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

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 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 изученни, 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.

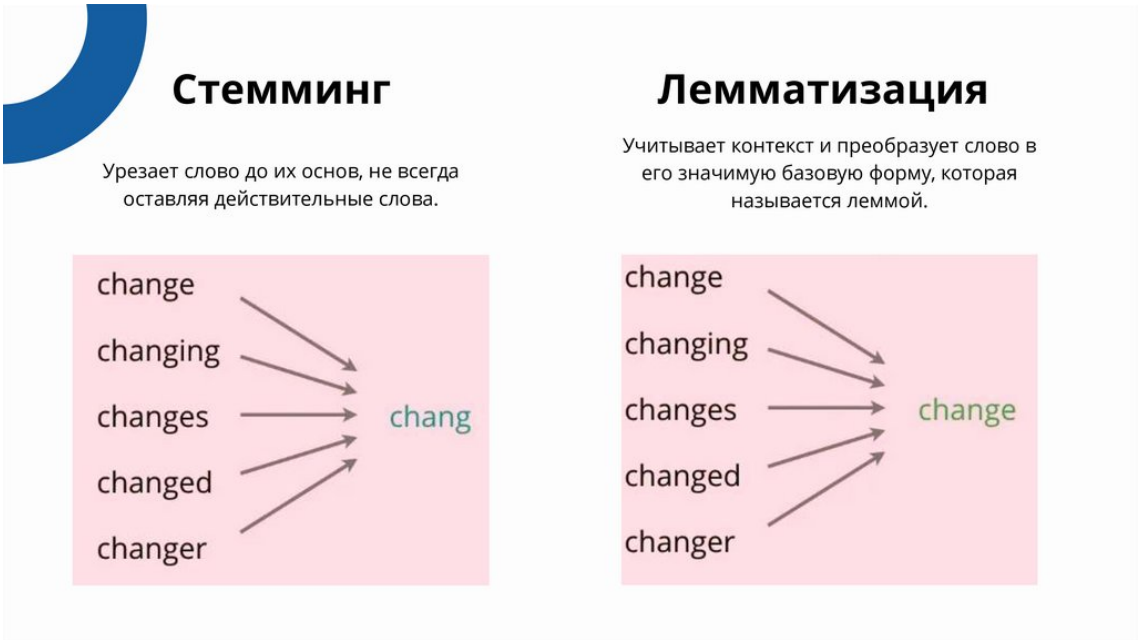

Рисунок 1 – Иллюстрация разницы между стеммингом и лемматизацией

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.ическую классификацию текстов на основе частоты и взаимовстречаемости слов.

1.4.2 Аксиоматика тематического моделирования

Каждый текст можно количественно охарактеризовать. Ниже приведены основные количественные характеристики, использующиеся 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 терма 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 терма w в документе d;
— $n_W = \sum_d n_{dw}$ — частота терма 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 терма 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 $p(w|d,t)$.

