому развитию»

10