

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**АВТОМАТИЧЕСКАЯ ТЕМАТИЧЕСКАЯ КЛАССИФИКАЦИЯ
НОВОСТНОГО МАССИВА**

БАКАЛАВРСКАЯ РАБОТА

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Кондрашова Даниила Владиславовича

Научный руководитель
доцент, д. ф.-м. н.

С. В. Папшев

Заведующий кафедрой
к. ф.-м. н.

С. В. Миронов

Саратов 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Теоретические и методологические основы автоматической тематической классификации	7
1.1 Место автоматической классификации новостей в разведывательном поиске	7
1.2 Сбор новостных данных	8
1.2.1 Выбор метода получения новостных данных	8
1.2.2 Подбор новостной платформы для сбора данных	8
1.3 Подготовка собранных данных	9
1.4 Математические основы тематического моделирования	11
1.4.1 Основная гипотеза тематического моделирования	11
1.4.2 Аксиоматика тематического моделирования	11
1.4.3 Задача тематического моделирования	12
1.4.4 Решение обратной задачи	13
1.4.5 Регуляризаторы в тематическом моделировании	16
1.4.6 Оценка качества моделей	19
1.5 Методические основы работы с текстом с помощью нейросетей	21
1.5.1 Проблема представления текста	21
1.5.2 Выбор архитектуры нейронной сети	23
1.5.3 Оценка качества работы нейронных сетей	24
2 Практико-технологические основы автоматической тематической классификации	27
2.1 Получение новостного массива путём веб-скраппинга	27
2.1.1 Выбор инструментов получения новостных данных	27
2.1.2 Реализация алгоритма сбора новостных данных	27
2.2 Подготовка новостного массива	30
2.2.1 Выбор инструментов для подготовки данных	30
2.2.2 Удаление лишних пробелов и переносов строк	31
2.2.3 Разделение строк на русские и английские фрагменты	32
2.2.4 Очистка от неалфавитных токенов и удаление крайних неалфавитных символов из токенов	34
2.2.5 Токенизация, лемматизация и удаление стоп-слов по словарю	35
2.2.6 Удаление высокочастотных и низкочастотных токенов	36

2.2.7	Удаление стоп-слов с помощью метрики TF-IDF	37
2.2.8	Очистка набора данных от пустых документов	39
2.3	Количественные характеристики обработанного и необработанного датасета	40
2.4	Вычисление тематической модели	42
2.4.1	Выбор инструментов для тематического моделирования	42
2.4.2	Недостающий функционал библиотеки BigARTM	43
2.4.3	Функциональности классов <code>My_BigARTM_model</code> и <code>Hyperparameter_optimizer</code>	43
2.4.4	Преобразование новостного массива в приемлемый для BigARTM формат	44
2.4.5	Удобное добавление регуляризаторов	45
2.4.6	Вычисление когерентности	46
2.4.7	Вычисление тематической модели и формирование графиков метрик	47
2.4.8	Подбор гиперпараметров для тематического моделирования	48
2.5	Результаты тематического моделирования	50
2.6	Обучение модели классификатора	52
2.6.1	Выбор модели для тематической классификации	52
2.6.2	Выбор способа для получения предобученных моделей	52
2.6.3	Получение весов предобученной модели	53
2.6.4	Подготовка данных для работы с моделью	53
2.6.5	Дообучение модели	54
2.7	Результаты обучения классификатора	56
2.8	Выводы и возможные улучшения по практико-методической части	57
ЗАКЛЮЧЕНИЕ		58
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		60
Приложение А	Пример страницы новостного сайта ВШЭ	63
Приложение Б	Листинг вебскрапера	64
Приложение В	Листинг комплексной обработки текста: нормализация, токенизация, лемматизация, фильтрация стоп-слов по словарю	67
Приложение Г	Листинг обработчика новостного массива	68
Приложение Д	Листинг функции вычисления тематической модели с пошаговым расчётом метрик	77

Приложение Е	Полный код класса <code>My_BigARTM_model</code>	78
Приложение Ж	Листинг функции выбора оптимальной конфигурации	87
Приложение З	Полный код класса <code>Hyperparameter_optimizer</code>	87
Приложение И	Количественные характеристики подготовленного и неподготовленного новостного массива	92
Приложение К	Полный код обучения модели классификатора	97

ВВЕДЕНИЕ

В настоящее время обработка больших объёмов текстовых данных, включая новостные потоки, становится критически важной задачей. Как в научной среде, так и в бизнесе требуется оперативно анализировать информацию, отслеживать тенденции и принимать решения. Однако анализ всего массива данных невозможен из-за его масштабов. Необходимо фильтровать информацию, оставляя только релевантную. Решением этой проблемы может стать тематическая классификация.

Хотя многие сайты и порталы предлагают рубрикацию контента, её точность часто оказывается низкой: теги присваиваются некорректно или поверхностно. Это приводит к ошибкам в поиске и анализе информации. Для устранения этих недостатков необходим механизм, обеспечивающий точную тематическую классификацию данных с возможностью автоматической разметки новостей. Одним из инструментов для реализации такого подхода являются тематические модели в сочетании с алгоритмами глубокого обучения. Первые позволяют выявить скрытые темы в текстовых данных и подготовить разметку для обучения вторых. Алгоритмы глубокого обучения, в свою очередь, могут классифицировать новые тексты по заданным темам.

Таким образом, целью данной работы является реализация одного из возможных механизмов автоматической тематической классификации новостей с использованием методов тематического моделирования и глубокого обучения.

Для достижения цели необходимо решить следующие задачи:

1. Реализовать сбор новостных данных;
2. Разработать механизм предобработки текстовых данных;
3. Выполнить подготовку данных;
4. Вычислить количественные характеристики данных и провести их анализ;
5. Выполнить тематическое моделирование подготовленных данных с различными параметрами;
6. Выбрать оптимальную тематическую модель с помощью сравнительного анализа;
7. Разметить данные для обучения нейронной сети-классификатора с помощью тематического моделирования;
8. Выполнить обучение нейронной сети-классификатора на размеченных данных;

9. Провести анализ метрик качества, вычисленных при обучении;
10. Сделать вывод об эффективности или неэффективности предложенного варианта реализации автоматической тематической классификации.

1 Теоретические и методологические основы автоматической тематической классификации

1.1 Место автоматической классификации новостей в разведывательном поиске

Разведывательный поиск — это процесс сбора, анализа и интерпретации информации из открытых источников для поддержки принятия решений в различных сферах: от бизнеса до государственного управления. В условиях информационной перегрузки автоматическая классификация новостей становится ключевым инструментом, обеспечивающим структуризацию и фильтрацию данных. Её задача — преобразовать неупорядоченные массивы текстов в категоризированные наборы, которые могут быть эффективно использованы для дальнейшего анализа.

Интеграция в процесс разведывательного поиска:

1. Сбор данных: новостные потоки формируют основу для разведывательного поиска. Однако их объёмы и разнообразие форматов затрудняют ручную обработку;
2. Предварительная обработка: автоматическая классификация группирует статьи по темам, сокращая время на первичный анализ;
3. Целевой анализ: категоризированные данные позволяют экспертам фокусироваться на конкретных аспектах — например, отслеживать кризисные события или выявлять скрытые тенденции.

Практическая значимость:

1. Скорость обработки: ручная классификация тысяч новостных статей в день невозможна. Алгоритмы на базе BigARTM и глубокого обучения справляются с этим за минуты, обеспечивая актуальность данных для принятия решений;
2. Масштабируемость: автоматизация позволяет работать с постоянно растущими объёмами информации без значительного увеличения ресурсных затрат;
3. Снижение субъективности: исключаются человеческие ошибки, связанные с усталостью или предвзятостью, что повышает достоверность результатов.

Автоматическая классификация новостей не заменяет экспертов, но становится их основным помощником, беря на себя рутинные задачи. В разведыва-

тельном поиске это критически важно, так как позволяет перейти от обработки данных к их осмысленному использованию — будь то стратегическое планирование или оперативное управление. Технологии вроде BigARTM и методов глубокого обучения обеспечивают баланс между скоростью, точностью и адаптивностью, что делает их незаменимыми в работе с динамичными новостными потоками.

1.2 Сбор новостных данных

1.2.1 Выбор метода получения новостных данных

Для получения данных с сайтов существует три основных метода:

- Ручной сбор — извлечение информации человеком вручную;
- Запрос данных — получение информации от владельцев с последующим скачиванием;
- Программный сбор — автоматизированное извлечение данных.

Первый метод можно исключить из рассмотрения из-за низкой эффективности. Второй метод применим не во всех случаях: владельцы информационных платформ вряд ли будут оперативно предоставлять данные по каждому запросу. Таким образом, наиболее целесообразным остаётся третий метод — программный сбор.

Среди методов программного сбора оперативно и эффективно получать данные в большинстве случаев позволяют инструменты веб-скрапинга [1], который мы выбираем в качестве основного подхода. Далее в работе будет использован именно этот метод для формирования новостного массива, так как он прост в изучении, а также обеспечивает баланс между скоростью получения данных и минимальными требованиями к стороннему участию.

1.2.2 Подбор новостной платформы для сбора данных

В рамках данной работы основным объектом исследования являются новостные текстовые данные. Для их сбора необходимо выбрать подходящий веб-ресурс.

При наличии нескольких потенциальных источников выбор следует осуществлять на основе анализа HTML структуры сайта по следующим критериям:

1. Единая структура документов на всём сайте;
2. Отсутствие блокировок HTTP-запросов от скраперов;

3. Статичность контента — полная доступность HTML-кода страницы при первичном запросе без динамической подгрузки.

Идеальный случай — соответствие всем трём пунктам. При этом:

1. Ограничения по пунктам 2 и 3 в большинстве случаев можно обойти стандартными методами;
2. Нарушение пункта 1 создаёт принципиальные сложности: обработка разноформатных данных может потребовать ручной настройки для каждого документа.

В качестве источника выбран новостной сайт НИУ ВШЭ. Этот ресурс:

1. Имеет единую структуру новостных материалов;
2. Не блокирует автоматизированные запросы;
3. Предоставляет полный HTML-код страницы без динамической генерации контента.

Указанные характеристики делают сайт ВШЭ оптимальным вариантом для реализации поставленных задач.

1.3 Подготовка собранных данных

Полученные данные требуют предварительной обработки для устранения шума и повышения качества анализа. Основные этапы предобработки включают [2]:

1. Очистка от технического шума:
 - Удаление лишних пробелов и переносов строк;
 - Очистка от специальных символов (скобки, HTML-теги, эмодзи);
 - Нормализация регистра (приведение текста к нижнему регистру).
2. Токенизация: разделение текста на семантические единицы (слова, предложения);
3. Лемматизация: приведение словоформ к лемме (словарной форме);
4. Удаление стоп-слов: исключение частотных слов с низкой смысловой нагрузкой (предлоги, союзы, частицы);

Обоснование выбора лемматизации: В отличие от стемминга (например, алгоритм Snowball), который применяет шаблонное усечение окончаний, лемматизация обеспечивает точное приведение слов к нормальной форме с сохранением семантики [2]. Это критически важно для тематического моделирования, где искажение смысла слов может привести к некорректной интерпретации контекста. На рис. 1 показаны принципиальные различия между двумя

подходами.

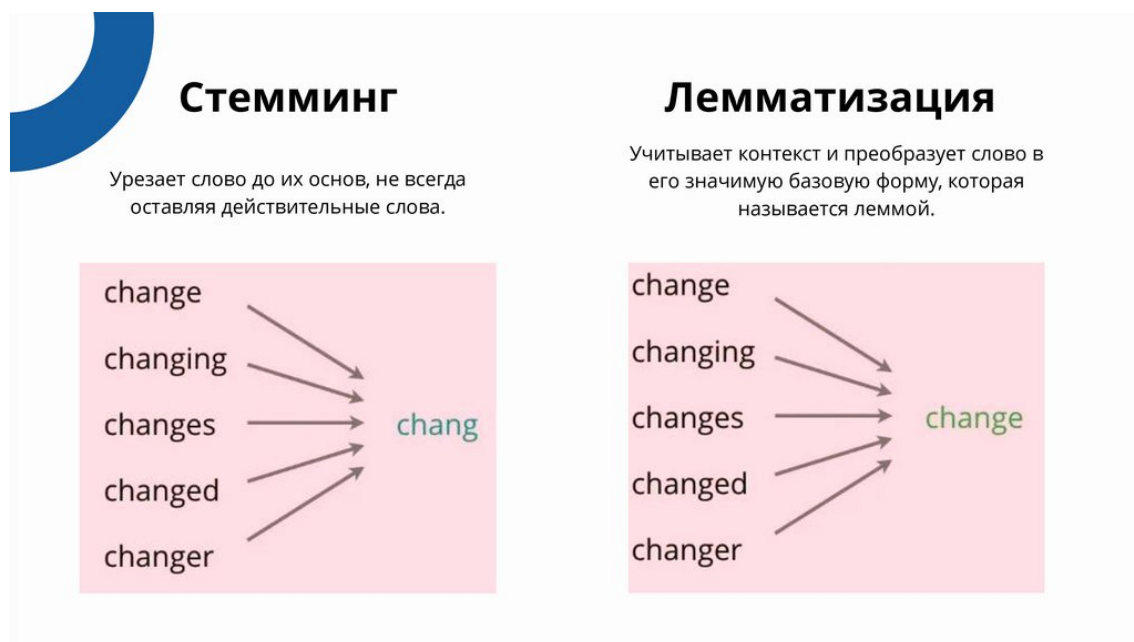


Рисунок 1 – Иллюстрация разницы между стеммингом и лемматизацией

1.4 Математические основы тематического моделирования

1.4.1 Основная гипотеза тематического моделирования

Тематическое моделирование — это метод анализа текстовых данных, который позволяет выявить семантические структуры в коллекциях документов.

Основная идея тематического моделирования [3] заключается в том, что слова в тексте связаны не с конкретным документом, а с темами. Сначала текст разбивается на темы, и каждая из них генерирует слова для соответствующих позиций в документе. Таким образом, сначала формируется тема, а затем тема определяет термины.

Эта гипотеза позволяет проводить тематическую классификацию текстов на основе частоты и совместной встречаемости слов. Искуственную классификацию текстов на основе частоты и взаимовстречаемости слов.

1.4.2 Аксиоматика тематического моделирования

Каждый текст можно количественно охарактеризовать. Ниже приведены основные количественные характеристики, используемые при тематическом моделировании [4]:

- W — конечное множество термов;
- D — конечное множество текстовых документов;
- T — конечное множество тем;
- $D \times W \times T$ — дискретное вероятностное пространство;
- коллекция — i.i.d выборка $(d_i, w_i, t_i)_{i=1}^n$;
- $n_{dwt} = \sum_{i=1}^n [d_i = d][w_i = w][t_i = t]$ — частота (d, w, t) в коллекции;
- $n_{wt} = \sum_d n_{dwt}$ — частота термина w в документе d ;
- $n_{td} = \sum_w n_{dwt}$ — частота термов темы t в документе d ;
- $n_t = \sum_{d,w} n_{dwt}$ — частота термов темы t в коллекции;
- $n_{dw} = \sum_t n_{dwt}$ — частота термина w в документе d ;
- $n_W = \sum_d n_{dw}$ — частота термина w в коллекции;
- $n_d = \sum_w n_{dw}$ — длина документа d ;
- $n = \sum_{d,w} n_{dw}$ — длина коллекции.

Также в тематическом моделировании используются следующие гипотезы и аксиомы [3]:

- независимость слов от порядка в документе: порядок слов в документе не важен;

- независимость от порядка документов в коллекции: порядок документов в коллекции не важен;
- зависимость термина от темы: каждый терм связан с соответствующей темой и порождается ей;
- гипотеза условной независимости: $p(w|d, t) = p(w|t)$.

1.4.3 Задача тематического моделирования

Как уже говорилось ранее, документ порождается следующим образом [3]:

1. для каждой позиции в документе генерируется тема $p(t|d)$;
2. для каждой сгенерированной темы в соответствующей позиции генерируется терм $p(w|d, t)$.

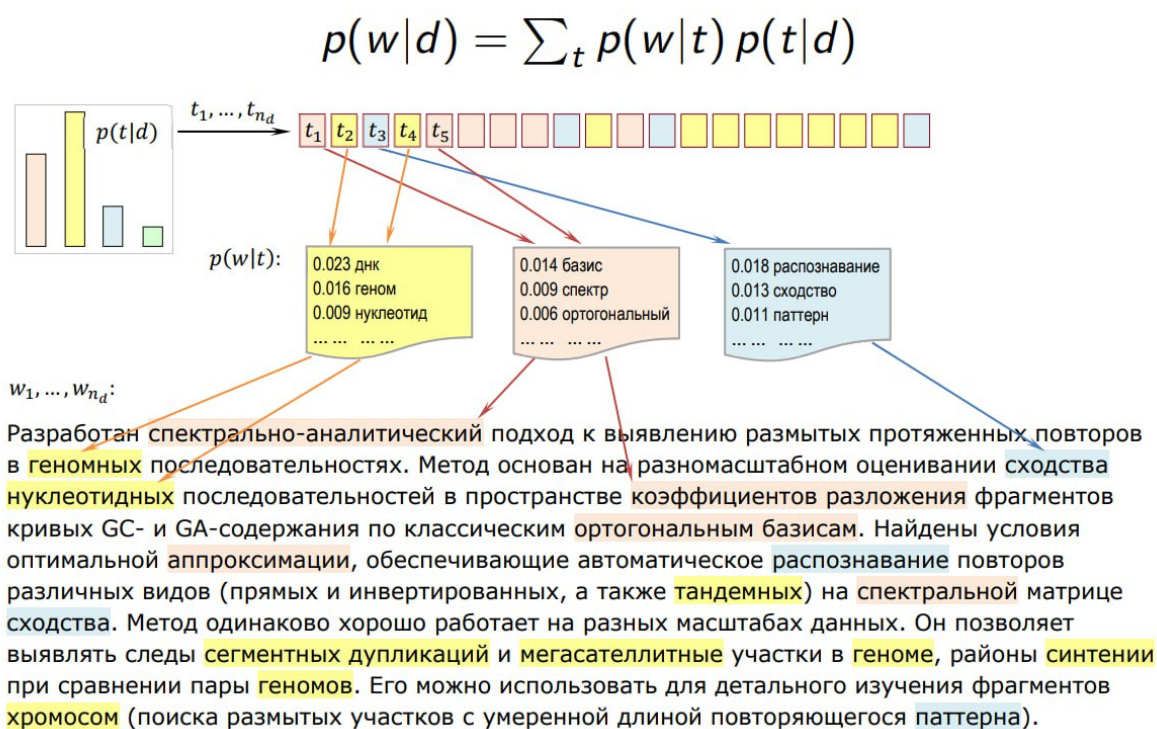


Рисунок 2 – Алгоритм формирования документа

Тогда вероятность появления слова в документе можно описать по формуле полной вероятности [3, 5]:

$$p(w|d) = \sum_{t \in T} p(w|d, t) p(t|d) = \sum_{t \in T} p(w|t) p(t|d) \quad (1)$$

Такой алгоритм является прямой задачей порождения текста. Тематическое моделирование призвано решить обратную задачу:

1. для каждого термина w в тексте найти вероятность появления в теме t (найти $p(w|t) = \phi_{wt}$);
2. для каждой темы t найти вероятность появления в документе d (найти $p(t|d) = \theta_{td}$).

Обратную задачу можно представить в виде стохастического матричного разложения [3](#).