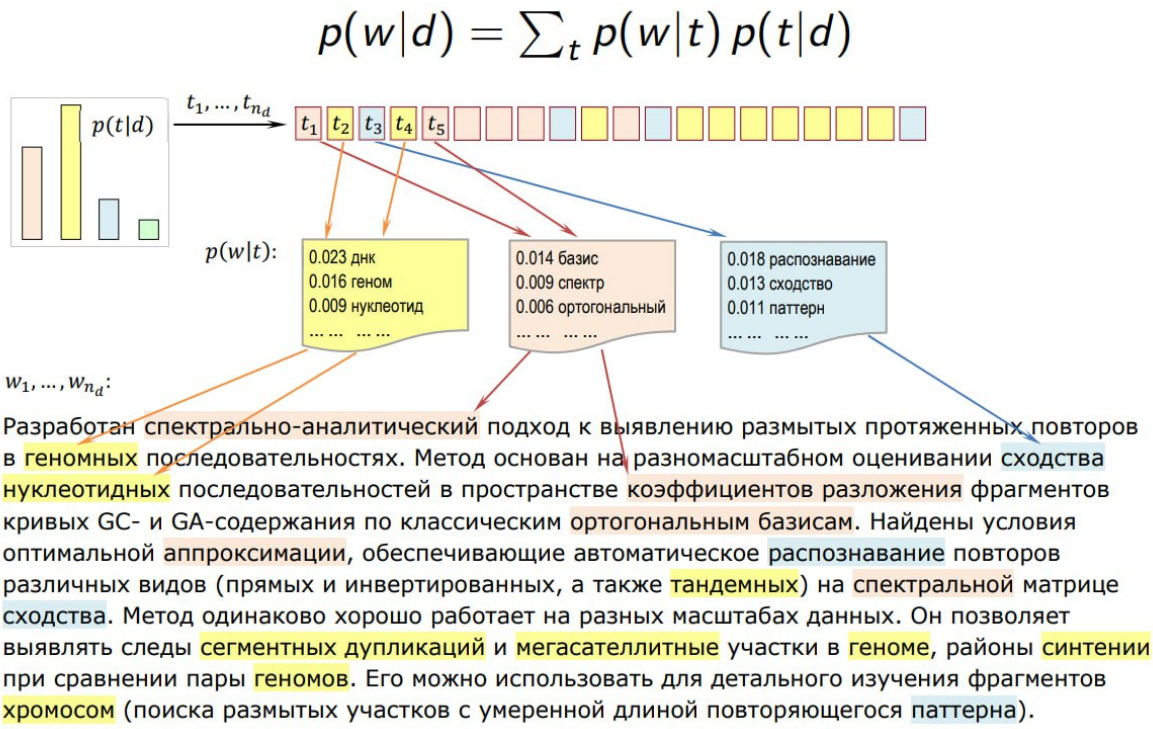

Рисунок 2 – Алгоритм формирования документа

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

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

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

1. для каждого терма 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.

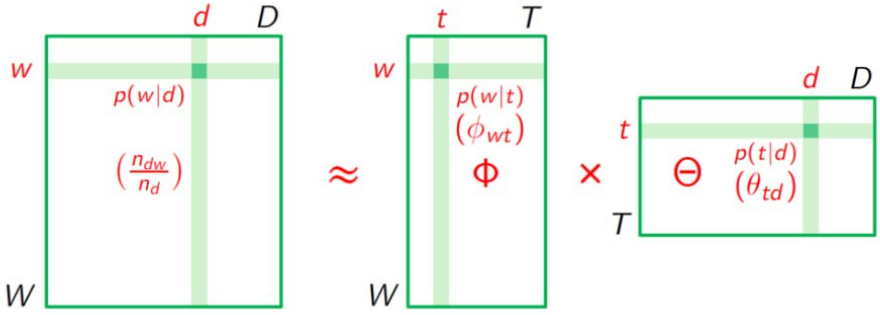

Рисунок 3 – Стохастическое матричное разложение

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:

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

Лемма о максимизации функции на единичных симплексах [3, 6]:

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

локального экстремума задачи

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

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

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

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

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

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

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

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.

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

Прологарифмировав правдоподобие, перейдём к задаче максимизации логарифма правдоподобия.

$$\sum_{d \in D} \sum_{w \in d} n_{dw} \ln p(w|d) \underbrace{p(d)}_{const} \to \max_{\Phi, \Theta} \tag{7}$$

Данная задача эквивалентна задаче максимизации функционала

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

при ограничениях неотрицательности и нормировки

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

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 $\tau_i, \quad i = 1, \ldots, k$.

Преобразуем задачу к ARTM виду:

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

при ограничениях неотрицательности и нормировки 9.

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

E-M 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 $p_{twd} = p(t|d, w)$, если из решения исключить нулевые столбцы матриц Φ и Θ:

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

Полученная модель соответствует E-M алгоритму, где первая строка системы уравнений соответствует E-шагу, а вторая и третья строки — M-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 оценитьваться 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:

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

Свойства KL-дивергенции:

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

2. $KL(P||Q) = 0 \quad \Leftrightarrow \quad P = Q$;
3. Минимизация KL эквивалентна максимизации правдоподобия:

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

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 сематническое 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-дивергеннции 1.4.5 выразим задачу сглаживания:

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

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

$$R(\Phi, \Theta) = \beta_o \sum_{t \in T} \sum_{w \in W} \beta_w \ln \phi_{wt} + \alpha_0 \sum_{d \in D} \sum_{t \in T} \alpha_t \ln \theta_{td} \to \max. \tag{14}$$

Перепишем EM-алгоритм 11 в соответствии с полученной формулой:

$$\begin{cases} p_{tdw} = \underset{t \in T}{norm}(\phi_{wt}\theta_{td}) \\ \phi_{wt} = \underset{w \in W}{norm}\left(n_{wt} + \beta_0\beta_w\right); \\ \theta_{td} = \underset{t \in T}{norm}\left(n_{td} + \alpha_0\alpha_t\right) \end{cases} \tag{15}$$

Таким образом был получен модифицированный EM-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 регуялризатор 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-дивергеннции 1.4.5 выразим задачу разреживания:

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

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

$$R(\Phi,\Theta) = -\beta_o \sum_{t\in T}\sum_{w\in W} \beta_w \ln \phi_{wt} - \alpha_0 \sum_{d\in D}\sum_{t\in T} \alpha_t \ln \theta_{td} \to \max. \tag{17}$$

Перепишем EM-алгоритм 11 в соответствии с полученной формулой:

$$\begin{cases} p_{tdw} = \underset{t\in T}{norm}(\phi_{wt}\theta_{td}) \\ \phi_{wt} = \underset{w\in W}{norm}\left(n_{wt} - \beta_0\beta_w\right); \\ \theta_{td} = \underset{t\in T}{norm}\left(n_{td} - \alpha_0\alpha_t\right) \end{cases} \tag{18}$$

Таким образом был получен модифицированный EM-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:

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

Перепишем EM-алгоритм 11 в соответствии с полученной формулой:

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

Таким образом был получен модифицированный EM-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].

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

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:

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

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

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

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 вероятностно относится ядро темы к фоновым словам темы, следовательно, чем больше вероятность ядра, тем лучше [8].

Контрастность темы: Контрастность темы:

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

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:

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

   где N — размер словаря.
3. Предложение представляется суммой векторов слов:

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

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].

Эмбеддинги 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 эмбеддинги;

— Матрица эмбеддингов — 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 эмбеддингов 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 принадлжности 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]:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{27}$$

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]:

$$Precision = \frac{TP}{TP + FP} \tag{28}$$

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

Recall Recall (полнота) определяется формулой [13]:

$$Recall = \frac{TP}{TP + FN} \tag{29}$$

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

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

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{30}$$

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:

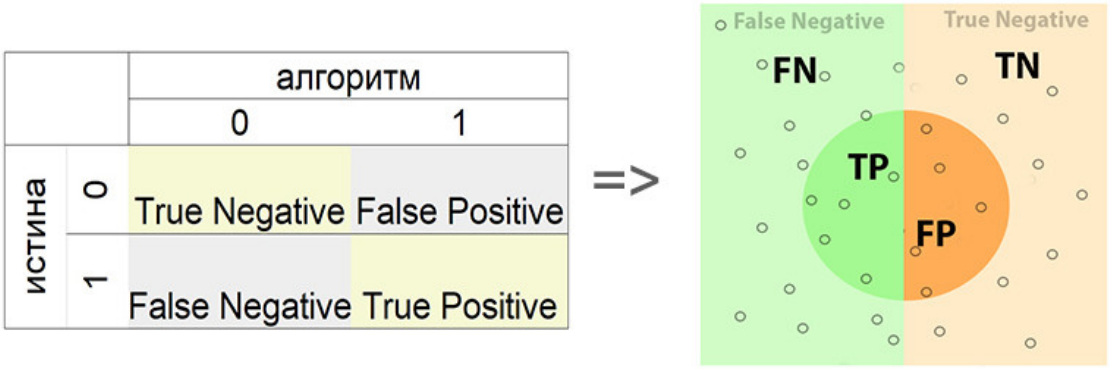

Рисунок 4 – Пример матрицы ошибок

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-скрапинга 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.

```
def __getPage__(url: str, file_name: str) -> None:
    r = requests.get(url=url)
    with open(file_name, "w", encoding="utf-8") as file:
        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()).