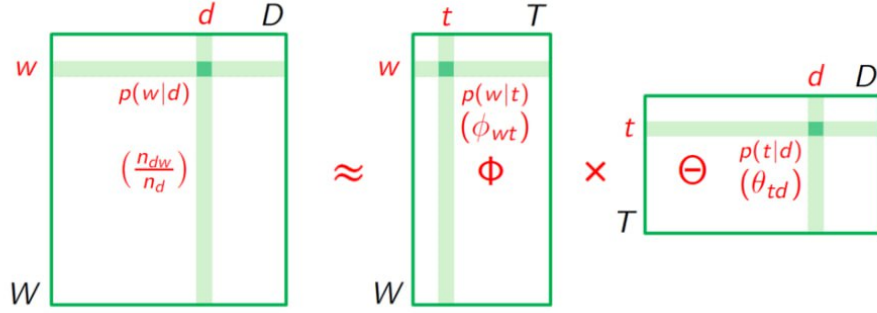


Рисунок 3 – Стохастическое матричное разложение

Таким образом, тематическое моделирование ищет величину $p(w|d)$.

1.4.4 Решение обратной задачи

Для решения задачи тематического моделирования необходимо найти величину $p(w|d)$, сделать это можно с помощью метода максимального правдоподобия.

Лемма о максимизации функции на единичных симплексах: Перед тем как перейти к решению обратной задачи, сформулируем лемму, которая поможет в этом процессе [\[3\]](#).

Введём операцию нормировки вектора:

$$p_i = \underset{i \in I}{\text{norm}}(x_i) = \frac{\max\{x_i, 0\}}{\sum_{k \in I} \max\{x_k, 0\}} \quad (2)$$

Лемма о максимизации функции на единичных симплексах [\[3, 6\]](#):

Пусть функция $f(\Omega)$ непрерывно дифференцируема по набору векторов $\Omega = (w_i)_{i \in J}$, $w_j = (w_{ij})_{i \in I_j}$ различных размерностей $|I_j|$. Тогда векторы w_j

локального экстремума задачи

$$\begin{cases} f(\Omega) \rightarrow \max_{\Omega} \\ \sum_{i \in I_j} w_{ij} = 1, \quad j \in J \\ w_{ij} \geq 0, \quad i \in I_j, j \in J \end{cases}$$

при условии 1^0 : $(\exists i \in I_j) w_{ij} \frac{\partial f}{\partial w_{ij}} > 0$ удовлетворяют уравнениям

$$w_{ij} = \underset{i \in I_j}{\text{norm}} \left(w_{ij} \frac{\partial f}{\partial w_{ij}} \right), \quad i \in I_j; \quad (3)$$

при условии 2^0 : $(\forall i \in I_j) w_{ij} \frac{\partial f}{\partial w_{ij}} \leq 0$ и $(\exists i \in I_j) w_{ij} \frac{\partial f}{\partial w_{ij}} < 0$ удовлетворяют уравнениям

$$w_{ij} = \underset{i \in I_j}{\text{norm}} \left(-w_{ij} \frac{\partial f}{\partial w_{ij}} \right), \quad i \in I_j; \quad (4)$$

в противном случае (условие 3^0) — однородным уравнениям

$$w_{ij} \frac{\partial f}{\partial w_{ij}} = 0, \quad i \in I_j. \quad (5)$$

Данная лемма служит для оптимизации любых моделей, параметрами которых являются неотрицательные нормированные векторы.

Сведение обратной задачи к максимизации функционала: Чтобы вычислить величину $p(w|d)$ воспользуемся принципом максимума правдоподобия [5], согласно которому будут подобраны параметры Φ, Θ такие, что $p(w|d)$ примет наибольшее значение.

$$\prod_{i=1}^n p(d_i, w_i) = \prod_{d \in D} \prod_{w \in d} p(d, w)^{n_{dw}} \quad (6)$$

Прологарифмировав правдоподобие, перейдём к задаче максимизации логарифма правдоподобия.

$$\sum_{d \in D} \sum_{w \in d} n_{dw} \ln p(w|d) \xrightarrow[\Phi, \Theta]{\text{const}} \max \quad (7)$$

Данная задача эквивалентна задаче максимизации функционала

$$L(\Phi, \Theta) = \sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt} \theta_{td} \rightarrow \max_{\Phi, \Theta} \quad (8)$$

при ограничениях неотрицательности и нормировки

$$\phi_{wt} \geq 0; \quad \sum_{w \in W} \phi_{wt} = 1; \quad \theta_{td} \geq 0; \quad \sum_{t \in T} \theta_{td} = 1 \quad (9)$$

Таким образом, обратная задача сводится к задаче максимизации функционала [4].

Аддитивная регуляризация тематических моделей: Задача 8 не соответствует критериям корректно поставленной задачи по Адамару [7], поскольку в общем случае она имеет бесконечное множество решений. Это свидетельствует о необходимости доопределения задачи.

Для доопределения некорректно поставленных задач применяется регуляризация [7]: к основному критерию добавляется дополнительный критерий — регуляризатор, который соответствует специфике решаемой задачи.

Метод ARTM (аддитивная регуляризация тематических моделей [3]) основывается на максимизации линейной комбинации логарифма правдоподобия и регуляризаторов $R_i(\Phi, \Theta)$ с неотрицательными коэффициентами регуляризации τ_i , $i = 1, \dots, k$.

Преобразуем задачу к ARTM виду:

$$\sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt} \theta_{td} + R(\Phi, \Theta) \rightarrow \max_{\Phi, \Theta}; \quad R(\Phi, \Theta) = \sum_{i=1}^k \tau_i R_i(\Phi, \Theta) \quad (10)$$

при ограничениях неотрицательности и нормировки 9.

Регуляризатор (или набор регуляризаторов) выбирается в соответствии с решаемой задачей.

Е-М алгоритм: Из представленных выше ограничений 9 следует, что столбцы матриц можно считать неотрицательными единичными векторами. Таким образом, задача сводится к максимизации функции на единичных симплексах [3].

Воспользуемся леммой о максимизации функции на единичных симплексах 1.4.4 и перепишем задачу.

Пусть функция $R(\Phi, \Theta)$ непрерывно дифференцируема. Тогда точка (Φ, Θ) локального экстремума задачи с ограничениями, удовлетворяет системе уравнений с вспомогательными переменными $p_{tdw} = p(t|d, w)$, если из решения исключить нулевые столбцы матриц Φ и Θ :

$$\begin{cases} p_{tdw} = \underset{t \in T}{\text{norm}}(\phi_{wt}\theta_{td}) \\ \phi_{wt} = \underset{w \in W}{\text{norm}}\left(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}}\right); \\ \theta_{td} = \underset{t \in T}{\text{norm}}\left(n_{td} + \theta_{td} \frac{\partial R}{\partial \theta_{td}}\right) \end{cases} \quad (11)$$

Полученная модель соответствует Е-М алгоритму, где первая строка системы уравнений соответствует Е-шагу, а вторая и третья строки — М-шагу.

Решив полученную систему уравнений, методом простых итерации получим искомые матрицы Φ и Θ .

1.4.5 Регуляризаторы в тематическом моделировании

В этом разделе будут рассмотрены некоторые возможные варианты регуляризаторов.

Дивергенция Кульбака-Лейблера: Перед тем как перейти к регуляризаторам необходимо ввести меру оценки близости тем.

Чтобы оценить близость тем можно воспользоваться дивергенцией Кульбака-Лейблера [3, 7] (KL или KL-дивергенция). KL-дивергенция позволяет оценить степень вложенности одного распределения в другое, в случае тематического моделирования будет оцениваться вложенность матриц.

Определим KL-дивергенцию:

Пусть $P = (p_i)_{i=1}^n$ и $Q = (q_i)_{i=1}^n$ некоторые распределения. Тогда дивергенция Кульбака-Лейблера имеет следующий вид:

$$KL(P||Q) = KL_i(p_i||q_i) = \sum_{i=1}^n p_i \ln \frac{p_i}{q_i}. \quad (12)$$

Свойства KL-дивергенции:

1. $KL(P||Q) \geq 0$;

2. $KL(P||Q) = 0 \Leftrightarrow P = Q$;

3. Минимизация KL эквивалентна максимизации правдоподобия:

$$KL(P||Q(\alpha)) = \sum_{i=1}^n p_i \ln \frac{p_i}{q_i(\alpha)} \rightarrow \min_{\alpha} \Leftrightarrow \sum_{i=1}^n p_i \ln q_i(\alpha) \rightarrow \max_{\alpha};$$

4. Если $KL(P||Q) < KL(Q||P)$, то P сильнее вложено в Q , чем Q в P .

Теперь можно перейти к рассмотрению регуляризаторов.

Регуляризатор сглаживания: Сглаживание предполагает семантическое сближение тем, это может быть полезно в следующих случаях [8]:

1. Темы могут быть похожи между собой по терминологии, например, основы теории вероятностей и линейной алгебры обладают рядом одинаковых терминов;
2. При выделении фоновых тем важно максимально вобрать в них слова, следовательно, сглаживание поможет решить эту задачу.

Определим регуляризатор сглаживания:

Пусть распределения ϕ_{wt} близки к заданному распределению β_w и пусть распределения θ_{td} близки к заданному распределению α_t . Тогда в форме KL-дивергенции 1.4.5 выразим задачу сглаживания:

$$\sum_{t \in T} KL(\beta_w || \phi_{wt}) \rightarrow \min_{\Phi}; \quad \sum_{d \in D} KL(\alpha_t || \theta_{td}) \rightarrow \min_{\Theta}. \quad (13)$$

Согласно свойству 3 KL-дивергенции перейдём к задаче максимизации правдоподобия:

$$R(\Phi, \Theta) = \beta_o \sum_{t \in T} \sum_{w \in W} \beta_w \ln \phi_{wt} + \alpha_o \sum_{d \in D} \sum_{t \in T} \alpha_t \ln \theta_{td} \rightarrow \max. \quad (14)$$

Перепишем EM-алгоритм 11 в соответствии с полученной формулой:

$$\begin{cases} p_{tdw} = \underset{t \in T}{\text{norm}}(\phi_{wt} \theta_{td}) \\ \phi_{wt} = \underset{w \in W}{\text{norm}}(n_{wt} + \beta_o \beta_w); \\ \theta_{td} = \underset{t \in T}{\text{norm}}(n_{td} + \alpha_o \alpha_t) \end{cases} \quad (15)$$

Таким образом был получен модифицированный ЕМ-алгоритм соответствующий модели LDA [3, 7].

Регуляризатор разреживания: Разреживание подразумевает разделение тем и документов, исключая общие слова из них. Этот тип регуляризации основывается на предположении, что темы и документы в основном являются специфичными и описываются относительно небольшим набором терминов, которые не встречаются в других темах [3, 8].

Определим регуляризатор разреживания:

Пусть распределения ϕ_{wt} далеки от заданного распределения β_w и пусть распределения θ_{td} далеки от заданного распределения α_t . Тогда в форме KL-дивергенции 1.4.5 выразим задачу разреживания:

$$\sum_{t \in T} KL(\beta_w || \phi_{wt}) \rightarrow \max_{\Phi}; \quad \sum_{d \in D} KL(\alpha_t || \theta_{td}) \rightarrow \max_{\Theta}. \quad (16)$$

Согласно свойству 3 KL-дивергенции перейдём к задаче максимизации правдоподобия:

$$R(\Phi, \Theta) = -\beta_o \sum_{t \in T} \sum_{w \in W} \beta_w \ln \phi_{wt} - \alpha_o \sum_{d \in D} \sum_{t \in T} \alpha_t \ln \theta_{td} \rightarrow \max. \quad (17)$$

Перепишем ЕМ-алгоритм 11 в соответствии с полученной формулой:

$$\begin{cases} p_{tdw} = \underset{t \in T}{\text{norm}}(\phi_{wt} \theta_{td}) \\ \phi_{wt} = \underset{w \in W}{\text{norm}}(n_{wt} - \beta_o \beta_w); \\ \theta_{td} = \underset{t \in T}{\text{norm}}(n_{td} - \alpha_o \alpha_t) \end{cases} \quad (18)$$

Таким образом был получен модифицированный ЕМ-алгоритм, разреживающий матрицы Φ и Θ [3, 8].

Регуляризатор декоррелирования тем: Декоррелятор тем — это частный случай разреживания, призванный выделить для каждой темы лексическое ядро — набор термов, отличающий её от других тем [3, 8].

Определим регуляризатор декоррелирования:

Минимизируем ковариации между вектор-столбцами ϕ_t :

$$R(\Phi) = -\frac{\tau}{2} \sum_{t \in T} \sum_{s \in T \setminus t} \sum_{w \in W} \phi_{wt} \phi_{ws} \rightarrow \max. \quad (19)$$

Перепишем ЕМ-алгоритм 11 в соответствии с полученной формулой:

$$\begin{cases} p_{tdw} = \underset{t \in T}{\text{norm}}(\phi_{wt} \theta_{td}) \\ \phi_{wt} = \underset{w \in W}{\text{norm}} \left(n_{wt} - \tau \phi_{wt} \sum_{s \in T \setminus t} \phi_{ws} \right); \\ \theta_{td} = \underset{t \in T}{\text{norm}} \left(n_{td} + \theta_{td} \frac{\partial R}{\partial \theta_{td}} \right) \end{cases} \quad (20)$$

Таким образом был получен модифицированный ЕМ-алгоритм, декоррелирующий темы [3, 8].

1.4.6 Оценка качества моделей

После построения модели, очевидно, нужно оценить её качество.

Перечислим основные критерии оценки качества тематических моделей [3]:

1. Внешние критерии (оценка производится экспертами):
 - а) полнота и точность тематического поиска;
 - б) качество ранжирования при тематическом поиске;
 - в) качество классификации / категоризации документов;
 - г) качество суммаризации / сегментации документов;
 - д) экспертные оценки качества тем.
2. Внутренние критерии (оценка производится программно):
 - а) правдоподобие и перплексия;
 - б) средняя когерентность (согласованность тем);
 - в) разреженность матриц Φ и Θ ;
 - г) различность тем;
 - д) статический тест условной независимости.

Поскольку оценка по внешним критериям невозможна в рамках данной работы, сосредоточимся на внутренних критериях оценки, которые можно вычислять автоматически.

Правдоподобие и перплексия: Перплексия основывается на логарифме правдоподобия и является его некоторой модификацией [3].

$$P(D) = \exp \left(-\frac{1}{n} \sum_{d \in D} \sum_{w \in d} n_{dw} \ln p(w|d) \right), \quad n = \sum_{d \in D} \sum_{w \in d} n_{dw} \quad (21)$$

Не трудно заметить, что при равномерном распределении слов в тексте выполняется равенство $p(w|d) = \frac{1}{|W|}$. В этом случае значение перплексии равно мощности словаря $P = |W|$. Это позволяет сделать вывод, что перплексия является мерой разнообразия и неопределенности слов в тексте: чем меньше значение перплексии, тем более разнообразны вероятности появления слов.

Таким образом, чем меньше перплексия, тем больше слов с большей вероятностью $p(w|d)$, которые модель умеет лучше предсказывать, следовательно, чем меньше перплексия, тем лучше.

Когерентность: Когерентность является мерой, коррелирующей с экспертной оценкой интерпретируемости тем [3].

Когерентность (согласованность) темы t по k топовым словам:

$$PMI_t = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^k PMI(w_i, w_j), \quad (22)$$

где w_i — i -ое слово в порядке убывания ϕ_{wt} ; $PMI(u, v) = \ln \frac{|D|N_{uv}}{N_u N_v}$ — пото-
чечная взаимная информация; N_{uv} — число документов, в которых слова u, v хотя бы один раз встречаются рядом (расстояние определяется отдельно); N_u — число документов, в которых u встретился хотя бы один раз.

Гипотезу когерентности можно выразить так: когда человек говорит о какой-либо теме, то часто употребляет достаточно ограниченный набор слов, относящийся к этой теме, следовательно, чем чаще будут встречаться вместе слова этой темы, тем лучше её можно будет интерпретировать.

Сама когерентность берёт самые часто встречающиеся слова из тем, и вычисляет для каждой пары из них насколько они часто встречаются, соответственно, чем выше будет значение взаимовстречаемости, тем лучше.

Разреженность: Разреженность — доля нулевых элементов в матрицах Φ и Θ .

Разреженность играет ключевую роль в выявлении различий между темами [3]. Каждая тема формируется на основе ограниченного набора слов, в то время как остальные слова должны встречаться реже, что отражается в нулевых элементах матриц. Оптимальный уровень разреженности должен быть высоким, но не чрезмерным: в таком случае темы будут четко различимы. Если разреженность слишком низка, темы могут сливаться, а если слишком высока — содержать недостаточное количество слов для адекватного представления.

Чистота темы: Чистота темы:

$$\sum_{w \in W_t} p(w|t), \quad (23)$$

где W_t — ядро темы: $W_t = \{w : p(w|t) > \alpha\}$, где α подбирается по разному, например $\alpha = 0.25$ или $\alpha = \frac{1}{|W|}$.

Данная характеристика показывает как вероятно относится ядро темы к фоновым словам темы, следовательно, чем больше вероятность ядра, тем лучше [8].

Контрастность темы: Контрастность темы:

$$\frac{1}{|W_t|} \sum_{w \in W_t} p(t|w). \quad (24)$$

Данная характеристика показывает насколько часто слова из ядра темы встречаются в других темах, очевидно, что чем меньше ядро будет встречаться в других темах, тем лучше.

1.5 Методические основы работы с текстом с помощью нейросетей

1.5.1 Проблема представления текста

Нейронные сети умеют работать только с числами, поэтому встаёт вопрос о том, как наилучшим образом переносить текст в пространство чисел. Такой способ переноса должен быть не только быстрым, точным и способным вмещать в себя тысячи слов, но ещё и учитывать, что естественный язык имеет временную зависимость: слова в предложении складываются последовательно и зависят друг от друга, а не существуют в вакууме, что дополнительно усложняет задачу.

Тогда формализуем качества, которыми должен обладать способ представления текста в виде чисел:

- Выразительность:
 1. Способность различать тысячи слов;
 2. Способность учитывать контекст (временную зависимость между словами).
- Скорость: эффективно работать с высокоразмерными данными на современном оборудовании;
- Эффективность: иметь компактное представление и адаптироваться к новым словам.

Теперь кратко рассмотрим некоторые из методов представления текста в виде чисел:

Мешок слов (Bag-of-Words): Одним из самых простых способов численного представления текста является мешок слов [9].

Данный метод работает следующим образом [10]:

1. Создаётся словарь с уникальными индексами;
2. Каждое слово кодируется one-hot вектором:

$$v_i = [a_1, \dots, a_N], \quad a_j = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases} \quad (25)$$

где N — размер словаря.

3. Предложение представляется суммой векторов слов:

$$s = [f_1, \dots, f_N], \quad f_j = \text{частота слова } j \text{ в предложении.} \quad (26)$$

Данный метод, несмотря на свою простоту, не может быть выбран из-за ряда существенных недостатков [9, 10]:

1. Высокая размерность и разреженность данных;
2. Игнорирование порядка слов;
3. Отсутствие учёта семантики (все слова ортогональны);
4. Сложность адаптации к новым словам (требуется пересчёт словаря).

TF-IDF взвешивание: Улучшение BoW: элементы вектора предложения умножаются на TF-IDF веса слов, это частично решает проблему семантической значимости, но сохраняет другие недостатки BoW [9, 10].

Эмбединги слов: Семантические векторные представления слов [9, 10]:

- Каждому слову сопоставляется плотный вектор фиксированной размерности (обычно 50-300);
- Векторы обучаются так, чтобы семантически близкие слова имели схожие эмбединги;
- Матрица эмбедингов — обучаемый параметр нейросети.

Данный способ максимально полно соответствует описанным ранее критериям, обладая благодаря своей природе следующими преимуществами [9, 10]:

1. Низкая размерность;
2. Учёт семантики;
3. Возможность учёта контекста;
4. Гибкость: новые слова можно добавлять через дообучение.

1.5.2 Выбор архитектуры нейронной сети

Так как представление текста в виде эмбедингов удовлетворяет критериям то, будем рассматривать архитектуры, разработанные для работы с ними: рекуррентные нейронные сети (RNN) и трансформеры.