```
with open(page_file_name, encoding="utf-8") as file:
        src = file.read()
soup = BeautifulSoup(src, "lxml")
news = soup.find("div", class_="post")
try:
    link = news.find("h2",
            class_="first_child").find("a").get("href")
    if not link.startswith("https://"):
        link = 'https://www.hse.ru' + link
except:
    link = ""
try:
    news_short_content = news.find("p",
            class_="first_child").find_next_sibling("p").text.strip()
except:
    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.

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

```
            class_="main").find(
            "div", class_="post__text"
        ).text.strip()
        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.

```
def __parse_page__(page_file_name: str, news_container:
    pd.DataFrame) -> None:
    for i in range(10):
        try:
            if link.startswith("https://www.hse.ru/news/"):
                news_content = __parse_news__(link)
        except:
            news_content = ""
        if len(
            news_day + news_month + news_year + news_name +
                news_short_content +
            news_content
        ) > 0:
            news_container.loc[len(news_container.index)] =
                [
                link, news_date, news_name,
                    news_short_content, news_content]
        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.

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

```
            __getPage__("https://www.hse.ru/news/page{0}.html".forma
                    page_file_name)
            __parse_page__(page_file_name, news_container)
        except:
            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.

```
def crawling_pages(off_pc: bool, pages: int) -> None:
    columns = ["url", "date", "title", "summary", "content"]
    news_container1 = pd.DataFrame(columns=columns)
    news_container2 = pd.DataFrame(columns=columns)
    thread1 = threading.Thread(target=__crawling_pages__,
            args=(0, pages // 2, news_container1, 1))
    thread2 = threading.Thread(target=__crawling_pages__,
            args=(pages // 2, pages, news_container2, 2))
    thread1.start()
    thread2.start()
    thread1.join()
    thread2.join()
    try:
        news = pd.concat([news_container1,
                news_container2], ignore_index=True)
        news.to_excel("./news.xlsx")
    except:
        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, Pymorphy3, SpaCy и Gensim [19].

Сделаем выбор между связкой NLTK + Pymorphy3 и SpaCy. Обе группы

библиотек позволяют проводить лемматизацию и удаление стоп-слов, но реализуют это по-разному. NLTK и Pymorphy3 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]:

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

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 билиотеки 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]:

```
def __remove_extra_spaces_and_line_breaks__(self, text: str) ->
    str:
    processed = ""
    if type(text) != str or len(text) == 0:
        return ""
    flag = True
    for symb in text:
        if flag and (symb == " " or symb == "\n"):
            processed += " "
            flag = False
        if symb != " " and symb != "\n":
            flag = True
        if flag:
            processed += symb
    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]:

```
def __first_is_en__(self, cell: str) -> bool:
    index_first_en = re.search(r"[a-zA-Z]", cell)
    index_first_ru = re.search(r"[а-яА-Я]", cell)
    return True if index_first_en and ( not (index_first_ru) or
        index_first_en.start() < index_first_ru.start() ) else
            False
def __split_into_en_and_ru__(self, cell: str) -> list[(bool,
    str)]:
    parts = []
    is_en = self.__first_is_en__(cell)
    part = ""
    for symb in cell:
        if is_en == (symb in string.ascii_letters) or not
            (symb.isalpha()):
            part += symb
        else:
            parts.append((is_en, part))
            part = symb
            is_en = not (is_en)
    if part:
        parts.append((is_en, part))
    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].

```
def __count_letters_in_token__(self, token: str) -> int:
    num_letters = 0
    for symb in token:
        if ("a" <= symb and symb <= "z") or ("A" <= symb and
                symb <= "Z"):
            num_letters += 1
        if ("a" <= symb and symb <= "я") or ("A" <= symb and
                symb <= "Я"):
            num_letters += 1
    return num_letters
def __strip_non_letters__(self, text: str) -> str:
    return re.sub(r"^[^a-zA-Za-яA-ЯёЁ]+|[^a-zA-Za-яA-ЯёЁ]+$",
            "", text)
def __processing_token__(self, token: str) -> str:
    new_token = self.__strip_non_letters__(token)
    return new_token if
            (self.__count_letters_in_token__(new_token) +
                                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].

```
self.nlp_ru = spacy.load("ru_core_news_sm")
parts = self.__split_into_en_and_ru__(cell)
if part[1]:
    tokens += [
        self.__processing_token__(token.lemma_)
        for token in self.nlp_ru(
            self.__remove_extra_spaces_and_line_breaks__(part[1])
        ) if not (token.is_stop) and not (token.is_punct) and
        len(self.__processing_token__(token.lemma_)) > 1
    ]
```

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].

```
def __calc_num_docs_for_words__(self) -> None:
    self.num_docs_for_words = dict()
    for row in range(self.p_data.shape[0]):
        for column in self.processing_columns:
            words = self.p_data.loc[row, column].split(" ")
            for word in words:
                if word in self.num_docs_for_words.keys():
                    self.num_docs_for_words[word] += 1
                else:
                    self.num_docs_for_words[word] = 1
def __calc_up_and_down_threshold__(self):
    self.up_threshold = self.p_data.shape[0] * (
        len(self.processing_columns) / 2.0)
    self.down_threshold = self.p_data.shape[0] / 1000.0
for word in words:
    if self.num_docs_for_words[word] >= self.down_threshold and \
    self.num_docs_for_words[word] <= self.up_threshold:
```

```
18                 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]:

```
def add_in_tfidf_corpus_zero_score_tokens(self) -> None:
    full_corpus = []
    for doc_idx, doc in enumerate(self.tfidf_corpus):
        original_words = self.original_tokens[doc_idx]
        term_weights = {
            self.tfidf_dictionary.get(term_id): weight
            for term_id, weight in doc
        }
        full_doc = []
        for word in original_words:
            if word in term_weights:
                weight = term_weights[word]
            else:
                weight = 0.0
            full_doc.append((word, weight))
        full_corpus.append(full_doc)
    self.tfidf_corpus = full_corpus
```

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

3. Определение порога отсечения:

   а) Вычисление n-го процентиля распределения TF-IDF;

   б) Установка границы для отбора малозначимых терминов;

   Логика расчёта границы показана в листинге 14 [24]:

```
def calc_threshold_for_tfidf_stop_words(self,
        tfidf_percent_treshold) -> None:
    all_tfidf_values = []
    for doc in self.tfidf_corpus:
        for _, tfidf_value in doc:
            all_tfidf_values.append(tfidf_value)
    self.threshold_for_tfidf_stop_words = np.percentile(
        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:

```
def del_tfidf_stop_words(self, tfidf_percent_treshold) ->
    None:
    self.__calc_tfidf_corpus_without_zero_score_tokens__()
    self.__add_in_tfidf_corpus_zero_score_tokens__()
    self.__calc_threshold_for_tfidf_stop_words__(tfidf_percent_tresh
    for row, doc in zip(range(self.p_data.shape[0]),
        self.tfidf_corpus):
        tfidf_stop_words = [
            word for word, tfidf_value in doc
            if tfidf_value <
                self.threshold_for_tfidf_stop_words
        ]
        for column in self.processing_columns:
            words_without_tfidf_stop_words = []
            for word in self.p_data.loc[row,
                column].split(" "):
                if word in tfidf_stop_words:
                    continue
                words_without_tfidf_stop_words.append(word)
            self.p_data.loc[
                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].