Рекуррентные нейронные сети (RNN) Данные сети обрабатывают последовательность слов рекуррентно, шаг за шагом обновляя своё состояние на основе текущего слова и предыдущих значений [10, 11]. Это позволяет учитывать [10, 11]:

- Порядок слов;
- Контекст (благодаря механизмам памяти в LSTM/GRU).

Недостатки [10, 11]:

1. Низкая скорость: вычисления последовательны, невозможна параллелизация;
2. Проблемы с длинными последовательностями:
 - а) Забывание раннего контекста;
 - б) Затухание/взрыв градиентов при обучении.

Преимущества [10, 11]:

1. Менее требовательны к вычислительным ресурсам;
2. Эффективны на малых объёмах данных.

Трансформеры Обрабатывают всю последовательность слов одновременно благодаря механизму внимания (attention) [10, 12].

Ключевые особенности [10, 12]:

- Параллельные вычисления, а следовательно и высокая скорость;
- Учёт контекста через self-attention;
- Позиционные энкодинги позволяют учитывать порядок слов.

Недостатки [10, 12]:

1. Высокие требования к вычислительным ресурсам;
2. Требуют больших объёмов данных для обучения.

Преимущества [10, 12]:

1. Эффективны для длинных текстов;
2. Имеют лучшее качество на сложных задачах.

Определение с типом В рамках данной работы рассматривается тематическая классификация текстов, то есть предполагается, что по длинной входящей последовательности принимается решение о её принадлежности к той или иной теме.

Тогда для данной задачи критичны:

1. Обработка длинных последовательностей;
2. Скорость предсказания;
3. Использование современных вычислительных ресурсов.

Таким образом, для решения поставленной задачи больше подходят сети-трансформеры, так как:

- Проблемы с ресурсами решаются облачными сервисами;
- Доступны предобученные модели (BERT, GPT);
- Механизм внимания лучше улавливает тематические связи.

1.5.3 Оценка качества работы нейронных сетей

Для оценки качества классификации нейронными сетями используются несколько базовых метрик [13].

Прежде чем перейти к рассмотрению метрик, введем основные обозначения [13]:

- TP (true positive) — объект верно отнесён к целевому классу;
- TN (true negative) — объект верно не отнесён к целевому классу;
- FP (false positive) — объект ошибочно отнесён к целевому классу;
- FN (false negative) — объект ошибочно не отнесён к целевому классу.

Accuracy Accuracy вычисляется по формуле [13]:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \quad (27)$$

Эта метрика показывает общую долю верных классификаций. Несмотря на простоту интерпретации, accuracy часто оказывается недостаточно информативной при работе с несбалансированными данными [13].

Precision Precision (точность предсказания) вычисляется как [13]:

$$Precision = \frac{TP}{TP + FP} \quad (28)$$

Метрика отражает долю верно классифицированных объектов среди всех примеров, отнесённых классификатором к целевому классу [13].

Recall Recall (полнота) определяется формулой [13]:

$$Recall = \frac{TP}{TP + FN} \quad (29)$$

Метрика показывает долю верно распознанных объектов целевого класса относительно их общего количества [13].

F1-мера F1-мера вычисляется по формуле [13]:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (30)$$

Эта метрика представляет собой гармоническое среднее precision и recall. Она полезна при необходимости балансировки двух показателей [13].

Confusion matrix Матрица ошибок (confusion matrix) наглядно визуализирует распределение ошибок классификации по классам. Хотя её использование для

сравнения моделей может быть затруднительно из-за большого размера, она эффективна для демонстрации качества итоговой модели [13].

Пример матрицы ошибок представлен на рис. 4:

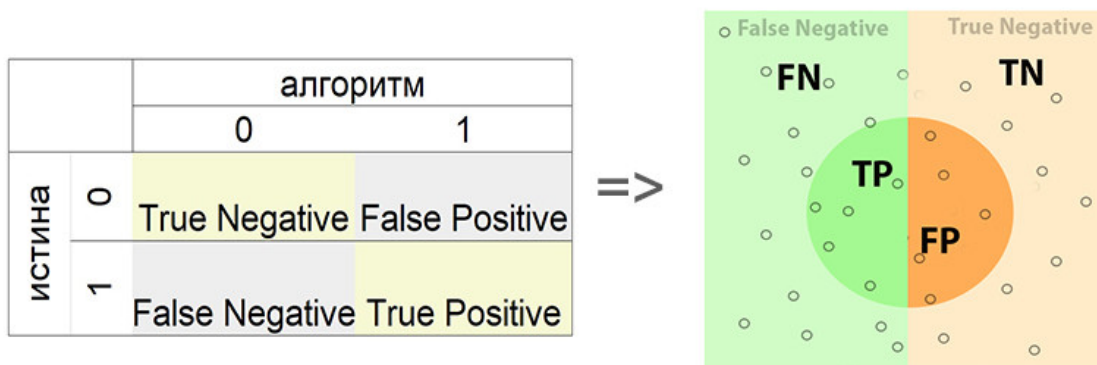


Рисунок 4 – Пример матрицы ошибок

2 Практико-технологические основы автоматической тематической классификации

2.1 Получение новостного массива путём веб-скраппинга

2.1.1 Выбор инструментов получения новостных данных

Для веб-скраппинга доступны библиотеки на разных языках, однако выбор логично сделать в пользу Python — наиболее популярного языка для обработки данных и работы с глубоким обучением. Среди Python-библиотек ключевыми являются [14]:

- requests — для отправки HTTP-запросов;
- BeautifulSoup4 — для парсинга HTML-кода в удобную объектную структуру;
- selenium — для работы с динамическими сайтами, где контент генерируется JavaScript.

Первые две библиотеки эффективны для статических страниц: requests получает исходный код, а BeautifulSoup4 извлекает данные через поиск по тегам. Selenium же имитирует взаимодействие реального браузера, что позволяет обрабатывать страницы с отложенной загрузкой контента [14].

Этот набор инструментов покрывает потребности работы с подавляющим большинством сайтов — от простых статических ресурсов до сложных веб-приложений [14].

2.1.2 Реализация алгоритма сбора новостных данных

Для такого простого и имеющего хорошую структуру новостного сайта ВШЭ не потребуется библиотеки Selenium, достаточно только BeautifulSoup4 и requests.

Алгоритм сбора данных включает следующие этапы [14]:

1. Анализ структуры сайта:

- Многостраничный ресурс с 10 новостными карточками на каждой странице;
- Карточка новости содержит: ссылку, дату публикации, заголовок, краткое содержание;
- Полный текст доступен по отдельной ссылке внутри карточки.

Пример страницы сайта можно увидеть в соответствующем приложении А.

2. Реализация базовых функций (листинг 1) [14, 15]:

- Получение HTML-кода страницы через `requests.get()`;
- Сохранение сырых данных для последующей обработки.

```

1 def __getPage__(url: str, file_name: str) -> None:
2     r = requests.get(url=url)
3     with open(file_name, "w", encoding="utf-8") as file:
4         file.write(r.text)

```

Листинг 1: Функция получения HTML-кода страницы

3. Извлечение метаданных (листинг 2) [14, 16]:

- Парсинг сохранённого HTML через BeautifulSoup4;
- Поиск элементов по тегам и CSS-классам (`find()`, `find_all()`);
- Извлечение текстового содержимого (`text`, `get()`).

```

1 with open(page_file_name, encoding="utf-8") as file:
2     src = file.read()
3     soup = BeautifulSoup(src, "lxml")
4     news = soup.find("div", class_="post")
5     try:
6         link = news.find("h2",
7                           class_="first_child").find("a").get("href")
8         if not link.startswith("https://"):
9             link = 'https://www.hse.ru' + link
10    except:
11        link = ""
12    try:
13        news_short_content = news.find("p",
14                                        class_="first_child").find_next_sibling("p").text.strip()
15    except:
16        news_short_content = ""

```

Листинг 2: Извлечение ссылок и кратких описаний

4. Получение полного текста новости (листинг 3) [14, 16]:

- Рекурсивное использование `get_page()` для целевых URL;
- Анализ структуры контентной страницы.

```

1 def __parse_news__(url: str) -> str:
2     news_file_name = "news.html"
3     __getPage__(url, news_file_name)
4     with open(news_file_name, encoding="utf-8") as file:
5         src = file.read()
6         content = BeautifulSoup(src, "lxml").find("div",

```

```

        class_="main").find(
7         "div", class_="post__text"
8     ).text.strip()
9     return content

```

Листинг 3: Функция извлечения полного текста новости

5. Обработка страницы целиком (листинг 4) [14, 16]:

- Итерация по 10 элементам div.post на странице;
- Использование find_next_sibling() для навигации;
- Сохранение результатов в pandas DataFrame для анализа.

```

1 def __parse_page__(page_file_name: str, news_container:
    pd.DataFrame) -> None:
2     for i in range(10):
3         try:
4             if link.startswith("https://www.hse.ru/news/"):
5                 news_content = __parse_news__(link)
6         except:
7             news_content = ""
8         if len(
9             news_day + news_month + news_year + news_name +
10             news_short_content +
11             news_content
12         ) > 0:
13             news_container.loc[len(news_container.index)] =
                [
14                 link, news_date, news_name,
                    news_short_content, news_content]
15     news = news.find_next_sibling("div", class_="post")

```

Листинг 4: Обработка новостной страницы

6. Масштабирование на все страницы (листинг 5):

- Динамическое формирование URL через модификацию параметров;
- Пакетная обработка через цикл с изменяемым индексом страницы.

```

1 def __crawling_pages__(start: int, end: int,
    news_container: pd.DataFrame, num_of_thread: int) ->
    pd.DataFrame:
2     page_file_name = "page.html"
3     for i in range(start, end + 1):
4         try:

```

```

5         __getPage__("https://www.hse.ru/news/page{0}.html".format
                        page_file_name)
6         __parse_page__(page_file_name, news_container)
7     except:
8         continue

```

Листинг 5: Функция обработки всего архива новостей

7. Оптимизация производительности (листинг 6) [17, 18]:

- Реализация многопоточности через стандартные средства Python;
- Создание изолированных DataFrame для каждого потока;
- Агрегация результатов после завершения параллельных задач.

```

1 def crawling_pages(off_pc: bool, pages: int) -> None:
2     columns = ["url", "date", "title", "summary", "content"]
3     news_container1 = pd.DataFrame(columns=columns)
4     news_container2 = pd.DataFrame(columns=columns)
5     thread1 = threading.Thread(target=__crawling_pages__,
6                                args=(0, pages // 2, news_container1, 1))
7     thread2 = threading.Thread(target=__crawling_pages__,
8                                args=(pages // 2, pages, news_container2, 2))
9     thread1.start()
10    thread2.start()
11    thread1.join()
12    thread2.join()
13    try:
14        news = pd.concat([news_container1,
15                           news_container2], ignore_index=True)
16        news.to_excel("./news.xlsx")
17    except:
18        print("Не получилось!")

```

Листинг 6: Многопоточная реализация парсера

Полная реализация веб-скрапера доступна в приложении Б.

2.2 Подготовка новостного массива

2.2.1 Выбор инструментов для подготовки данных

Чтобы не повышать количество используемых языков, будем рассматривать только инструменты, доступные на Python. Среди них выделяются: NLTK, Rymorphy3, SpaCy и Gensim [19].

Сделаем выбор между связкой NLTK + Rymorphy3 и SpaCy. Обе группы

библиотек позволяют проводить лемматизацию и удаление стоп-слов, но реализуют это по-разному. NLTK и Rymorphy3 приводят слова к начальной форме без учёта контекста, тогда как SpaCy — нейросетевой инструмент, анализирующий окружение терминов [19, 20]. Определение стоп-слов в обоих случаях происходит по заранее заданным словарям, поэтому разницы здесь нет [19, 20]. Однако SpaCy обеспечивает не только более точную лемматизацию, но и лаконичный интерфейс, что упрощает её использование [19, 20].

Как упоминалось ранее библиотека SpaCy определяет стоп-слова только по предопределённому списку, который не является исчерпывающим. Это связано с тем, что набор стоп-слов зависит от тематики текста, и универсального решения не существует. Для дополнительной фильтрации применим метрику TF-IDF, которая оценивает значимость слов. Формула расчёта [21]:

$$tfidf(w, d) = \frac{n_{wd}}{n_d} \cdot \log \left(\frac{|D|}{|\{d \in D : w \in d\}|} \right), \quad (31)$$

где:

- w — термин;
- d — документ;
- n_{wd} — частота встречаемости w в d ;
- n_d — число терминов в d ;
- $|D|$ — число документов в коллекции;
- $|\{d \in D : w \in d\}|$ — количество документов, содержащих w .

Данная метрика будет тем выше для термина w в документе d , чем чаще будет встречаться термин w в документе d и реже во всех остальных документах коллекции. Таким образом, данную метрику можно интерпретировать как метрику значимости слова w для документа d [21]. Её расчёт будет производиться с помощью библиотеки Gensim.

Таким образом, для обработки текста выбраны SpaCy (токенизация, лемматизация, базовые стоп-слова) и Gensim (расширенная фильтрация через TF-IDF).

2.2.2 Удаление лишних пробелов и переносов строк

Для корректной токенизации и анализа текстовых данных требуется предварительная очистка от лишних пробелов и переносов строк. Реализацию этой процедуры можно выполнить с помощью встроенных методов обработки строк

в Python.

Алгоритм функции включает три этапа:

1. Копирование значимых символов: Посимвольное добавление содержимого исходной строки в результирующий буфер до обнаружения пробела или переноса строки.
2. Нормализация пробелов: При обнаружении пробела/переноса:
 - Добавление одного пробела в буфер
 - Пропуск всех последующих пробелов/переносов до первого непробельного символа
3. Циклическая обработка: Повтор шагов 1-2 до полного прохода исходной строки.

Реализация функции представлена в листинге 7 [17]:

```
1 def __remove_extra_spaces_and_line_breaks__(self, text: str) ->
    str:
2     processed = ""
3     if type(text) != str or len(text) == 0:
4         return ""
5     flag = True
6     for symb in text:
7         if flag and (symb == " " or symb == "\n"):
8             processed += " "
9             flag = False
10        if symb != " " and symb != "\n":
11            flag = True
12        if flag:
13            processed += symb
14    return processed.strip()
```

Листинг 7: Функция нормализации пробелов и переносов строк

2.2.3 Разделение строк на русские и английские фрагменты

Библиотека SpaCy использует предобученные языковые модели, каждая из которых оптимизирована для обработки одного языка (например, отдельно для русского и английского) [22].

Для новостных материалов ВШЭ, содержащих смешанные языковые фрагменты, применение единой модели недопустимо. Решение заключается в предварительном разделении текста на русскоязычные и англоязычные сегменты с последующей обработкой соответствующими моделями.

Алгоритм разделения текста:

1. Инициализация языка:

- Определение языка первого буквенного символа строки
- Установка текущего языкового идентификатора (RU/EN)

2. Построение сегментов:

- Посимвольное накопление символов во временном буфере
- Прерывание потока при обнаружении символа другого языка

3. Сохранение результата:

- Фиксация сегмента в формате (язык, текст)
- Сброс временного буфера

4. Циклическое выполнение: Повтор шагов 2-3 до полной обработки строки с автоматическим переключением языкового идентификатора.

Реализация функции представлена в листинге 8 [17]:

```
1 def __first_is_en__(self, cell: str) -> bool:
2     index_first_en = re.search(r"[a-zA-Z]", cell)
3     index_first_ru = re.search(r"[a-яА-Я]", cell)
4     return True if index_first_en and ( not (index_first_ru) or
5         index_first_en.start() < index_first_ru.start() ) else
6         False
7
6 def __split_into_en_and_ru__(self, cell: str) -> list[(bool,
7     str)]:
8     parts = []
9     is_en = self.__first_is_en__(cell)
10    part = ""
11    for symb in cell:
12        if is_en == (symb in string.ascii_letters) or not
13            (symb.isalpha()):
14            part += symb
15        else:
16            parts.append((is_en, part))
17            part = symb
18            is_en = not (is_en)
19    if part:
20        parts.append((is_en, part))
21    return parts
```

Листинг 8: Функция разделения текста на русско- и англоязычные фрагменты

2.2.4 Очистка от неалфавитных токенов и удаление крайних неалфавитных символов из токенов

В текстах часто встречаются токены, содержащие неалфавитные символы. Кроме того, при токенизации могут сохраняться примыкающие к словам знаки пунктуации (например, точки или дефисы), что требует дополнительной обработки.

Удаление всех неалфавитных символов из текста некорректно, поскольку некоторые из них могут быть частью терминов и аббревиатур. Их полное удаление может исказить смысл. Более обоснованный подход — удаление токенов, в которых доля неалфавитных символов превышает установленный порог (например, 50 процентов). Это реализуется путём подсчёта соотношения буквенных и небуквенных символов в каждом токене.

Алгоритм фильтрации включает следующие шаги:

1. Удаление неалфавитных символов в начале и конце токена;
2. Подсчёт количества неалфавитных символов в токене;
3. Удаление токена, если доля неалфавитных символов превышает 50 процентов.

Реализация соответствующих функций представлена в листинге 9 [17].

```
1 def __count_letters_in_token__(self, token: str) -> int:
2     num_letters = 0
3     for symb in token:
4         if ("a" <= symb and symb <= "z") or ("A" <= symb and
5             symb <= "Z"):
6             num_letters += 1
7         if ("а" <= symb and symb <= "я") or ("А" <= symb and
8             symb <= "Я"):
9             num_letters += 1
10    return num_letters
11 def __strip_non_letters__(self, text: str) -> str:
12    return re.sub(r"^[^a-zA-Za-zA-яА-ЯёЁ]+|[^a-zA-Za-zA-яА-ЯёЁ]+$",
13        "", text)
14 def __processing_token__(self, token: str) -> str:
15    new_token = self.__strip_non_letters__(token)
16    return new_token if
17        (self.__count_letters_in_token__(new_token) +
18            1.0) / (len(new_token) + 1.0) >= 0.5
```

else ""

Листинг 9: Реализация удаления неалфавитных токенов

2.2.5 Токенизация, лемматизация и удаление стоп-слов по словарю

Библиотека SpaCy предоставляет унифицированный интерфейс для лингвистической обработки текста [22]. Её функционал позволяет выполнять всё в одном конвейере [22]:

- Токенизацию;
- Лемматизацию;
- Идентификацию стоп-слов

Принцип работы [22]:

1. На вход подаётся текстовая строка;
2. Обработанные данные возвращаются в виде последовательности токенов;
3. Каждый токен содержит:
 - Исходную словоформу;
 - Нормализованную лемму;
 - Флаг принадлежности к стоп-словам

Результирующая строка формируется путём фильтрации: сохраняются только леммы токенов, не отнесённых к стоп-словам.

Пример обработки русскоязычного текста показан в листинге 10 [22].

```
1 self.nlp_ru = spacy.load("ru_core_news_sm")
2 parts = self.__split_into_en_and_ru__(cell)
3 if part[1]:
4     tokens += [
5         self.__processing_token__(token.lemma_)
6         for token in self.nlp_ru(
7             self.__remove_extra_spaces_and_line_breaks__(part[1])
8         ) if not (token.is_stop) and not (token.is_punct) and
9         len(self.__processing_token__(token.lemma_)) > 1
10    ]
```

Листинг 10: Обработка строки русского языка средствами SpaCy

Полный алгоритм предобработки, объединяющий нормализацию пробелов, токенизацию и фильтрацию, реализован в листинге В [22].

2.2.6 Удаление высокочастотных и низкочастотных токенов

Помимо стандартных стоп-слов и стоп-слов, вычисленных с помощью метрики TF-IDF, следует учитывать токены, встречающиеся либо в слишком большом, либо в слишком малом количестве документов.

Токены, присутствующие в подавляющем большинстве документов, обычно не несут смысловой нагрузки для конкретной темы, поскольку являются общеупотребительными для всего корпуса.