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

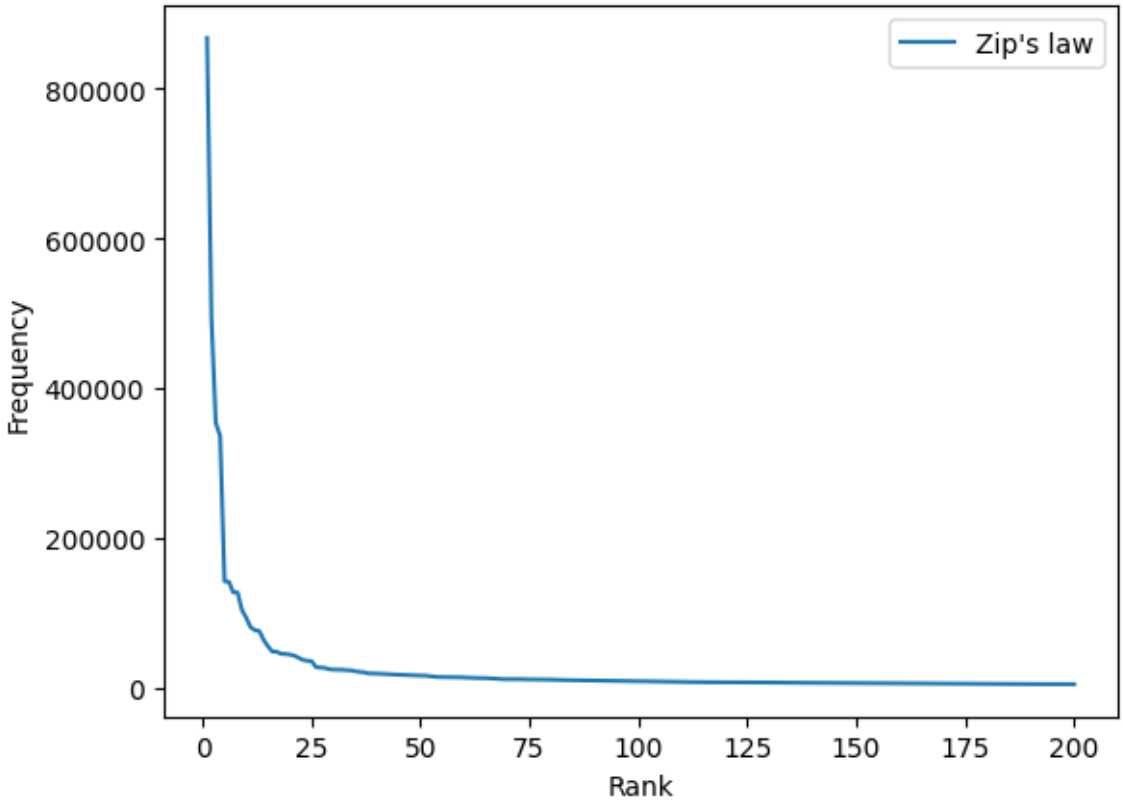

Рисунок 5 – Распределение частот слов по закону Ципфа (исходные данные)

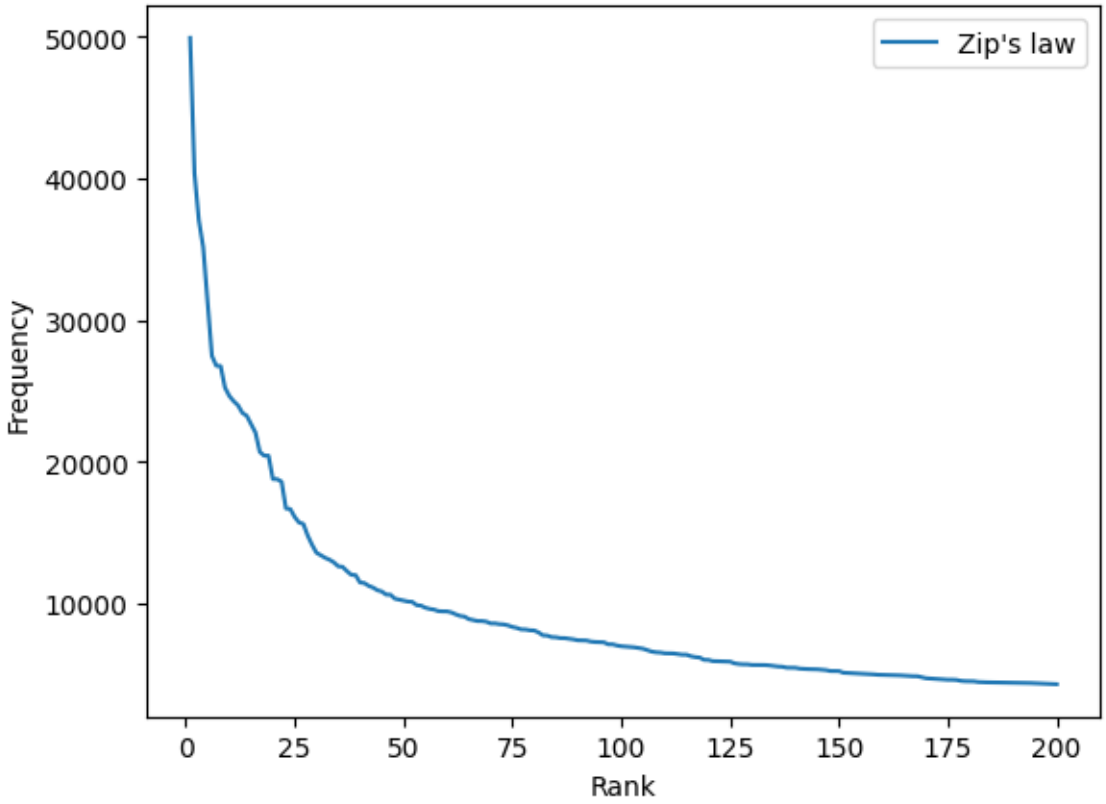

Рисунок 6 – Распределение частот слов по закону Ципфа (обработанные данные)

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 регуляляризаторов (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 туториалов: 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 Hyperparamete 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].

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

```
            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]:

```
def __make_batches__(self) -> None:
    self.batches = artm.BatchVectorizer(
        data_path=self.path_vw,
        data_format="vowpal_wabbit",
        batch_size=self.batch_size,
        target_folder=self.dir_batches
    )
    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]):

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

```
            self.model.regularizers.add(
                artm.SmoothSparseThetaRegularizer(name=name, tau=tau)
            )
            self.user_regularizers[name] = tau
        elif name == "SmoothSparsePhiRegularizer":
            self.model.regularizers.add(
                artm.SmoothSparsePhiRegularizer(name=name, tau=tau)
            )
        # остальные регуляризаторы ...
        else:
            print(
                "Регуляризатора {0} нет! Проверьте корректность названия!".
                format(name)
            )
```

Листинг 19: Функция добавления одиночного регуляризатора

Реализация массового добавления регуляризаторов (листинг 20):

```
def add_regularizers(self, regularizers: dict[str, float]) -> None:
    for regularizer in regularizers:
        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]:

```
def __calc_coherence__(self) -> None:
    last_tokens =
            self.model.score_tracker["top_tokens"].last_tokens
    valid_topics = [tokens for tokens in last_tokens.values() if
            tokens]
    texts = []
    for row in range(self.data.shape[0]):
        words = []
        for column in self.data.columns:
            cell_content = self.data.loc[row, column]
            if isinstance(cell_content, str) and
                    cell_content.strip():
                words += cell_content.split()
        if words:
            texts.append(words)
    dictionary = Dictionary(texts)
    coherence_model = CoherenceModel(
        topics=valid_topics,
        texts=texts,
        dictionary=dictionary,
        coherence="c_v"
    )
    self.coherence = coherence_model.get_coherence()
```

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]):

```
def print_coherence_by_epochs(self) -> None:
    plt.plot(
        range(len(self.coherence_by_epoch)),
        self.coherence_by_epoch,
        label="coherence"
    )
    plt.title("График когерентности")
    plt.xlabel("Epoch")
    plt.ylabel("Coherence")
    plt.legend()
    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 E.