Токены, встречающиеся в крайне малом числе документов, также имеют ограниченную ценность, так как их редкость снижает способность характеризовать тематические различия.

Алгоритм удаления таких токенов включает следующие шаги:

1. Определение нижнего и верхнего порогов встречаемости токенов в документах;
2. Вычисление для каждого токена количества документов, в которых он встречается;
3. Удаление токенов, частота встречаемости которых выходит за установленные пороги.

Реализация алгоритма представлена в листинге 11 [17, 18].

```
1 def __calc_num_docs_for_words__(self) -> None:
2     self.num_docs_for_words = dict()
3     for row in range(self.p_data.shape[0]):
4         for column in self.processing_columns:
5             words = self.p_data.loc[row, column].split(" ")
6             for word in words:
7                 if word in self.num_docs_for_words.keys():
8                     self.num_docs_for_words[word] += 1
9                 else:
10                     self.num_docs_for_words[word] = 1
11 def __calc_up_and_down_threshold__(self):
12     self.up_threshold = self.p_data.shape[0] * (
13         len(self.processing_columns) / 2.0)
14     self.down_threshold = self.p_data.shape[0] / 1000.0
15 for word in words:
16     if self.num_docs_for_words[word] >= self.down_threshold and \
17     self.num_docs_for_words[word] <= self.up_threshold:
```

```
new_words.append(word)
```

Листинг 11: Удаление токенов с экстремальной частотой встречаемости в документах

2.2.7 Удаление стоп-слов с помощью метрики TF-IDF

Как отмечалось ранее, удаление стоп-слов исключительно по предзаданному словарю имеет ограниченную эффективность. Для повышения качества фильтрации предлагается дополнительное использование метрики TF-IDF, позволяющей оценивать значимость терминов в корпусе документов [21].

Алгоритм расширенной фильтрации:

1. Вычисление TF-IDF:

- а) Формирование словаря терминов с помощью Gensim;
- б) Построение частотного корпуса документов;
- в) Расчёт весов TF-IDF для каждого термина

Реализация базового расчёта представлена в листинге 12 [23]:

```
1 def calc_tfidf_corpus_without_zero_score_tokens(self) ->
    None:
2     texts = []
3     self.original_tokens = []
4     for row in range(self.p_data.shape[0]):
5         words = []
6         for column in self.processing_columns:
7             for word in self.p_data.loc[row,
                column].split(" "):
8                 words.append(word)
9             self.original_tokens.append(words)
10            texts.append(words)
11    dictionary = gensim.corpora.Dictionary(texts)
12    corpus = [dictionary.doc2bow(text) for text in texts]
13    tfidf = gensim.models.TfidfModel(corpus)
14    self.tfidf_corpus = tfidf[corpus]
15    self.tfidf_dictionary = dictionary
```

Листинг 12: Вычисление TF-IDF метрик для текстового корпуса

2. Коррекция словаря:

- а) Добавление терминов с нулевым TF-IDF, исключённых Gensim по умолчанию [23];

б) Нормализация структуры данных для последующего анализа;
Соответствующая доработка реализована в листинге 13 [23]:

```
1 def add_in_tfidf_corpus_zero_score_tokens(self) -> None:
2     full_corpus = []
3     for doc_idx, doc in enumerate(self.tfidf_corpus):
4         original_words = self.original_tokens[doc_idx]
5         term_weights = {
6             self.tfidf_dictionary.get(term_id): weight
7             for term_id, weight in doc
8         }
9         full_doc = []
10        for word in original_words:
11            if word in term_weights:
12                weight = term_weights[word]
13            else:
14                weight = 0.0
15            full_doc.append((word, weight))
16        full_corpus.append(full_doc)
17    self.tfidf_corpus = full_corpus
```

Листинг 13: Дополнение словаря токенами с нулевыми TF-IDF значениями

3. Определение порога отсека:

- а) Вычисление n-го перцентиля распределения TF-IDF;
- б) Установка границы для отбора малозначимых терминов;

Логика расчёта границы показана в листинге 14 [24]:

```
1 def calc_threshold_for_tfidf_stop_words(self,
2     tfidf_percent_treshold) -> None:
3     all_tfidf_values = []
4     for doc in self.tfidf_corpus:
5         for _, tfidf_value in doc:
6             all_tfidf_values.append(tfidf_value)
7     self.threshold_for_tfidf_stop_words = np.percentile(
8         all_tfidf_values, tfidf_percent_treshold
```

Листинг 14: Определение порогового значения TF-IDF

4. Фильтрация датасета:

- а) Итеративное удаление терминов с $TF' = IDF$ ниже порога;
- б) Дополнительная очистка низкочастотных слов (менее k вхождений);

Финальный этап обработки представлен в листинге 15:

```
1 def del_tfidf_stop_words(self, tfidf_percent_threshold) ->
    None:
2     self.__calc_tfidf_corpus_without_zero_score_tokens__()
3     self.__add_in_tfidf_corpus_zero_score_tokens__()
4     self.__calc_threshold_for_tfidf_stop_words__(tfidf_percent_thresho
5     for row, doc in zip(range(self.p_data.shape[0]),
        self.tfidf_corpus):
6         tfidf_stop_words = [
7             word for word, tfidf_value in doc
8             if tfidf_value <
                self.threshold_for_tfidf_stop_words
9         ]
10        for column in self.processing_columns:
11            words_without_tfidf_stop_words = []
12            for word in self.p_data.loc[row,
                column].split(" "):
13                if word in tfidf_stop_words:
14                    continue
15                words_without_tfidf_stop_words.append(word)
16            self.p_data.loc[
17                row, column] = "
                ".join(words_without_tfidf_stop_words)
```

Листинг 15: Удаление стоп-слов на основе TF-IDF метрики

2.2.8 Очистка набора данных от пустых документов

После удаления стоп-слов и неалфавитных символов необходимо выполнить заключительный шаг — удаление документов, содержащих недостаточное количество токенов или не содержащих их вовсе. Это важно для обеспечения корректности последующего тематического моделирования и глубокого обучения.

Реализация данного этапа представлена в листинге 16 [17].

```
1 def __count_num_words__(self, doc: str) -> int:
2     return len(doc.split(" "))
3 def __del_docs_with_low_num_words__(self) -> None:
4     mask = self.p_data[self.processing_columns].apply(
5         lambda col: col.apply(self.__count_num_words__)
6     ).sum(axis=1)
7     self.p_data = self.p_data[mask >= 80]
```

```
8 self.p_data = self.p_data.reset_index(drop=True)
```

Листинг 16: Удаление документов с недостаточным количеством токенов

Таким образом, был реализован полный процесс подготовки текстовых данных для последующего анализа. Исходный код обработчика данных доступен в приложении Г.

2.3 Количественные характеристики обработанного и необработанного датасета

В ходе исследования выполнена обработка новостного массива с вариацией параметров, включая:

1. Пороговые значения TF-IDF;
2. Комбинации методов предобработки.

Количественные характеристики результатов представлены в таблицах приложения И.

Ключевые наблюдения:

1. Объём документов: медианное количество токенов на документ составляет 305, что свидетельствует о содержательной насыщенности материалов;
2. Эффективность фильтрации:
 - Частота наиболее распространённого слова снизилась с 800,000+ до 50,000 вхождений;
 - Количество уникальных токенов сократилось на 50 процентов (рис. 5-6).

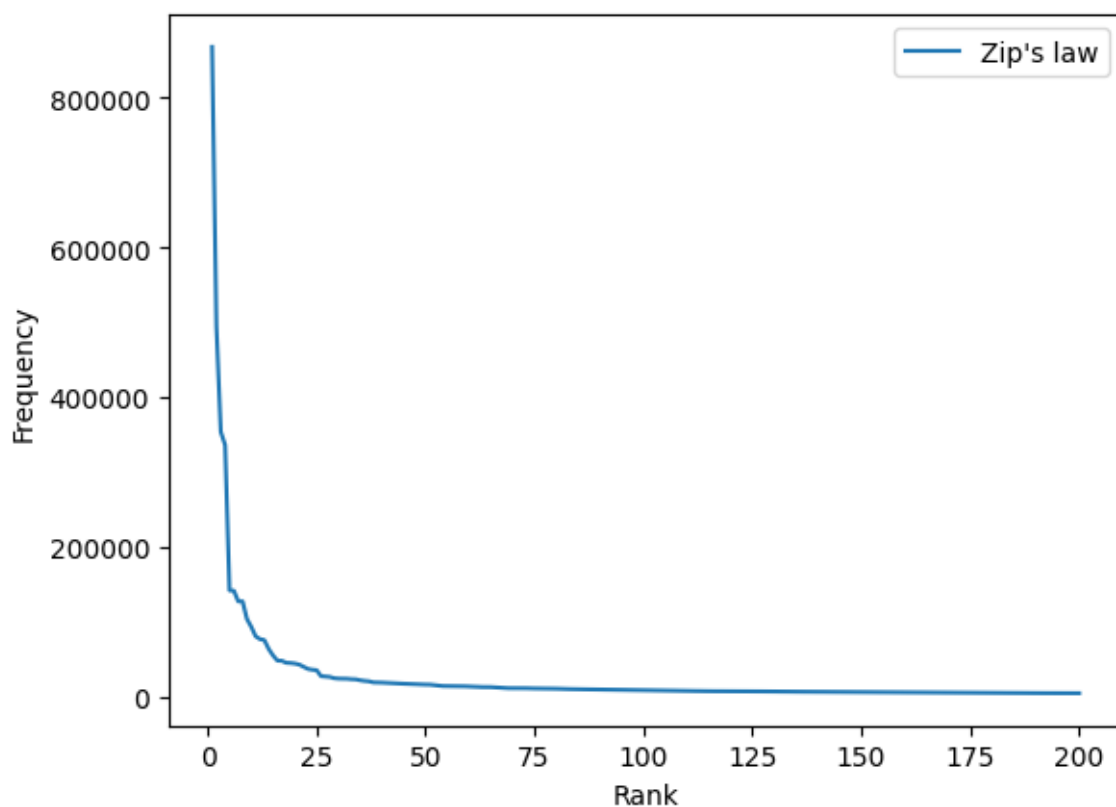


Рисунок 5 – Распределение частот слов по закону Ципфа (исходные данные)

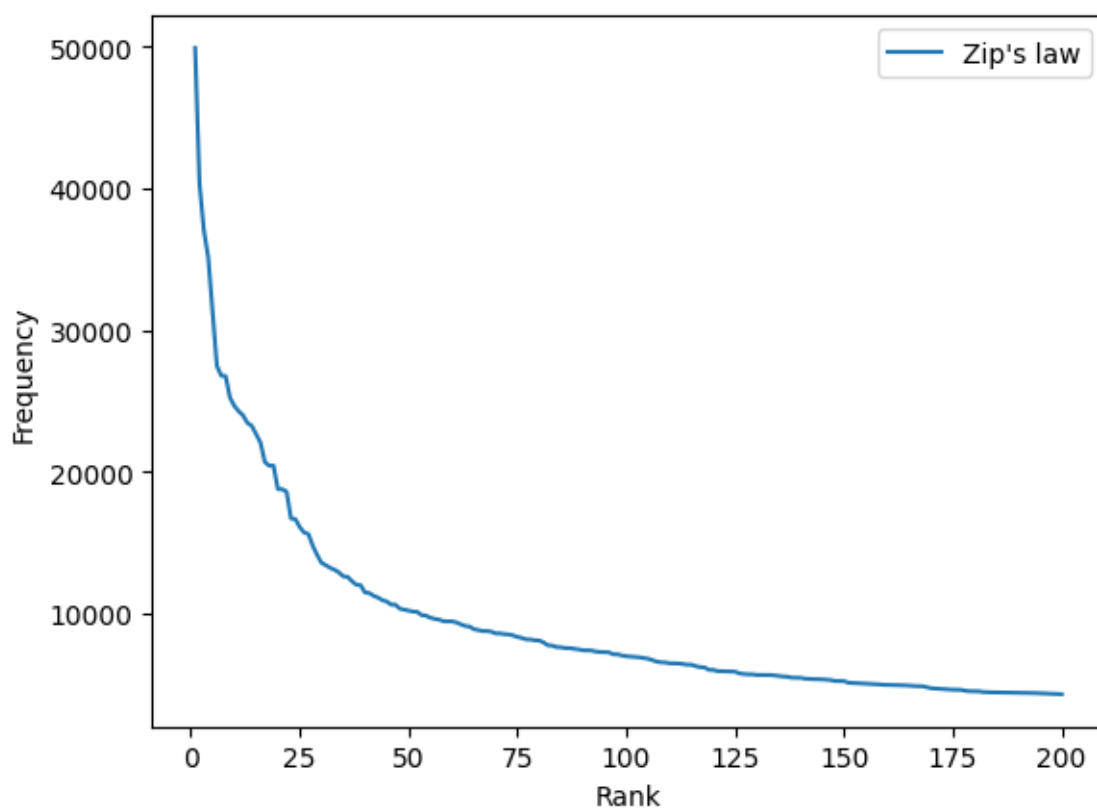


Рисунок 6 – Распределение частот слов по закону Ципфа (обработанные данные)

Проблемные аспекты:

1. Сохранение высокого числа уникальных токенов (?45 процентов от исходного);
2. Наличие шумовых компонентов:
 - Опечатки;
 - Ненормализованные словоформы;
 - Специфические аббревиатуры.

Указанные факторы могут негативно влиять на качество:

- Тематического моделирования;
- Обучения ML-алгоритмов;
- Интерпретации результатов.

2.4 Вычисление тематической модели

2.4.1 Выбор инструментов для тематического моделирования

При разработке системы автоматической классификации новостей выбор инструментов напрямую влияет на гибкость, скорость и качество модели. Библиотека BigARTM (Additive Regularization of Topic Models) была выбрана по нескольким ключевым критериям, которые делают её предпочтительной на фоне альтернатив, таких как Gensim или Mallet.

Критерии выбора:

1. Удобный интерфейс: BigARTM предоставляет простой API для работы с тематическими моделями, что ускоряет интеграцию в существующие пайплайны обработки текстов. Например, загрузка данных, настройка параметров и запуск обучения выполняются минимальным количеством кода, снижая риск ошибок и время на разработку;
2. Разнообразие регуляризаторов: библиотека поддерживает множество регуляризаторов (например, сглаживание, разреживание тем), которые можно комбинировать для улучшения интерпретируемости и точности модели. Это критически важно для новостных данных, где темы часто пересекаются (например, «экономика» и «политика»);
3. Блочный синтаксис: настройка модели в BigARTM осуществляется через декларативное описание компонентов (блоков), что упрощает эксперименты с архитектурой. Например, можно быстро добавить регуляризатор для контроля за размером тем или подключить модуль для обработки мультимодальных данных;

4. Доступность tutorиалов: BigARTM имеет подробную документацию и примеры использования, включая готовые сценарии для классификации текстов. Это сокращает время на изучение библиотеки и позволяет сосредоточиться на решении прикладных задач.

BigARTM сочетает в себе специализацию для работы с текстами, гибкость настройки и низкий порог входа благодаря понятному синтаксису. Это делает её оптимальным выбором для задач автоматической классификации новостей, где важно быстро адаптировать модель под изменяющиеся условия (например, появление новых тем) и контролировать качество результатов.

2.4.2 Недостающий функционал библиотеки BigARTM

Тематическое моделирование с использованием библиотеки BigARTM обладает практической ценностью, но имеет ряд ограничений:

1. Отсутствие встроенной метрики оценки когерентности тематик;
2. Сложность интеграции регуляризаторов из-за многоэтапного API;
3. Трудоёмкое преобразование данных в требуемый формат представления;
4. Недостаток инструментов визуализации для мониторинга качества моделей;
5. Отсутствие автоматизированных методов подбора гиперпараметров.

Наибольшее влияние на качество моделирования оказывает первый фактор. Остальные ограничения преимущественно связаны с эргономикой рабочего процесса, но их совокупность существенно увеличивает сложность поддержки кодовой базы.

Для компенсации выявленных недостатков предлагается разработка двух вспомогательных классов, расширяющих функционал библиотеки:

1. `My_BigARTM_model` — обёртка над BigARTM для добавления недостающих метрик, их визуализаций, а также для упрощения взаимодействия с BigARTM;
2. `Hyperparameter_optimizer` — автоматический оптимизатор гиперпараметров.

2.4.3 Функциональности классов `My_BigARTM_model` и `Hyperparameter_optimizer`

В рамках класса `My_BigARTM_Model` целесообразно реализовать:

- Расчёт метрик когерентности тематик;

- Упрощённый интерфейс для добавления регуляризаторов;
- Автоматизация преобразования данных в требуемый формат;
- Визуализация динамики метрик качества через графики.

Интеграция функциональности по подбору гиперпараметров в данный класс нецелесообразно, так как это:

- Нарушит принцип единственной ответственности;
- Усложнит поддержку кодовой базы;
- Снизит читаемость реализации.

Для решения этих задач предложено выделение отдельного класса `Hyperparameter` который:

- Реализует логику оптимизации гиперпараметров;
- Обеспечивает удобное сохранение настроенных моделей.

Такое разделение обеспечивает модульность архитектуры и упрощает дальнейшее расширение системы.

Следующим этапом работы является последовательная реализация обоих классов.

2.4.4 Преобразование новостного массива в приемлемый для BigARTM формат

Модель BigARTM поддерживает ограниченный набор форматов данных, включая Vowpal Wabbit [25]. Для интеграции с `pandas DataFrame` требуется предварительное преобразование новостного массива, которое целесообразно реализовать отдельной функцией.

Алгоритм преобразования:

1. Извлечение строки из `DataFrame`;
2. Конкатенация ячеек строки в единый текстовый блок;
3. Запись результата в файл формата Vowpal Wabbit с меткой документа;
4. Итеративная обработка всего массива новостей.

Реализация функции преобразования представлена в листинге 17 [17, 18].

```

1 def __make_vowpal_wabbit__(self) -> None:
2     f = open(self.path_vw, "w")
3     for row in range(self.data.shape[0]):
4         string = ""
5         for column in self.data.columns:
6             string += str(self.data.loc[row, column]) + " "
```

```
7         f.write("doc_{0} ".format(row) + string.strip() + "\n")
```

Листинг 17: Преобразование новостного массива в формат Vowpal Wabbit

Последующие этапы обработки:

1. Разделение данных на батчи;
2. Генерация словаря терминов.

Оба действия выполняются средствами библиотеки BigARTM. Соответствующий код приведён в листинге 18 [25]:

```
1 def __make_batches__(self) -> None:
2     self.batches = artm.BatchVectorizer(
3         data_path=self.path_vw,
4         data_format="vowpal_wabbit",
5         batch_size=self.batch_size,
6         target_folder=self.dir_batches
7     )
8     self.dictionary = self.batches.dictionary
```

Листинг 18: Функция создания батчей и словаря

Подготовленные данные готовы для передачи в модель BigARTM для тематического моделирования.

2.4.5 Удобное добавление регуляризаторов

Библиотека BigARTM предоставляет обширный набор регуляризаторов, однако их интеграция в модель требует знания непростого синтаксиса, что затрудняет их использование. Для упрощения процесса предложен двухуровневый подход:

1. Базовая функция — добавляет регуляризатор по имени и значению гиперпараметра;
2. Обёрточная функция — применяет первый метод для массового добавления.

Преимущества решения:

- Устранение необходимости работы с низкоуровневым API BigARTM;
- Единообразный интерфейс для одиночных и групповых операций;
- Повышение читаемости и поддерживаемости кода.

Фрагмент реализации базовой функции (листинг 19 [25]):

```
1 def add_regularizer(self, name: str, tau: float = 0.0) -> None:
2     if name == "SmoothSparseThetaRegularizer":
```

```

3         self.model.regularizers.add(
4             artm.SmoothSparseThetaRegularizer(name=name, tau=tau)
5         )
6         self.user_regularizers[name] = tau
7     elif name == "SmoothSparsePhiRegularizer":
8         self.model.regularizers.add(
9             artm.SmoothSparsePhiRegularizer(name=name, tau=tau)
10        )
11    # остальные регуляризаторы ...
12    else:
13        print(
14            "Регуляризатора {0} нет! Проверьте корректность назва
15                ния!".
16            format(name)
17        )

```

Листинг 19: Функция добавления одиночного регуляризатора

Реализация массового добавления регуляризаторов (листинг 20):

```

1 def add_regularizers(self, regularizers: dict[str, float]) ->
    None:
2     for regularizer in regularizers:
3         self.add_regularizer(regularizer,
                               regularizers[regularizer])

```

Листинг 20: Функция добавления набора регуляризаторов

Данное решение существенно упрощает эксперименты с различными комбинациями регуляризаторов, сохраняя при этом гибкость подхода BigARTM.

2.4.6 Вычисление когерентности

Библиотека BigARTM включает набор встроенных метрик оценки качества, однако не поддерживает расчёт когерентности — ключевого показателя тематической согласованности [3]. Для восполнения этого функционала предлагается интеграция с библиотекой Gensim, предоставляющей методы вычисления различных видов когерентности [23].

Алгоритм расчёта метрики:

1. Экспорт тематических ядер:

Получение списка тем, где каждая тема представлена N ключевыми терминами;

2. Подготовка текстового корпуса:

Преобразование документов в структуру вида:

[[токен_1_док_1, токен_2_док_1, ...], [токен_1_док_2, ...], ...];

3. Вычисление показателя:

Передача данных в Gensim для расчёта выбранного типа когерентности.

Реализация функции представлена в листинге 21 [23]:

```
1 def __calc_coherence__(self) -> None:
2     last_tokens =
3         self.model.score_tracker["top_tokens"].last_tokens
4     valid_topics = [tokens for tokens in last_tokens.values() if
5         tokens]
6     texts = []
7     for row in range(self.data.shape[0]):
8         words = []
9         for column in self.data.columns:
10            cell_content = self.data.loc[row, column]
11            if isinstance(cell_content, str) and
12                cell_content.strip():
13                words += cell_content.split()
14        if words:
15            texts.append(words)
16    dictionary = Dictionary(texts)
17    coherence_model = CoherenceModel(
18        topics=valid_topics,
19        texts=texts,
20        dictionary=dictionary,
21        coherence="c_v"
```

Листинг 21: Функция вычисления метрики когерентности

2.4.7 Вычисление тематической модели и формирование графиков метрик

Библиотека BigARTM не поддерживает мониторинг динамики метрик качества в процессе обучения [25], особенно для пользовательских метрик. Для реализации этого функционала требуется дополнительная разработка.

Алгоритм отслеживания метрик:

1. Итеративное обучение модели:

- Установка `num_collection_passes=1` для пошагового прохода [25];
- Циклическое выполнение обучения с накоплением метрик после каждой эпохи.

2. Визуализация результатов:

- Использование `matplotlib` для построения графиков;
- Унифицированный подход для различных типов метрик.

Реализация итеративного обучения представлена в листинге 35 [25]:

Пример визуализации для метрики когерентности (листинг 22 [26]):

```

1 def print_coherence_by_epochs(self) -> None:
2     plt.plot(
3         range(len(self.coherence_by_epoch)),
4         self.coherence_by_epoch,
5         label="coherence"
6     )
7     plt.title("График когерентности")
8     plt.xlabel("Epoch")
9     plt.ylabel("Coherence")
10    plt.legend()
11    plt.show()

```

Листинг 22: Функция построения графика динамики когерентности

Для других метрик применяется аналогичная логика с заменой целевого показателя.

Данная реализация завершает базовый функционал класса `My_BigARTM_model`. Полный код доступен в приложении Е.

2.4.8 Подбор гиперпараметров для тематического моделирования

Для интеллектуального подбора гиперпараметров целесообразно использовать библиотеку `Optuna`, которая предоставляет [27]:

- Упрощённый API для настройки экспериментов;
- Поддержку байесовской оптимизации (вместо полного перебора);
- Автоматическое сокращение вычислительных ресурсов за счёт адаптивного выбора параметров.

Алгоритм работы:

1. Реализация целевой функции:

- Определение пространства поиска гиперпараметров через `trial.suggest_int()` и `trial.suggest_float()`;

— Вычисление и возврат метрик качества модели.

Ключевой фрагмент реализации (листинг 23 [27]):

```
1 def __objective__(self, trial) -> tuple[float, float,
    float]:
2     num_topics = trial.suggest_int(
3         self.num_topics[0], self.num_topics[1],
4         self.num_topics[2]
5     )
6     # скрытые остальные гиперпараметры ...
7     model = My_BigARTM_model(
8         data=self.data,
9         num_topics=num_topics,
10        num_document_passes=num_document_passes,
11        class_ids=class_ids,
12        num_collection_passes=num_collection_passes,
13        regularizers=regularizers
14    )
15    model.calc_model()
16    return model.get_perplexity(), model.get_coherence(
    ), model.get_topic_purities()
```

Листинг 23: Целевая функция для оптимизации гиперпараметров

2. Запуск оптимизации:

- Использование `study.optimize()` для выполнения экспериментов;
- Получение набора попыток с параметрами и метриками.

3. Выбор оптимальной конфигурации:

- Нормализация метрик;
- Выбор попытки с минимальной совокупной ошибкой.

Логика выбора (листинг 37 [27]):

4. Финализация модели:

- Обучение на лучших гиперпараметрах;
- Возврат оптимизированной модели.

Завершающий этап (листинг 24 [27]):

```
1 def optimizer(self):
2     study = optuna.create_study(
3         directions=["minimize", "maximize", "maximize"])
4     study.optimize(self.__objective__,
5                   n_trials=self.n_trials)
6     best_trial = self.__select_best_trial__(study,
```

```

        weights=[1, -1, -1])
6     best_params = best_trial[0]
7     num_topics = best_params["num_topics"]
8     # скрытые остальные параметры ...
9     # скрытый фрагмент создания финальной модели
10    final_model.calc_model()
11    self.model = final_model

```

Листинг 24: Обучение модели с оптимальными параметрами

Полная реализация класса `Hyperparameter_optimizer` доступна в приложении 3.

2.5 Результаты тематического моделирования

В ходе исследования проведено тематическое моделирование 13 конфигураций предобработанных данных. Для каждой конфигурации выполнены:

1. Оптимизация гиперпараметров;
2. Расчёт финальной модели;
3. Оценка метрик качества.

Результаты оценки представлены в таблице 1 (когерентность и перплексия) и таблице 2 (оптимальные гиперпараметры).

Таблица 1 – Метрики моделей

Данные	perplexity	coherence
Без tfidf и add.	3486	0.470
Без tfidf с add.	2974	0.456
С tfidf 1 пр.	3643	0.476
С tfidf 2 пр.	3848	0.479
С tfidf 5 пр.	4094	0.495
С tfidf 6 пр.	3982	0.505
С tfidf 7 пр.	4620	0.491
С tfidf 8 пр.	4183	0.514
С tfidf 9 пр.	3811	0.496
С tfidf 10 пр.	4022	0.490
С tfidf 10 пр. с add.	3284	0.486

Таблица 2 – Гиперпараметры моделей

Данные	topics	cols	docs	tau phi	tau theta
Без tfidf и add.	8	6	7	-1.561	0.809
Без tfidf с add.	8	5	6	-0.004	-0.653
С tfidf 1 пр.	6	7	5	-1.540	-0.038
С tfidf 2 пр.	8	6	4	-0.101	0.146
С tfidf 5 пр.	8	6	6	1.139	-1.981
С tfidf 6 пр.	8	6	7	0.954	-1.353
С tfidf 7 пр.	8	5	5	0.942	-0.102
С tfidf 8 пр.	6	7	7	1.757	-1.222
С tfidf 9 пр.	8	6	7	-0.449	-0.365
С tfidf 10 пр.	8	5	6	-0.184	-1.826
С tfidf 10 пр. с add.	8	5	6	0.385	-1.165

Ключевые наблюдения:

- Наилучшее качество (когерентность 0.514) достигнуто при:
 - Удалении низкочастотных слов;
 - Отказе от TF-IDF фильтрации стоп-слов;
 - Пороге TF-IDF 8 процентов.
- Потенциальные причины результата:
 - Ограничения оптимизации: Неполный перебор гиперпараметров;
 - Недостаток вариантов: Не исследованы комбинации TF-IDF с удалением стоп-слов и низкочастотных терминов;
 - Методические риски: Возможная некорректность TF-IDF фильтрации стоп-слов (требует дополнительной проверки).
- Влияние порогов TF-IDF:
 - Пороги > 8 процентов приводят к снижению качества;
 - Высокие пороги удаляют смысловые термины;
 - Оптимальный диапазон: 5-8 процентов.

Возможные пути улучшения:

- Расширить пространство поиска гиперпараметров;
- Исследовать комбинированные стратегии очистки данных;
- Провести валидацию метода TF-IDF фильтрации стоп-слов.

2.6 Обучение модели классификатора

2.6.1 Выбор модели для тематической классификации

Как установлено ранее [1.5.2](#), для решения задачи наиболее эффективны сети-трансформеры. Существует три основных типа архитектур [\[10\]](#):

- Encoder-only (BERT, RoBERTa): Содержат только кодирующую часть;
- Decoder-only (GPT): Содержат только декодирующую часть;
- Encoder-Decoder (BART, T5): Комбинируют обе части.

Их функциональные различия можно описать следующим образом [\[10\]](#):

- Encoder модели (BERT, RoBERTa) специализируются на понимании текста (задачи классификации, извлечения информации);
- Decoder модели (GPT) оптимизированы для задачи генерации текста;
- Гибридные модели (BART, T5) предназначены для задачи трансформации текста (перевод, суммаризация).

Для тематической классификации требуется глубокое понимание контекста, поэтому оптимальны encoder-only модели. Среди них RoBERTa (Robustly optimized BERT approach) демонстрирует преимущества перед BERT [\[28\]](#):

- Обучена на большем объёме данных;
- Использует динамическое маскирование слов;
- Исключает задачу предсказания следующего предложения;
- Показывает лучшие результаты на NLU-задачах.

Таким образом, для классификации новостей выберем RoBERTa.

2.6.2 Выбор способа для получения предобученных моделей

Существует несколько способов получения весов предобученной модели: от их скачивания с облака и github репозиторий, до получения через API разных сайтов. Из этих методов будет предпочтительнее выбрать последний, так как есть портал Hugging Face.

Hugging Face представляет собой большое хранилище различных моделей, в том числе и предобученных крупными компаниями и исследователями (Google, Facebook, Sberbank). Кроме того, данный сайт предоставляет удобный, лаконичный и унифицированный интерфейс для работы с ним, что позволяет делать код максимально компактным и читабельным.

Таким образом, будет получать предобученные модели с помощью портала Hugging Face.

2.6.3 Получение весов предобученной модели

Для начала работы с нейронными сетями с платформы Hugging Face необходимо подключить следующие зависимости [29]:

```
1 !pip install transformers datasets evaluate
2
3 from datasets import Dataset
4 from transformers import (
5     AutoTokenizer ,
6     AutoModelForSequenceClassification ,
7     TrainingArguments ,
8     Trainer ,
9     EarlyStoppingCallback
10 )
11 import evaluate
```

Листинг 25: Подключение необходимых зависимостей для работы с Hugging Face

С помощью данных библиотек будут происходить подготовка данных, загрузка весов моделей и их обучение.

Для загрузки модели потребуется класс `AutoModelForSequenceClassification` и его метод `from_pretrained`, в который будут задаваться параметры загрузки (название модели и тип решаемой ей задачи, для загрузки предобученной на соответствующих данных модели) [29]. Реализация соответствующего кода представлена в соответствующем листинге 26.

```
1 self.model = AutoModelForSequenceClassification.from_pretrained(
2     self.model_name ,
3     num_labels=self.num_labels ,
4     problem_type="single_label_classification" ,
5     ignore_mismatched_sizes=True
6 ).to(self.device)
```

Листинг 26: Загрузка весов модели

2.6.4 Подготовка данных для работы с моделью

Для обработки текста используется токенизатор, соответствующий выбранной модели. Его загрузка осуществляется через класс `AutoTokenizer` [29] (Листинг 27):

```
1 self.tokenizer = AutoTokenizer.from_pretrained(self.model_name)
```

Листинг 27: Загрузка предобученного токенизатора

Токенизатор преобразует сырой текст в формат, пригодный для нейросети. Обработка данных выполняется через метод `map` класса `Dataset` с применением функции токенизации (Листинг 28) [29]:

```
1 def __tokenize_data__(self, df: pd.DataFrame) -> Dataset:
2     dataset = Dataset.from_pandas(df[['text', 'label']])
3     def tokenize_function(examples):
4         return self.tokenizer(
5             examples["text"],
6             padding="max_length",
7             truncation=True,
8             max_length=self.maximum_sequence_length
9         )
10    return dataset.map(tokenize_function, batched=True)
```

Листинг 28: Функция токенизации текста

Отдельно преобразуются текстовые метки классов в числовые индексы (Листинг 29) [29]:

```
1 def __prepare_data__(self):
2     self.data['text'] = self.data[self.columns].apply(
3         lambda x: ' '.join(x.dropna().astype(str)), axis=1)
4     unique_topics = self.data['topic'].unique()
5     self.topic2id = {topic: i for i, topic in
6                     enumerate(unique_topics)}
7     self.id2topic = {i: topic for i, topic in
8                     enumerate(unique_topics)}
9     self.num_labels = len(self.topic2id)
10    self.data['label'] = self.data['topic'].map(self.topic2id)
```

Листинг 29: Кодировка меток классов

2.6.5 Дообучение модели

Выбранная модель (RoBERTa) не является сверхбольшой, а ресурсы Google Colab предоставляют доступ к мощным GPU (Tesla T4/V100), что позволяет дообучить всю архитектуру без заморозки слоёв.

Перед обучением нужно сначала задать его параметры, реализуется это с

помощью класса `TrainingArguments`, в конструктор которого передаются соответствующие параметры [29]. Среди них можно выделить следующие [29]:

- Стратегия обучения (`eval_strategy`);
- Стратегия сохранения результата (`save_strategy`);
- Шаг ошибки (`learning_rate`);
- Размер батча (`per_device_train_batch_size`, `per_device_eval_batch_size`);
- Количество эпох обучения (`num_train_epochs`);
- Метрика качества подбора лучшей модели (`metric_for_best_model`).

Соответствующий код можно увидеть в следующем листинге 30.

```
1 training_args = TrainingArguments(  
2     output_dir=self.output_dir,  
3     eval_strategy="epoch",  
4     save_strategy="epoch",  
5     learning_rate=2e-5,  
6     lr_scheduler_type="linear",  
7     warmup_steps=100,  
8     per_device_train_batch_size=32,  
9     per_device_eval_batch_size=32,  
10    num_train_epochs=10,  
11    weight_decay=0.01,  
12    load_best_model_at_end=True,  
13    metric_for_best_model="accuracy",  
14    logging_dir='./logs',  
15    logging_steps=10,  
16    report_to="none",  
17    save_total_limit=1  
18 )
```

Листинг 30: Код установки параметров обучения

Осталось только создать объект тренировщика и запустить его. Делается это с помощью класса `Trainer` следующим образом 31.

```
1 self.trainer = Trainer(  
2     model=self.model,  
3     args=training_args,  
4     train_dataset=train_dataset,  
5     eval_dataset=val_dataset,  
6     compute_metrics=self.__compute_metrics__,  
7     callbacks=[EarlyStoppingCallback(early_stopping_patience=3)]  
8 )
```

```
9 self.trainer.train()
```

Листинг 31: Код класса обучения

Таким образом, была реализована основная функциональность для обучения тематического классификатора. Полный код можно увидеть в соответствующем приложении **К**.

2.7 Результаты обучения классификатора

Эксперименты по обучению классификатора на результатах тематического моделирования показали низкую эффективность (таблица **3**):

Таблица 3 – Метрики классификатора для разных наборов данных

Конфигурация данных	Accuracy	F1
Без TF-IDF фильтрации	0.17	0.17
Без TF-IDF + доп. очистка	0.18	0.17
TF-IDF порог 1%	0.15	0.15
TF-IDF порог 2%	0.17	0.17

После получения результатов выше были выполнены следующие попытки улучшения:

1. Дополнительная фильтрация слов:
 - Удаление редких (< 20 документов) и частых (> 3000 документов) терминов;
 - Исключение документов с коротким текстом (< 80 и < 150 токенов);
2. Сокращение словаря до 5000 наиболее значимых слов (по матрице ϕ)
3. Расширенный подбор гиперпараметров и регуляризаторов в BigARTM;

Ни один метод не дал значительных улучшений, полученная Accuracy не превысила 0.22. Тестирование альтернативных моделей (BERT, полносвязные сети) также показало низкое качество (Accuracy ≤ 0.18).

Основное предположение заключается в следующем: проблема в некорректных тематических метках. Для его проверки была использована тематическая разметка самого сайта ВШЭ. Результаты получились следующие:

- Точность на первой эпохе: Accuracy = 0.60;
- Подтверждено: низкое качество обусловлено ошибками тематического моделирования.

Таким образом, основная проблема — некорректное присвоение тем документам при тематическом моделировании, что подтверждается:

1. Низкими значениями метрик качества при использовании BigARTM разметки обучающих данных и высоким при разметки ВШЭ;
2. Сравнимым объёмом данных (количество документов/токенов) и их характеристиках при тестировании разметки ВШЭ.

2.8 Выводы и возможные улучшения по практико-методической части

Исходя из разделов 2.7, 2.5, 2.3, узким местом выбранного подхода автоматической классификации новостей является этап тематического моделирования.

Для решения этой проблемы предлагаются следующие методы:

1. Улучшение подготовки данных;
2. Расширенная настройка гиперпараметров и регуляризаторов.

Однако оба подхода имеют ограничения:

- Подготовка данных уже включает стандартные методы (кроме N-грамм и продвинутой коррекции опечаток), что снижает потенциал улучшений;
- Библиотека BigARTM не поддерживает GPU-ускорение, что делает широкий поиск гиперпараметров вычислительно неэффективным.

Возможные улучшения классификатора:

- Автоматический подбор гиперпараметров (например, через Optuna);
- Тестирование альтернативных моделей (CTM, BERTopic).

Перспективы развития работы, если будет решена проблема с тематическим моделированием:

1. Рефакторинг кода: повышение модульности и читаемости классов;
2. Создание API для интеграции классификатора в приложения;
3. Разработка веб-интерфейса для пользовательской классификации.

ЗАКЛЮЧЕНИЕ

В ходе данной дипломной работы было рассмотрено одно из возможных решений задачи автоматической тематической классификации.

Для этого было выполнено следующее:

1. Проведён анализ инструментов по сбору данных и выбраны наиболее удобные из них (BeautifulSoup4, requests);
2. Проведён сбор данных;
3. Проанализированы способы обработки текстовых данных и выбраны наиболее удобные из них;
4. Проанализированы популярные инструменты для обработки текстовых данных (NLTK, Rymorphy3, SpaCy) и выбран наиболее удобный и точный из них (SpaCy);
5. Проведена подготовка данных для тематического моделирования и проведён анализ её результатов;
6. Изучен механизм тематического моделирования с помощью аддитивной тематической регуляризации;
7. Разработаны инструменты для тематической классификации с помощью библиотеки BigARTM;
8. Проведены эксперименты по проведению тематической классификации над подготовленными различными способами данными, а также проведён анализ результатов экспериментов;
9. Рассмотрены различные способы обработки текстовых данных нейронными сетями и выбран наиболее подходящий из них (семантическое векторное представление);
10. Проведён анализ архитектур подходящих типов нейронных сетей и выбрана наиболее подходящая из них (transformer);
11. Проведён анализ доступных предобученных сетей и сервисов, которые их предоставляют, в ходе которого выбран наиболее удобный из них (Hugging Face и Roberta);
12. Проведены эксперименты по обучению тематического классификатора новостей, а также выполнен анализ результатов и сделаны соответствующие выводы.

Также по результатам всей работы и разделов с анализом каждой из частей сделано заключение, что предложенный метод автоматической классификации

пока что не может считаться успешным или неуспешным, так как для этого не проведено достаточное количество экспериментов. Однако предложены возможные пути проведения экспериментов, которые помогут в дальнейшем подтвердить или опровергнуть результативность предложенного варианта автоматической тематической классификации.

Таким образом, все поставленные задачи работы были решены, а следовательно цель достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Парсинг данных: эффективные методы извлечения информации [Электронный ресурс]. — URL: <https://sky.pro/wiki/analytics/parsing-dannyh-effektivnye-metody-izvlecheniya-informatsii/> (Дата обращения 30.09.2024). Загл. с экр. Яз. рус.
- 2 Акжолов, Р. К. Предобработка текста для решения задач nlp / Р. К. Акжолов, А. В. Верига // *Вестник науки*. — 2020. — Т. 1, № 3. — С. 66–68.
- 3 Вероятностное тематическое моделирование: теория регуляризации ARTM и библиотека с открытым исходным кодом BigARTM [Электронный ресурс]. — URL: <http://www.machinelearning.ru/wiki/images/d/d5/Voron17survey-artm.pdf> (Дата обращения 30.10.2024). Загл. с экр. Яз. рус.
- 4 Воронцов, К. В. Аддитивная регуляризация тематических моделей коллекций текстовых документов / К. В. Воронцов // *Доклады академии наук*. — 2014. — Т. 456, № 3. — С. 676–687.
- 5 Николаевич, Ш. Вероятность-1 / Ш. Николаевич. — Москва: МЦНМО, 2021.
- 6 Таха, Х. Введение в исследование операций / Х. Таха. — Москва: Вильямс, 2007.
- 7 Вероятностные тематические модели Лекция 2. Постановка задачи, оптимизация и регуляризация [Электронный ресурс]. — URL: <http://www.machinelearning.ru/wiki/images/8/86/Voron25ptm-intro.pdf> (Дата обращения 13.11.2024). Загл. с экр. Яз. рус.
- 8 Воронцов, К. В. Регуляризация вероятностных тематических моделей для повышения интерпретируемости и определения числа тем / К. В. Воронцов, А. А. Потапенко // *Компьютерная лингвистика и интеллектуальные технологии*. — 2014. — Т. 13, № 20. — С. 268–271.
- 9 Введение в NLP. Эмбединги слов [Электронный ресурс]. — URL: <https://stepik.org/lesson/1421109/step/1?auth=login&unit=1439152> (Дата обращения 27.11.2024). Загл. с экр. Яз. рус.
- 10 Гольдберг, Й. Нейросетевые методы в обработке естественного языка / Й. Гольдберг. — Москва: ДМК Пресс, 2019.

- 11 Рекуррентные нейронные сети (RNN) [Электронный ресурс]. — URL: <https://stepik.org/lesson/1421112/step/1?auth=login&unit=1439155> (Дата обращения 03.12.2024). Загл. с экр. Яз. рус.
- 12 Архитектура Transformer [Электронный ресурс]. — URL: <https://stepik.org/lesson/1264624/step/1?auth=login&unit=1493641> (Дата обращения 10.12.2024). Загл. с экр. Яз. рус.
- 13 Основные метрики задач классификации в машинном обучении [Электронный ресурс]. — URL: <https://webiomed.ru/blog/osnovnye-metriki-zadach-klassifikatsii-v-mashinnom-obuchenii/> (Дата обращения 12.12.2024). Загл. с экр. Яз. рус.
- 14 Основы парсинга на Python: от Requests до Selenium [Электронный ресурс]. — URL: <https://habr.com/ru/companies/selectel/articles/754674/> (Дата обращения 20.01.2025). Загл. с экр. Яз. рус.
- 15 Requests: HTTP for Humans [Электронный ресурс]. — URL: <https://requests.readthedocs.io/en/latest/> (Дата обращения 20.01.2025). Загл. с экр. Яз. англ.
- 16 Модуль BeautifulSoup4 в Python, разбор HTML [Электронный ресурс]. — URL: <https://docs-python.ru/packages/paket-beautifulsoup4-python/> (Дата обращения 20.01.2025). Загл. с экр. Яз. рус.
- 17 Васильев, А. Программирование на PYTHON в примерах и задачах / А. Васильев. — Москва: Эксмо, 2021.
- 18 pandas documentation [Электронный ресурс]. — URL: <https://pandas.pydata.org/docs/> (Дата обращения 20.01.2025). Загл. с экр. Яз. англ.
- 19 Макаров, К. С. Сравнительный анализ библиотек для обработки естественного языка (nlp) / К. С. Макаров, А. А. Халин, Д. А. Костенков, Э. Э. Муханов // *Auditorium*. — 2024. — Т. 41, № 1.
- 20 Краткий обзор NLP библиотеки SpaCy [Электронный ресурс]. — URL: <https://habr.com/ru/articles/504680/> (Дата обращения 15.02.2025). Загл. с экр. Яз. рус.
- 21 Мера TF-IDF, сила связи слов и ключевые сочетания для безызыточной передачи единицы знаний [Электронный ресурс]. — URL: <http://www.>

- machinelearning.ru/wiki/images/7/79/Biomed_engin_2020_mdv_pres.pdf
(Дата обращения 15.02.2025). Загл. с экр. Яз. рус.
- 22 Industrial-Strength Natural Language Processing [Электронный ресурс]. — URL: <https://spacy.io/> (Дата обращения 17.02.2025). Загл. с экр. Яз. англ.
- 23 NLP Gensim Tutorial - Complete Guide For Beginners [Электронный ресурс]. — URL: <https://www.geeksforgeeks.org/nlp-gensim-tutorial-complete-guide-for-beginners/> (Дата обращения 19.02.2025). Загл. с экр. Яз. англ.
- 24 NumPy user guide [Электронный ресурс]. — URL: <https://numpy.org/doc/stable/user/index.html> (Дата обращения 19.02.2025). Загл. с экр. Яз. англ.
- 25 BigARTM's documentation [Электронный ресурс]. — URL: <https://docs.bigartm.org/en/stable/index.html> (Дата обращения 26.02.2025). Загл. с экр. Яз. англ.
- 26 Matplotlib 3.10.3 documentation [Электронный ресурс]. — URL: <https://matplotlib.org/stable/index.html> (Дата обращения 02.03.2025). Загл. с экр. Яз. англ.
- 27 Optuna: A hyperparameter optimization framework [Электронный ресурс]. — URL: <https://optuna.readthedocs.io/en/stable/> (Дата обращения 04.03.2025). Загл. с экр. Яз. англ.
- 28 A review of pre-trained language models: from BERT, RoBERTa, to ELECTRA, DeBERTa, BigBird, and more. [Электронный ресурс]. — URL: <https://tungmphung.com/a-review-of-pre-trained-language-models-from-bert-RoBERTa-to-electra-deberta-bigbird/#DistilBERT> (Дата обращения 01.04.2025). Загл. с экр. Яз. англ.
- 29 Hugging Face Documentations [Электронный ресурс]. — URL: <https://huggingface.co/docs> (Дата обращения 01.04.2025). Загл. с экр. Яз. англ.

ПРИЛОЖЕНИЕ А

Пример страницы новостного сайта ВШЭ

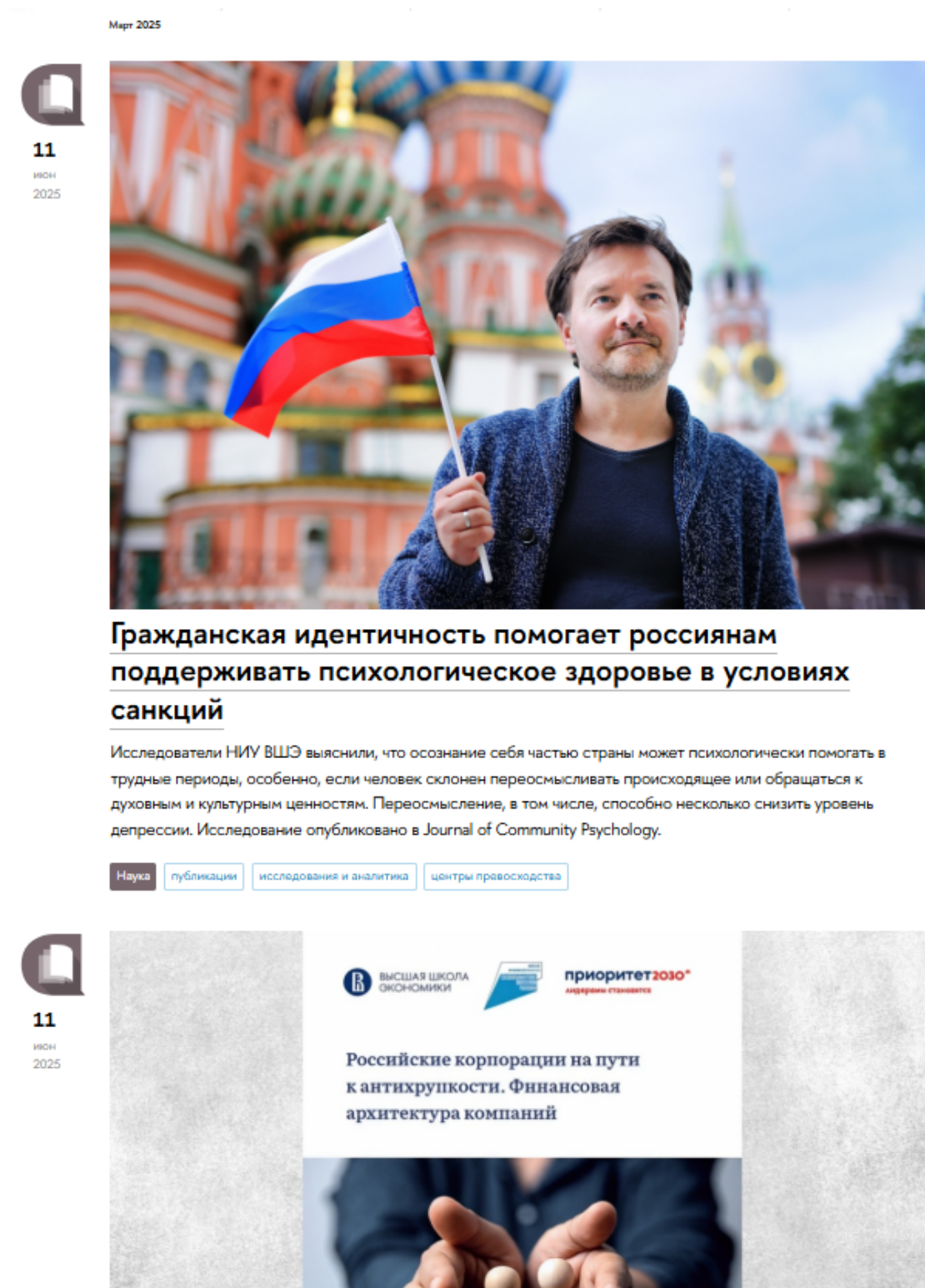


Рисунок 7 – Пример страницы новостного сайта ВШЭ

ПРИЛОЖЕНИЕ Б

Листинг вебскраппера

```
1 import requests
2 from bs4 import BeautifulSoup
3 import pandas as pd
4 import os
5 import time
6 import threading
7
8
9 def __loading_bar_and_info__(
10     start: bool, number_of_steps: int, total_steps: int,
11     number_of_thread: int
12 ) -> None:
13     '''Вывод информации о прогрессе выполнения программы.
14     start - нужно ли вывести начальную строку;
15     number_page - количество спаршенных страниц;
16     total_pages - всего страниц, которые нужно спарсить;
17     miss_count - число новостей, которые не удалось спарсить;
18     whitour_whole_content - число новостей, у которых не получило
19         сь полностью спарсить контент.'''
20     done = int(number_of_steps / total_steps * 100) if int(
21         number_of_steps / total_steps * 100
22     ) < 100 or number_of_steps == total_steps else 99
23     stars = int(
24         40 / 100 * done
25     ) if int(20 / 100 * done) < 20 or number_of_steps ==
26         total_steps else 39
27     tises = 40 - stars
28
29     if start:
30         stars = 0
31         tises = 40
32         done = 0
33
34     print("thread{0} <".format(number_of_thread), end="")
35     for i in range(stars):
36         print("*", end=" ")
37
38     for i in range(tises):
39         print("-", end=" ")
```



```

37     print("> {0}% ||| {1} / {2}".format(done, number_of_steps,
38                                           total_steps))
39
40 def __getPage__(url: str, file_name: str) -> None:
41     '''Получение html файла страницы.
42     url - ссылка на страницу;
43     file_name - имя файла, в который будет сохранена страница.'''
44     r = requests.get(url=url)
45
46     with open(file_name, "w", encoding="utf-8") as file:
47         file.write(r.text)
48
49
50 def __parse_news__(url: str) -> str:
51     '''Получение полного контента новости.
52     url - ссылка на новость.
53     Функция возвращает полный текст новости.'''
54     news_file_name = "news.html"
55     __getPage__(url, news_file_name)
56
57     with open(news_file_name, encoding="utf-8") as file:
58         src = file.read()
59
60     content = BeautifulSoup(src, "lxml").find("div",
61                                                class_="main").find(
62         "div", class_="post__text"
63     ).text.strip()
64
65     return content
66
67 def __parse_page__(page_file_name: str, news_container:
68                     pd.DataFrame) -> None:
69     '''Парсинг информации с новостной страницы: ссылка на новость
70     + короткая информация о ней.
71     page_file_name - имя файла, в который сохранён код страницы;
72     news_container - таблица, в которую заносится информация о но-
73     вости.
74     Функция также возвращает количество новостей, которые не удал-
75     ось спарсить

```

```

72     и количество новостей, полный контент которых спарсить не уда
        лось. '''
73     with open(page_file_name, encoding="utf-8") as file:
74         src = file.read()
75
76     soup = BeautifulSoup(src, "lxml")
77
78     news = soup.find("div", class_="post")
79     for i in range(10):
80         try:
81             news_day = news.find("div",
                                   class_="post-meta__day").text.strip()
82         except:
83             news_day = ""
84
85         try:
86             news_month = news.find("div",
87                                     class_="post-meta__month").text.strip()
88         except:
89             news_month = ""
90
91         try:
92             news_year = news.find("div",
93                                   class_="post-meta__year").text.strip()
94         except:
95             news_year = ""
96
97         news_date = news_day + "." + news_month + "." + news_year
98
99         try:
100             news_name = news.find("h2",
101                                    class_="first_child").find("a").text.st
102         except:
103             news_name = ""
104
105         try:
106             news_short_content = news.find("p",
107                                              class_="first_child"
108                                              ).find_next_sibling("p").text.st
109         except:
110             news_short_content = ""

```

```

109
110     try:
111         link = news.find("h2",
112                           class_="first_child").find("a").get("href")
113         if not link.startswith("https://"):
114             link = 'https://www.hse.ru' + link
115     except:
116         link = ""
117
118     try:
119         if link.startswith("https://www.hse.ru/news/"):
120             news_content = __parse_news__(link)
121     except:
122         news_content = ""
123
124     if len(
125         news_day + news_month + news_year + news_name +
126         news_short_content +
127         news_content
128     ) > 0:
129         news_container.loc[len(news_container.index)] = [
130             link, news_date, news_name, news_short_content,
131             news_content

```

Листинг 32: Полный код вебскраппера

ПРИЛОЖЕНИЕ В

Листинг комплексной обработки текста: нормализация, токенизация, лемматизация, фильтрация стоп-слов по словарю

```

1  def __processing_cell__(self, cell: str) -> str:
2      parts = self.__split_into_en_and_ru__(cell)
3      tokens = []
4      for part in parts:
5          if part[0]:
6              tokens += [
7                  self.__processing_token__(token.lemma_)
8                  for token in self.nlp_en(
9                      self.__remove_extra_spaces_and_line_breaks__(part[1])

```

```

10         ) if not (token.is_stop) and not
              (token.is_punct) and
11         len(self.__processing_token__(token.lemma_)) > 1
12     ]
13     else:
14         tokens += [
15             self.__processing_token__(token.lemma_)
16             for token in self.nlp_ru(
17                 self.__remove_extra_spaces_and_line_breaks__(part[1])
18             ) if not (token.is_stop) and not
                  (token.is_punct) and
19             len(self.__processing_token__(token.lemma_)) > 1
20         ]
21     return " ".join(tokens)

```

Листинг 33: Комплексная обработка текста: нормализация, токенизация, лемматизация, фильтрация стоп-слов по словарю

ПРИЛОЖЕНИЕ Г

Листинг обработчика новостного массива

```

1 class Text_preparer:
2     def __init__(self, additional_stop_words_path: str = ""):
3         '''Инициализация.\n
4         additional_stop_words: пользовательский список стоп
5         -слов.'''
6         self.nlp_en = spacy.load("en_core_web_sm")
7         self.nlp_ru = spacy.load("ru_core_news_sm")
8
9         self.tfidf_corpus = None
10        self.tfidf_dictionary = None
11
12    def __first_is_en__(self, cell: str) -> bool:
13        '''Определяет начинается строка с символа русского алфави
14        та или
15        английского алфавита.\n
16        cell: строка.\n
17        Возвращает true, если строка начинается с символа английс
18        кого алфавита.
19        '''
20        index_first_en = re.search(r"[a-zA-Z]", cell)
21        index_first_ru = re.search(r"[а-яА-Я]", cell)

```

```

20     return True if index_first_en and (
21         not (index_first_ru) or
22         index_first_en.start() < index_first_ru.start()
23     ) else False
24
25 def __split_into_en_and_ru__(self, cell: str) -> list[(bool,
26     str)]:
27     '''Разделяет строку на части, в которых содержатся символ
28     ы принадлежащие
29     только русскому или английскому алфавиту (то есть в строк
30     е с русскими
31     символами не будет символов английского языка и наоборот
32     , остальные символы
33     не удаляются).\n
34     cell: строка.\n
35     Возвращает массив кортежей
36     (True(если начинается с символа английского алфавита), по
37     дстрока).
38     '''
39     parts = []
40     is_en = self.__first_is_en__(cell)
41     part = ""
42     for symb in cell:
43         if is_en == (symb in string.ascii_letters) or not
44             (symb.isalpha()):
45             part += symb
46         else:
47             parts.append((is_en, part))
48             part = symb
49             is_en = not (is_en)
50
51     if part:
52         parts.append((is_en, part))
53
54     return parts
55
56 def __remove_extra_spaces_and_line_breaks__(self, text: str)
57     -> str:
58     '''Удаляет из строки лишние пробелы и переносы строки.\n
59     text: строка.\n
60     Возвращает строку, с удалёнными лишними пробелами и перенос

```

```

54         самм строк.
55     """
56     processed = ""
57     if type(text) != str or len(text) == 0:
58         return ""
59
60     flag = True
61     for symb in text:
62         if flag and (symb == " " or symb == "\n"):
63             processed += " "
64             flag = False
65
66         if symb != " " and symb != "\n":
67             flag = True
68
69         if flag:
70             processed += symb
71
72     return processed.strip()
73
74 def __count_letters_in_token__(self, token: str) -> int:
75     num_letters = 0
76
77     for symb in token:
78         if ("a" <= symb and symb <= "z") or ("A" <= symb and
79             symb <= "Z"):
80             num_letters += 1
81         if ("а" <= symb and symb <= "я") or ("А" <= symb and
82             symb <= "Я"):
83             num_letters += 1
84
85     return num_letters
86
87 def __strip_non_letters__(self, text: str) -> str:
88     return
89         re.sub(r"^[^a-zA-Za-яА-ЯёЁ]+|^[^a-zA-Za-яА-ЯёЁ]+$",
90             "", text)
91
92 def __processing_token__(self, token: str) -> str:
93     new_token = self.__strip_non_letters__(token)

```

```

90
91     return new_token if
92         (self.__count_letters_in_token__(new_token) +
93             1.0) / (len(new_token) + 1.0) >=
94             0.5 else ""
95
96
97 def __processing_cell__(self, cell: str) -> str:
98     '''Полностью обрабатывает 1 ячейку pandas DataFrame. То е
99     сть проводит
100     токенизацию, лемматизацию, удаление стоп слов и перевод в
101     нижний регистр,
102     потом происходит склейка и возвращается обработанная ячей
103     ка.\n
104     cell: строка - ячейка pandas DataFrame.\n
105     Возвращает обработанную строку.
106     '''
107     parts = self.__split_into_en_and_ru__(cell)
108
109     tokens = []
110
111     for part in parts:
112         if part[0]:
113             tokens += [
114                 self.__processing_token__(token.lemma_)
115                 for token in self.nlp_en(
116                     self.__remove_extra_spaces_and_line_breaks__(part
117 ) if not (token.is_stop) and not
118                     (token.is_punct) and
119                     len(self.__processing_token__(token.lemma_))
120                         > 1
121                 ]
122         else:
123             tokens += [
124                 self.__processing_token__(token.lemma_)
125                 for token in self.nlp_ru(
126                     self.__remove_extra_spaces_and_line_breaks__(part
127 ) if not (token.is_stop) and not
128                     (token.is_punct) and
129                     len(self.__processing_token__(token.lemma_))
130                         > 1
131                 ]

```

```

122
123         return " ".join(tokens)
124
125     def
126         __calc_tfidf_corpus_without_zero_score_tokens_and_tfidf_dictionary(
127         self
128         ) -> None:
129         '''Вычисление tfidf метрики для слов документов + tfidf с
130         лова. '''
131         texts = []
132         self.original_tokens = []
133
134         for row in range(self.p_data.shape[0]):
135             words = []
136             for column in self.processing_columns:
137                 for word in self.p_data.loc[row, column].split("
138                 "):
139                     words.append(word)
140             self.original_tokens.append(words)
141             texts.append(words)
142
143         dictionary = gensim.corpora.Dictionary(texts)
144         corpus = [dictionary.doc2bow(text) for text in texts]
145         tfidf = gensim.models.TfidfModel(corpus)
146
147         self.tfidf_corpus = tfidf[corpus]
148         self.tfidf_dictionary = dictionary
149
150     def __add_in_tfidf_corpus_zero_score_tokens__(self) -> None:
151         '''Добавление слов в tfidf_corpus, которые были исключены
152         gensim при
153         подсчёте метрики tfidf (gensim не добавляет слова, которы
154         е встречаются
155         во всех документах или которые имеют 0 метрику tfidf в
156         tfidf_corpus).'''
157         full_corpus = []
158
159         for doc_idx, doc in enumerate(self.tfidf_corpus):
160             original_words = self.original_tokens[doc_idx]
161             term_weights = {
162                 self.tfidf_dictionary.get(term_id): weight

```



```

157         for term_id, weight in doc
158     }
159
160     full_doc = []
161     for word in original_words:
162         if word in term_weights:
163             weight = term_weights[word]
164         else:
165             weight = 0.0
166         full_doc.append((word, weight))
167
168     full_corpus.append(full_doc)
169
170     self.tfidf_corpus = full_corpus
171
172     def __calc_threshold_for_tfidf_stop_words__(
173         self, tfidf_percent_treshold
174     ) -> None:
175         '''Вычисляет порог tfidf метрики, при котором слова, знач
176             ение tfidf
177             которых меньше, считаются стоп-словами.\n
178             tfidf_percent_threshold: процент от всех слов, которые бу
179             дут считаться
180             стоп-словами. То есть берём список всех значений tfidf, с
181             ортируем их и
182             значение, которое отсекает от остальной базы 1 процент са
183             мых низких
184             значений и будет threshold.'''
185         all_tfidf_values = []
186         for doc in self.tfidf_corpus:
187             for _, tfidf_value in doc:
188                 all_tfidf_values.append(tfidf_value)
189
190         self.threshold_for_tfidf_stop_words = np.percentile(
191             all_tfidf_values, tfidf_percent_treshold
192         )
193
194     def del_tfidf_stop_words(self, tfidf_percent_treshold) ->
195         None:
196         '''Удаляет стоп-слова на основе посчитанного
197             tfidf_corpus и

```

```

192         tfidf_threshold. '''
193     self.__calc_tfidf_corpus_without_zero_score_tokens_and_tfidf_dicti
194 )
195     self.__add_in_tfidf_corpus_zero_score_tokens__()
196     self.__calc_threshold_for_tfidf_stop_words__(tfidf_percent_tresho
197
198     for row, doc in zip(range(self.p_data.shape[0]),
199                         self.tfidf_corpus):
200         tfidf_stop_words = [
201             word for word, tfidf_value in doc
202             if tfidf_value <
203                 self.threshold_for_tfidf_stop_words
204         ]
205
206         for column in self.processing_columns:
207             words_without_tfidf_stop_words = []
208             for word in self.p_data.loc[row, column].split("
209                 "):
210                 if word in tfidf_stop_words:
211                     continue
212                 words_without_tfidf_stop_words.append(word)
213             self.p_data.loc[
214                 row, column] = "
215                 ".join(words_without_tfidf_stop_words)
216
217     def __calc_num_docs_for_words__(self) -> None:
218         self.num_docs_for_words = dict()
219
220         for row in range(self.p_data.shape[0]):
221             for column in self.processing_columns:
222                 words = self.p_data.loc[row, column].split(" ")
223
224                 for word in words:
225                     if word in self.num_docs_for_words.keys():
226                         self.num_docs_for_words[word] += 1
227                     else:
228                         self.num_docs_for_words[word] = 1
229
230     def __count_num_words__(self, doc: str) -> int:
231         return len(doc.split(" "))
232
233

```

```

229 def __del_docs_with_low_num_words__(self) -> None:
230     mask = self.p_data[self.processing_columns].apply(
231         lambda col: col.apply(self.__count_num_words__)
232     ).sum(axis=1)
233
234     self.p_data = self.p_data[mask >= 80]
235
236     self.p_data = self.p_data.reset_index(drop=True)
237
238 def __calc_up_and_down_threshold__(self):
239     self.up_threshold = self.p_data.shape[0] * (
240         len(self.processing_columns) / 2.0
241     )
242     self.down_threshold = self.p_data.shape[0] / 1000.0
243
244 def processing_data(
245     self,
246     standart_processing: bool = True,
247     tfidf_processing: bool = False,
248     tfidf_percent_treshold: int = 1
249 ) -> None:
250     '''
251     Функция, вызывающая вышеописанные функции для обработки
252     pandas DataFrame.\n
253     standart_processing: bool - говорит нужно ли делать станд
254     артную обработку;\n
255     tfidf_processing: bool - говорит нужно ли удалять стоп
256     -слова на основе
257     tfidf;\n
258     tfidf_percent_treshold: int - какой процент минимальных з
259     начений tfidf
260     отсеивать.'''
261     self.p_data = self.data.copy(deep=True)
262     self.p_data.fillna("", inplace=True)
263
264     if standart_processing:
265         for row in range(self.p_data.shape[0]):
266             for column in self.processing_columns:
267                 cell = self.p_data.loc[row, column]
268
269                 if len(cell) > 0:

```

```

266         self.p_data.loc[
267             row, column] =
268                 self.__processing_cell__(cell)
269
270     self.__del_docs_with_low_num_words__()
271     self.__calc_up_and_down_threshold__()
272     self.__calc_num_docs_for_words__()
273
274     for row in range(self.p_data.shape[0]):
275         for column in self.processing_columns:
276             words = self.p_data.loc[row, column].split("
277                 ")
278             new_words = []
279
280             for word in words:
281                 if self.num_docs_for_words[
282                     word
283                     ] >= self.down_threshold and
284                     self.num_docs_for_words[
285                         word] <= self.up_threshold:
286                     new_words.append(word)
287
288             self.p_data.loc[row, column] = "
289                 ".join(new_words)
290
291     self.__del_docs_with_low_num_words__()
292
293     if tfidf_processing:
294         self.p_data = self.data
295         self.p_data = self.p_data.fillna("")
296         self.p_data = self.p_data.astype(str)
297         self.del_tfidf_stop_words(tfidf_percent_treshold)
298         self.__del_docs_with_low_num_words__()
299
300     def add_data(self, data: pd.DataFrame) -> None:
301         self.data = data.copy(deep=True)
302
303     def get_data(self) -> pd.DataFrame:
304         return self.data
305
306     def add_processing_columns(self, processing_columns:

```

```

303         list[str])) -> None:
304             self.processing_columns = processing_columns
305
306     def get_processing_columns(self) -> list[str]:
307         return self.processing_columns
308
309     def get_processing_data(self) -> pd.DataFrame:
310         return self.p_data.copy(deep=True)
311
312     def save_processing_data(self, path: str) -> None:
313         self.p_data.to_excel(path, index=False)

```

Листинг 34: Полный код подготовки новостного массива

ПРИЛОЖЕНИЕ Д

Листинг функции вычисления тематической модели с пошаговым расчётом метрик

```

1  def calc_model(self):
2      self.perplexity_by_epoch = []
3      self.coherence_by_epoch = []
4      self.topic_purities_by_epoch = []
5
6      for epoch in range(self.num_collection_passes):
7          self.model.fit_offline(
8              batch_vectorizer=self.batches,
9              num_collection_passes=1
10         )
11         self.__calc_metrics__()
12         self.perplexity_by_epoch.append(self.perplexity)
13         self.coherence_by_epoch.append(self.coherence)
14         self.topic_purities_by_epoch.append(self.topic_purities)
15
16         if epoch > 0:
17             change_perplexity_by_percent = abs(
18                 self.perplexity_by_epoch[epoch - 1] -
19                 self.perplexity_by_epoch[epoch]
20             ) / (self.perplexity_by_epoch[epoch - 1] +
21                 self.epsilon) * 100
22             change_coherence_by_percent = \
23                 abs( self.coherence_by_epoch[epoch - 1] - \
24                     self.coherence_by_epoch[epoch] ) / \
25                 ( self.coherence_by_epoch[epoch - 1] + \

```

```

24         self.epsilon ) * 100
25     change_topics_purity_by_percent = \
26         abs( self.topic_purities_by_epoch[epoch - 1] - \
27             self.topic_purities_by_epoch[epoch] ) / \
28         ( self.topic_purities_by_epoch[epoch - 1] + \
29             self.epsilon ) * 100
30
31     if change_perplexity_by_percent < \
32         self.plateau_perplexity and \
33         change_coherence_by_percent < \
34         self.plateau_coherence and \
35         change_topics_purity_by_percent < \
36         self.plateau_topics_purity:
37         break

```

Листинг 35: Функция вычисления тематической модели с пошаговым расчётом метрик

ПРИЛОЖЕНИЕ Е

Полный код класса `My_BigARTM_model`

```

1  class My_BigARTM_model():
2      def __init__(
3          self,
4          data: pd.DataFrame = pd.DataFrame(),
5          num_topics: int = 1,
6          num_document_passes: int = 1,
7          class_ids: dict[str, float] = {"@default_class": 1.0},
8          num_processors: int = 8,
9          path_vw: str = "./vw.txt",
10         batch_size: int = 1000,
11         dir_batches: str = "./batches",
12         num_top_tokens: int = 10,
13         regularizers: dict[str, float] = {},
14         num_collection_passes: int = 1,
15         plateau_perplexity: float = 0.1,
16         plateau_coherence: float = 0.1,
17         plateau_topics_purity: float = 0.1,
18         epsilon: float = 0.0000001
19     ):
20         self.data = data.copy(deep=True)
21         self.num_topics = num_topics
22         self.num_document_passes = num_document_passes

```

```

23     self.class_ids = class_ids
24     self.num_processors = num_processors
25     self.path_vw = path_vw
26     self.batch_size = batch_size
27     self.dir_batches = dir_batches
28     self.num_top_tokens = num_top_tokens
29     self.user_regularizers = regularizers
30     self.num_collection_passes = num_collection_passes
31     self.epsilon = epsilon
32
33     self.perplexity_by_epoch = []
34     self.coherence_by_epoch = []
35     self.topic_purities_by_epoch = []
36
37     self.plateau_perplexity = plateau_perplexity
38     self.plateau_coherence = plateau_coherence
39     self.plateau_topics_purity = plateau_topics_purity
40
41     if data.empty:
42         print(
43             "Чтобы создать модель добавьте данные , на которых
44             будет строиться модель"
45         )
46     else:
47         self.__make_vowpal_wabbit__()
48         self.__make_batches__()
49         self.__make_model__()
50
51     if self.user_regularizers:
52         self.add_regularizers(self.user_regularizers)
53
54     def __make_vowpal_wabbit__(self) -> None:
55         f = open(self.path_vw, "w")
56
57         for row in range(self.data.shape[0]):
58             string = ""
59             for column in self.data.columns:
60                 string += str(self.data.loc[row, column]) + " "
61
62             f.write("doc_{0} ".format(row) + string.strip() +
63                 "\n")

```

```

62
63 def __make_batches__(self) -> None:
64     self.batches = artm.BatchVectorizer(
65         data_path=self.path_vw,
66         data_format="vowpal_wabbit",
67         batch_size=self.batch_size,
68         target_folder=self.dir_batches
69     )
70
71     self.dictionary = self.batches.dictionary
72
73 def __make_model__(self) -> None:
74     self.model = artm.ARTM(
75         cache_theta=True,
76         num_topics=self.num_topics,
77         num_document_passes=self.num_document_passes,
78         dictionary=self.dictionary,
79         class_ids=self.class_ids,
80         num_processors=8
81     )
82
83     self.__add_BigARTM_metrics__()
84
85 def __add_BigARTM_metrics__(self) -> None:
86     self.model.scores.add(
87         artm.PerplexityScore(name='perplexity',
88                               dictionary=self.dictionary)
89     )
90     self.model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_score'))
91     self.model.scores.add(
92         artm.SparsityThetaScore(name='sparsity_theta_score')
93     )
94     self.model.scores.add(
95         artm.TopTokensScore(
96             name="top_tokens", num_tokens=self.num_top_tokens
97         )
98     )
99
100 def __calc_coherence__(self) -> None:
101     topics = []
102     if "top_tokens" in self.model.score_tracker:

```



```

102         last_tokens =
            self.model.score_tracker["top_tokens"].last_tokens
103         topics = [last_tokens[topic] for topic in
                    last_tokens]

104
105         valid_topics = []
106         for topic in topics:
107             if isinstance(topic, list) and len(topic) > 0:
108                 valid_topics.append(topic)
109
110         if not valid_topics:
111             self.coherence = 0.0
112             return
113
114         texts = []
115         for row in range(self.data.shape[0]):
116             words = []
117             for column in self.data.columns:
118                 cell_content = self.data.loc[row, column]
119                 if isinstance(cell_content, str) and
                    cell_content.strip():
120                     words += cell_content.split()
121             if words:
122                 texts.append(words)
123
124         if not texts:
125             self.coherence = 0.0
126             return
127
128         try:
129             dictionary = Dictionary(texts)
130             coherence_model = CoherenceModel(
131                 topics=valid_topics,
132                 texts=texts,
133                 dictionary=dictionary,
134                 coherence="c_v"
135             )
136             self.coherence = coherence_model.get_coherence()
137         except Exception as e:
138             print(f"Ошибка при расчете когерентности: {e}")
139             self.coherence = 0.0

```

```

140
141 def __calc_phi__(self) -> None:
142     self.phi = np.sort(self.model.get_phi(), axis=0)[::-1, :]
143
144 def __calc_theta__(self) -> None:
145     self.theta = self.model.get_theta()
146
147 def __calc_topic_purity__(self, topic: int) -> None:
148     return np.sum(self.phi[:, topic]) / self.phi.shape[0]
149
150 def __calc_topics_purities__(self) -> None:
151     topics = range(self.phi.shape[1])
152     self.topic_purities = sum(
153         [self.__calc_topic_purity__(topic) for topic in
154          topics]
155     ) / len(topics)
156
157 def __calc_metrics__(self) -> None:
158     self.perplexity =
159         self.model.score_tracker['perplexity'].last_value
160     self.sparsity_phi_score =
161         self.model.score_tracker['sparsity_phi_score']
162         .last_value
163     self.sparsity_theta_score = self.model.score_tracker[
164         'sparsity_theta_score'].last_value
165     self.top_tokens =
166         self.model.score_tracker['top_tokens'].last_tokens
167     self.__calc_coherence__()
168     self.__calc_phi__()
169     self.__calc_topics_purities__()
170
171 def add_data(self, data: pd.DataFrame) -> None:
172     self.data = data
173
174     self.__make_vowpal_wabbit__()
175     self.__make_batches__()
176     self.__make_model__()
177
178 def add_regularizer(self, name: str, tau: float = 0.0) ->
179     None:
180     if name == "SmoothSparseThetaRegularizer":

```

```

176         self.model.regularizers.add(
177             artm.SmoothSparseThetaRegularizer(name=name,
178                 tau=tau)
179         )
180         self.user_regularizers[name] = tau
181     elif name == "SmoothSparsePhiRegularizer":
182         self.model.regularizers.add(
183             artm.SmoothSparsePhiRegularizer(name=name,
184                 tau=tau)
185         )
186         self.user_regularizers[name] = tau
187     elif name == "DecorrelatorPhiRegularizer":
188         self.model.regularizers.add(
189             artm.DecorrelatorPhiRegularizer(name=name,
190                 tau=tau)
191         )
192         self.user_regularizers[name] = tau
193     elif name == "LabelRegularizationPhiRegularizer":
194         self.model.regularizers.add(
195             artm.LabelRegularizationPhiRegularizer(name=name,
196                 tau=tau)
197         )
198         self.user_regularizers[name] = tau
199     elif name == "HierarchicalSparsityPhiRegularizer":
200         self.model.regularizers.add(
201             artm.HierarchicalSparsityPhiRegularizer(name=name,
202                 tau=tau)
203         )
204         self.user_regularizers[name] = tau
205     elif name == "TopicSelectionThetaRegularizer":
206         self.model.regularizers.add(
207             artm.TopicSelectionThetaRegularizer(name=name,
208                 tau=tau)
209         )
210         self.user_regularizers[name] = tau
211     elif name == "BitermsPhiRegularizer":
212         self.model.regularizers.add(
213             artm.BitermsPhiRegularizer(name=name, tau=tau)
214         )
215         self.user_regularizers[name] = tau
216     elif name == "BackgroundTopicsRegularizer":

```

```

211         self.model.regularizers.add(
212             artm.BackgroundTopicsRegularizer(name=name,
213                                             tau=tau)
214         )
215         self.user_regularizers[name] = tau
216     else:
217         print(
218             "Регуляризатора {0} нет! Проверьте корректность н
219             азвания!".
220             format(name)
221         )
222
223     def add_regularizers(self, regularizers: dict[str, float])
224         -> None:
225         for regularizer in regularizers:
226             self.add_regularizer(regularizer,
227                                 regularizers[regularizer])
228
229     def calc_model(self):
230         self.perplexity_by_epoch = []
231         self.coherence_by_epoch = []
232         self.topic_purities_by_epoch = []
233
234         for epoch in range(self.num_collection_passes):
235             self.model.fit_offline(
236                 batch_vectorizer=self.batches,
237                 num_collection_passes=1
238             )
239             self.__calc_metrics__()
240             self.perplexity_by_epoch.append(self.perplexity)
241             self.coherence_by_epoch.append(self.coherence)
242             self.topic_purities_by_epoch.append(self.topic_purities)
243
244             if epoch > 0:
245                 change_perplexity_by_percent = abs(
246                     self.perplexity_by_epoch[epoch - 1] -
247                     self.perplexity_by_epoch[epoch]
248                 ) / (self.perplexity_by_epoch[epoch - 1] +
249                     self.epsilon) * 100
250                 change_coherence_by_percent =
251                     abs(self.coherence_by_epoch[epoch - 1] -

```

```

245         self.coherence_by_epoch[epoch]) / \
                                                (self.coherence_by_epoch[epoch]
                                                  - 1] +
                                                  self.epsilon)
                                                  * 100
246     change_topics_purity_by_percent = abs(
247         self.topic_purities_by_epoch[epoch - 1] -
            self.topic_purities_by_epoch[epoch])
            / \
248
                                                    (self.topic_purities_by_epoch[epoch]
                                                      - 1] +
                                                      self.epsilon)
                                                      * 100
249
250     if change_perplexity_by_percent <
        self.plateau_perplexity and
        change_coherence_by_percent <
        self.plateau_coherence and
        change_topics_purity_by_percent <
        self.plateau_topics_purity:
251         break
252
253     def get_perplexity(self) -> float:
254         return self.perplexity
255
256     def get_perplexity_by_epochs(self) -> list[float]:
257         return self.perplexity_by_epoch
258
259     def print_perplexity_by_epochs(self) -> None:
260         plt.plot(
261             range(len(self.perplexity_by_epoch)),
262             self.perplexity_by_epoch,
263             label="perplexity"
264         )
265         plt.title("График перплексии")
266         plt.xlabel("Epoch")
267         plt.ylabel("Perplexity")
268         plt.legend()
269         plt.show()
270
271     def get_coherence(self) -> float:

```

```

272         return self.coherence
273
274     def get_coherence_by_epochs(self) -> list[float]:
275         return self.coherence_by_epoch
276
277     def print_coherence_by_epochs(self) -> None:
278         plt.plot(
279             range(len(self.coherence_by_epoch)),
280             self.coherence_by_epoch,
281             label="coherence"
282         )
283         plt.title("График когерентности")
284         plt.xlabel("Epoch")
285         plt.ylabel("Coherence")
286         plt.legend()
287         plt.show()
288
289     def get_topic_purities(self) -> float:
290         return self.topic_purities
291
292     def get_topic_purities_by_epochs(self) -> list[float]:
293         return self.topic_purities_by_epoch
294
295     def print_topic_purities_by_epochs(self) -> None:
296         plt.plot(
297             range(len(self.topic_purities_by_epoch)),
298             self.topic_purities_by_epoch,
299             label="topic purities"
300         )
301         plt.title("График чистоты тем")
302         plt.xlabel("Epoch")
303         plt.ylabel("Topics purity")
304         plt.legend()
305         plt.show()
306
307     def get_model(self):
308         return self.model
309
310     def save_model(self, dir_model: str =
311         ". / drive / MyDrive / model") -> None:

```

```
self.model.dump_artm_model(dir_model)
```

Листинг 36: Полный код класса My_BigRTM_model

ПРИЛОЖЕНИЕ Ж

Листинг функции выбора оптимальной конфигурации

```
1 def __select_best_trial__(self, study, weights):
2     params_and_metrics = [
3         (trial.params, trial.values) for trial in
4             study.best_trials
5     ]
6     metrics = np.array([item[1] for item in params_and_metrics])
7     scaled_metrics = np.zeros_like(metrics)
8     for i in range(metrics.shape[1]):
9         scaler = RobustScaler()
10        scaled_column = scaler.fit_transform(metrics[:,
11            i].reshape(-1, 1)
12            ).flatten()
13        if weights[i] < 0:
14            scaled_column = -scaled_column
15        scaled_metrics[:, i] = scaled_column
16    scaled_params_and_metrics = [
17        (item[0], item[1], scaled_metrics[i].tolist())
18        for i, item in enumerate(params_and_metrics)
19    ]
20    return min(scaled_params_and_metrics, key=lambda trial:
21        sum(trial[2]))
```

Листинг 37: Функция выбора оптимальной конфигурации

ПРИЛОЖЕНИЕ З

Полный код класса Hyperparameter_optimizer

```
1 class Hyperparameter_optimizer:
2     def __init__(
3         self,
4         data: pd.DataFrame,
5         n_trials: int = 50,
6         num_topics: tuple[str, int, int] = ("num_topics", 6, 8),
7         num_document_passes: tuple[str, int,
8             int] =
9             ("num_document_passes",
10                3, 7),
```

```

9         num_collection_passes: tuple[str, int,
10                                     int] =
11                                     ("num_collection_passes",
12                                     3, 7),
13
14     regularizers: dict[str, tuple[str, float, float]] = {
15         "SmoothSparseThetaRegularizer": ('tau_theta', -2.0,
16         2.0),
17         "SmoothSparsePhiRegularizer": ('tau_phi', -2.0, 2.0)
18     },
19     class_ids: dict[str, float] = {"@default_class": 1.0}
20 ):
21     self.data = data.copy(deep=True)
22     self.n_trials = n_trials
23     self.num_topics = num_topics
24     self.num_document_passes = num_document_passes
25     self.num_collection_passes = num_collection_passes
26     self.regularizers = regularizers
27     self.class_ids = class_ids
28
29     self.robust_scaler = RobustScaler()
30
31     def __generate_regularizers_dict__(self, trial) -> dict[str,
32                                     float]:
33         """Генерирует словарь с параметрами регуляризаторов для т
34             екущего trial"""
35         reg_dict = {}
36         for reg_name, (param_name, low, high) in
37             self.regularizers.items():
38             tau_value = trial.suggest_float(param_name, low,
39             high)
40             reg_dict[reg_name] = tau_value
41         return reg_dict
42
43     def __objective__(self, trial) -> tuple[float, float, float]:
44         # Основные параметры модели
45         num_topics = trial.suggest_int(
46             self.num_topics[0], self.num_topics[1],
47             self.num_topics[2]
48         )
49         num_document_passes = trial.suggest_int(
50             self.num_document_passes[0],

```



```

42         self.num_document_passes[1],
43         self.num_document_passes[2]
44     )
45     num_collection_passes = trial.suggest_int(
46         self.num_collection_passes[0],
47         self.num_collection_passes[1],
48         self.num_collection_passes[2]
49     )
50     # Динамическое создание параметров регуляризаторов
51     regularizers_dict =
52         self.__generate_regularizers_dict__(trial)
53     class_ids = self.class_ids
54
55     # Создание и расчет модели
56     model = My_BigARTM_model(
57         data=self.data,
58         num_topics=num_topics,
59         num_document_passes=num_document_passes,
60         class_ids=class_ids,
61         num_collection_passes=num_collection_passes,
62         regularizers=regularizers_dict
63     )
64     model.calc_model()
65
66     return model.get_perplexity(), model.get_coherence(
67     ), model.get_topic_purities()
68
69 def __extract_regularizers_params__(self, params: dict) ->
70     dict[str, float]:
71     """Извлекает параметры регуляризаторов из общего словаря
72     параметров"""
73     reg_params = {}
74     for reg_name, (param_name, _, _) in
75         self.regularizers.items():
76         if param_name in params:
77             reg_params[reg_name] = params[param_name]
78     return reg_params
79
80 def __select_best_trial__(self, study, weights):
81     """Выбирает trial с минимальной взвешенной суммой метрик

```

```

77         """
78         params_and_metrics = [
79             (trial.params, trial.values) for trial in
80             study.best_trials
81         ]
82         metrics = np.array([item[1] for item in
83             params_and_metrics])
84
85         scaled_metrics = np.zeros_like(metrics)
86         for i in range(metrics.shape[1]):
87             scaler = RobustScaler()
88             scaled_column = scaler.fit_transform(metrics[:,
89                 i].reshape(-1, 1)
90                 ).flatten()
91             if weights[i] < 0:
92                 scaled_column = -scaled_column
93             scaled_metrics[:, i] = scaled_column
94
95         scaled_params_and_metrics = [
96             (item[0], item[1], scaled_metrics[i].tolist())
97             for i, item in enumerate(params_and_metrics)
98         ]
99
100         return min(scaled_params_and_metrics, key=lambda trial:
101             sum(trial[2]))
102
103     def optimizer(self):
104         study = optuna.create_study(
105             directions=["minimize", "maximize", "maximize"]
106         )
107         study.optimize(self.__objective__,
108             n_trials=self.n_trials)
109         best_trial = self.__select_best_trial__(study,
110             weights=[1, -1, -1])
111         best_params = best_trial[0]
112
113         # Извлечение основных параметров
114         num_topics = best_params[self.num_topics[0]]
115         num_document_passes =
116             best_params[self.num_document_passes[0]]
117         num_collection_passes =

```

```

best_params[self.num_collection_passes[0]]
110
111 # Извлечение параметров регуляризаторов
112 regularizers_params =
113     self.__extract_regularizers_params__(best_params)
114
115 print("Лучшие параметры:")
116 print(f"Количество тем: {num_topics}")
117 print(f"Проходы по документу: {num_document_passes}")
118 print(f"Проходы по коллекции: {num_collection_passes}")
119
120 for reg_name, tau_value in regularizers_params.items():
121     print(f"{reg_name}: {tau_value:.4f}")
122
123 # Создание финальной модели
124 final_model = My_BigARTM_model(
125     data=self.data,
126     num_topics=num_topics,
127     num_document_passes=num_document_passes,
128     num_collection_passes=num_collection_passes,
129     regularizers=regularizers_params,
130     class_ids=self.class_ids
131 )
132 final_model.calc_model()
133 self.model = final_model
134
135 # Остальные методы без изменений
136 def get_model(self) -> My_BigARTM_model:
137     return self.model
138
139 def save_model(self, path_model: str =
140     ". / drive / MyDrive / model") -> None:
141     self.model.model.dump_artm_model(path_model)
142
143 def save_phi(self, path_phi: str =
144     ". / drive / MyDrive / phi.xlsx") -> None:
145     self.model.model.get_phi().to_excel(path_phi)
146
147 def save_theta(
148     self, path_theta: str = ". / drive / MyDrive / theta.xlsx "
149 ) -> None:

```

```
self.model.model.get_theta().T.to_excel(path_theta)
```

Листинг 38: Полный код класса Hyperparameter_optimizer

ПРИЛОЖЕНИЕ И

Количественные характеристики подготовленного и неподготовленного новостного массива

Характеристика	Неподгот.	Стоп-слова	+Низкочаст.	TF-IDF 1%	TF-IDF 2%	TF-IDF 3%
Кол. док.	17340	17340	17340	17340	17340	17340
Кол. токенов	1213111	16545045	-	6479545	6414045	6348544
Кол. уник. ток.	278724	148677	-	148677	148677	148677
Мин. кол. ток. в док.	6	4	-	4	4	4
Модальное кол. ток. в док.	47	31	-	31	31	30
Среднее кол. ток. в док.	695	375	-	371	367	364
Медианное кол. ток. в док.	-	313	-	312	310	309
Макс. кол. ток. в док.	6514	3151	-	2903	2825	2766
Мин. кол. уник. ток. в док.	6	4	-	4	4	4

Продолжение следует...

Продолжение таблицы

Характеристика	Неподгот.	Стоп-слова	+Низкочаст.	TF-IDF 1%	TF-IDF 2%	TF-IDF 3%
Мод. кол. уник. ток. в док.	39	27	-	27	27	30
Сред. кол. уник. ток. в док.	346	214	-	211	208	205
Мед. кол. уник. ток. в док.	-	186	-	185	183	182
Макс. кол. уник. ток. в док.	2287	1353	-	1299	1262	1214

Характеристика	TF-IDF 4%	TF-IDF 5%	TF-IDF 6%.	TF-IDF 7%	TF-IDF 8%	TF-IDF 9%
Кол. док.	17340	17340	17340	17340	17340	17340
Кол. токенов	6283046	6217544	6152044	6086544	6021044	5955543
Кол. уник. ток.	148677	148677	148677	148677	148677	148677
Мин. кол. ток. в док.	4	4	4	4	4	4

Продолжение следует...

Продолжение таблицы

Характеристика	TF-IDF 4%	TF-IDF 5%	TF-IDF 6%	TF-IDF 7%	TF-IDF 8%	TF-IDF 9%
Модальное кол. ток. в док.	30	30	30	30	29	29
Среднее кол. ток. в док.	360	356	352	349	345	341
Медианное кол. ток. в док.	307	306	305	303	301	299
Макс. кол. ток. в док.	2713	2662	2595	2545	2501	2424
Мин. кол. уник. ток. в док.	4	4	4	4	4	4
Мод. кол. уник. ток. в док.	27	29	29	28	28	28
Сред. кол. уник. ток. в док.	201	198	195	192	189	186
Мед. кол. уник. ток. в док.	181	179	177	176	174	172
Макс. кол. уник. ток. в док.	1164	1122	1085	1047	1018	986

Характеристика	TF-IDF 10%	TF-IDF 10% + Низк.
Кол. док.	17340	17340
Кол. токенов	5890042	-
Кол. уник. ток.	148677	-
Мин. кол. ток. в док.	4	-
Модальное кол. ток. в док.	30	-
Среднее кол. ток. в док.	337	-
Медианное кол. ток. в док.	297	-
Макс. кол. ток. в док.	2391	-
Мин. кол. уник. ток. в док.	4	-
Мод. кол. уник. ток. в док.	28	-
Сред. кол. уник. ток. в док.	182	-

Продолжение следует...

Продолжение таблицы

Характеристика	TF-IDF 10%	TF-IDF 10% + Низк.
Мед. кол. уник. ток. в док.	170	-
Макс. кол. уник. ток. в док.	946	-

ПРИЛОЖЕНИЕ К

Полный код обучения модели классификатора

```
1 class TopicClassifier:
2     def __init__(
3         self,
4         data_path: str,
5         columns: List[str],
6         maximum_sequence_length: int = 200,
7         output_dir: str = "./model"
8     ):
9         try:
10             self.data = pd.read_excel(data_path)
11         except FileNotFoundError:
12             raise ValueError(f"File {data_path} not found!")
13
14         self.model_name = "nikitast/multilang-classifier-roberta"
15         self.columns = columns
16         self.maximum_sequence_length = maximum_sequence_length
17         self.output_dir = output_dir
18         self.device = torch.device("cuda" if
19                                     torch.cuda.is_available() else "cpu")
20
21         self.topic2id: Dict[str, int] = {}
22         self.id2topic: Dict[int, str] = {}
23         self.num_labels: int = 0
24         self.tokenizer = None
25         self.model = None
26         self.trainer = None
27         self.evaluation_results: Dict[str, float] = {}
28
29     def __prepare_data__(self):
30         self.data['text'] = self.data[self.columns].apply(
31             lambda x: ' '.join(x.dropna().astype(str)), axis=1
32         )
33
34         unique_topics = self.data['topic'].unique()
35         self.topic2id = {topic: i for i, topic in
36                         enumerate(unique_topics)}
37         self.id2topic = {i: topic for i, topic in
38                         enumerate(unique_topics)}
```

```

37     self.num_labels = len(self.topic2id)
38     if self.num_labels < 2:
39         raise ValueError("At least 2 classes required for
                                classification")
40
41     self.data['label'] =
42         self.data['topic'].map(self.topic2id)
43
44     def __load_model__(self):
45         self.tokenizer =
46             AutoTokenizer.from_pretrained(self.model_name)
47         self.model =
48             AutoModelForSequenceClassification.from_pretrained(
49                 self.model_name,
50                 num_labels=self.num_labels,
51                 problem_type="single_label_classification",
52                 ignore_mismatched_sizes=True
53             ).to(self.device)
54
55     def __tokenize_data__(self, df: pd.DataFrame) -> Dataset:
56         dataset = Dataset.from_pandas(df[['text', 'label']])
57
58         def tokenize_function(examples):
59             return self.tokenizer(
60                 examples["text"],
61                 padding="max_length",
62                 truncation=True,
63                 max_length=self.maximum_sequence_length
64             )
65
66         return dataset.map(tokenize_function, batched=True)
67
68     def __compute_metrics__(self, eval_pred) -> Dict[str, float]:
69         accuracy_metric = evaluate.load("accuracy")
70         logits, labels = eval_pred
71         predictions = np.argmax(logits, axis=-1)
72
73         metrics = {
74             "accuracy":
75                 accuracy_metric.compute(predictions=predictions,
76                                         references=labels)["accuracy"],

```

```

72         "f1_micro": f1_score(labels, predictions,
73                               average="micro"),
74         "f1_macro": f1_score(labels, predictions,
75                               average="macro"),
76         "f1_weighted": f1_score(labels, predictions,
77                                   average="weighted"),
78     }
79
80     try:
81         if logits.shape[1] == 2:
82             metrics["roc_auc"] = roc_auc_score(labels,
83                                                  logits[:, 1])
84         else:
85             metrics["roc_auc"] = roc_auc_score(
86                 labels, logits, multi_class="ovo",
87                 average="macro"
88             )
89     except ValueError:
90         metrics["roc_auc"] = float("nan")
91
92     return metrics
93
94 def __print_final_metrics__(self):
95     if not self.evaluation_results:
96         raise ValueError("Model not evaluated yet. Call
97                             train_model() first")
98
99     print("\n" + "="*50)
100    print("Final Model Evaluation Metrics:")
101    print("-"*50)
102    for metric, value in self.evaluation_results.items():
103        if metric not in ["eval_loss", "epoch"]:
104            print(f"{metric.upper():<15}: {value:.4f}")
105    print("="*50 + "\n")
106
107 def train_model(self):
108     self.__prepare_data__()
109     train_df, val_df = train_test_split(
110         self.data,
111         test_size=0.2,
112         random_state=42,

```

```

107         stratify=self.data['topic']
108     )
109
110     self.__load_model__()
111
112     train_dataset = self.__tokenize_data__(train_df)
113     val_dataset = self.__tokenize_data__(val_df)
114
115     training_args = TrainingArguments(
116         output_dir=self.output_dir,
117         eval_strategy="epoch",
118         save_strategy="epoch",
119         learning_rate=2e-5,
120         lr_scheduler_type="linear",
121         warmup_steps=100,
122         per_device_train_batch_size=32,
123         per_device_eval_batch_size=32,
124         num_train_epochs=10,
125         weight_decay=0.01,
126         load_best_model_at_end=True,
127         metric_for_best_model="accuracy",
128         logging_dir='./logs',
129         logging_steps=10,
130         report_to="none",
131         save_total_limit=1
132     )
133
134     self.trainer = Trainer(
135         model=self.model,
136         args=training_args,
137         train_dataset=train_dataset,
138         eval_dataset=val_dataset,
139         compute_metrics=self.__compute_metrics__,
140         callbacks=[EarlyStoppingCallback(early_stopping_patience=3)]
141     )
142
143     self.trainer.train()
144
145     self.evaluation_results = self.trainer.evaluate()
146     self.__print_final_metrics__()
147

```

```

148     # self.model.save_pretrained(self.output_dir)
149     # self.tokenizer.save_pretrained(self.output_dir)
150
151     # with open(f"{self.output_dir}/id2topic.json", "w") as
152         f:
153         #     json.dump({str(k): v for k, v in
154             self.id2topic.items()}, f)
155
156 def load_trained_model(self, model_path: str):
157     self.tokenizer =
158         AutoTokenizer.from_pretrained(model_path)
159     self.model =
160         AutoModelForSequenceClassification.from_pretrained(model_path)
161
162     with open(f"{model_path}/id2topic.json", "r") as f:
163         self.id2topic = {int(k): v for k, v in
164             json.load(f).items()}
165
166 def predict(self, text: str) -> str:
167     self.model.eval()
168     inputs = self.tokenizer(
169         text,
170         return_tensors="pt",
171         truncation=True,
172         max_length=self.maximum_sequence_length
173     ).to(self.device)
174
175     with torch.no_grad():
176         logits = self.model(**inputs).logits
177
178     predicted_id = torch.argmax(logits, dim=-1).item()
179     return self.id2topic[predicted_id]

```

Листинг 39: Полный код обучения модели классификатора