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]):

```
def __objective__(self, trial) -> tuple[float, float,
    float]:
    num_topics = trial.suggest_int(
        self.num_topics[0], self.num_topics[1],
            self.num_topics[2]
    )
    # скрытые остальные гиперпараметры ...
    model = My_BigARTM_model(
        data=self.data,
        num_topics=num_topics,
        num_document_passes=num_document_passes,
        class_ids=class_ids,
        num_collection_passes=num_collection_passes,
        regularizers=regularizers
    )
    model.calc_model()
    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]):

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

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

Листинг 24: Обучение модели с оптимальными параметрами

Полная реализация класса Hyperparameter_optimizer доступна в приложении З.

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, для решения задачи наиболее эффективны трансформеры. Существует три основных типа архитектур:

- 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:

- 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:

- 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 способо получения весов предобученной модели: от их скачивания с облака и github репозиториев, до получения через API разных сайтов. Их этих методов будет предпочтительнее выбрать последний, так как есть портал Hugging Face.

Hugging Face предсталяет собой большое хранилище различных моделей, в том числе и предобученных крупными компаниями и исследователями (Google, Facebook, Sberbank). Кроме того, данный сайи 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:

```
%%capture
!pip install transformers datasets evaluate

from datasets import Dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    EarlyStoppingCallback
)
import evaluate
```

Листинг 25: Подключение необходимых зависимсотей 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). Реализация соответствующего кода представлена в соответствующем листинге 26.

```
self.model = AutoModelForSequenceClassification.from_pretrained(
    self.model_name,
    num_labels=self.num_labels,
    problem_type="single_label_classification",
    ignore_mismatched_sizes=True
).to(self.device)
```

Листинг 26: Загрузка весов модели

2.6.4 Подготовка данных для работы с моделью

Для обработки текста используется токенизатор, соответствующй выбранной модели. Его загрузка осуществляется через класс AutoTokenizer (Листинг 27):

```
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):

```
def __tokenize_data__(self, df: pd.DataFrame) -> Dataset:
    dataset = Dataset.from_pandas(df[['text', 'label']])
    def tokenize_function(examples):
        return self.tokenizer(
            examples["text"],
            padding="max_length",
            truncation=True,
            max_length=self.maximum_sequence_length
        )
    return dataset.map(tokenize_function, batched=True)
```

Листинг 28: Функция токенизации текста

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

```
def __prepare_data__(self):
    self.data['text'] = self.data[self.columns].apply(
        lambda x: ' '.join(x.dropna().astype(str)), axis=1)
    unique_topics = self.data['topic'].unique()
    self.topic2id = {topic: i for i, topic in
            enumerate(unique_topics)}
    self.id2topic = {i: topic for i, topic in
            enumerate(unique_topics)}
    self.num_labels = len(self.topic2id)
    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. Среди них можно выделить следующие:

— 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.

```
training_args = TrainingArguments(
    output_dir=self.output_dir,
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    lr_scheduler_type="linear",
    warmup_steps=100,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    num_train_epochs=10,
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    logging_dir='./logs',
    logging_steps=10,
    report_to="none",
    save_total_limit=1
)
```

Листинг 30: Код установки параметров обучения

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

```
self.trainer = Trainer(
    model=self.model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=self.__compute_metrics__,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=3)]
)
```

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

Листинг 31: Код класса обучения

Таким образом, была реализована основная функциональнось 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 сипользовании 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 выбранны наиболее удобные из них;
4. Проанализированы популярные инструменты для обработки текстовых данных (NLTK, Pymorphy3, 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 помошью аддитивной тематической регуляризации;
7. Разработаны иснтрументы 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 отпровергнуть 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. Эмбеддинги 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 естесвенного 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 безызбыточной 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.

ПРИЛОЖЕНИЕ А
Пример страницы новостного сайта ВШЭ

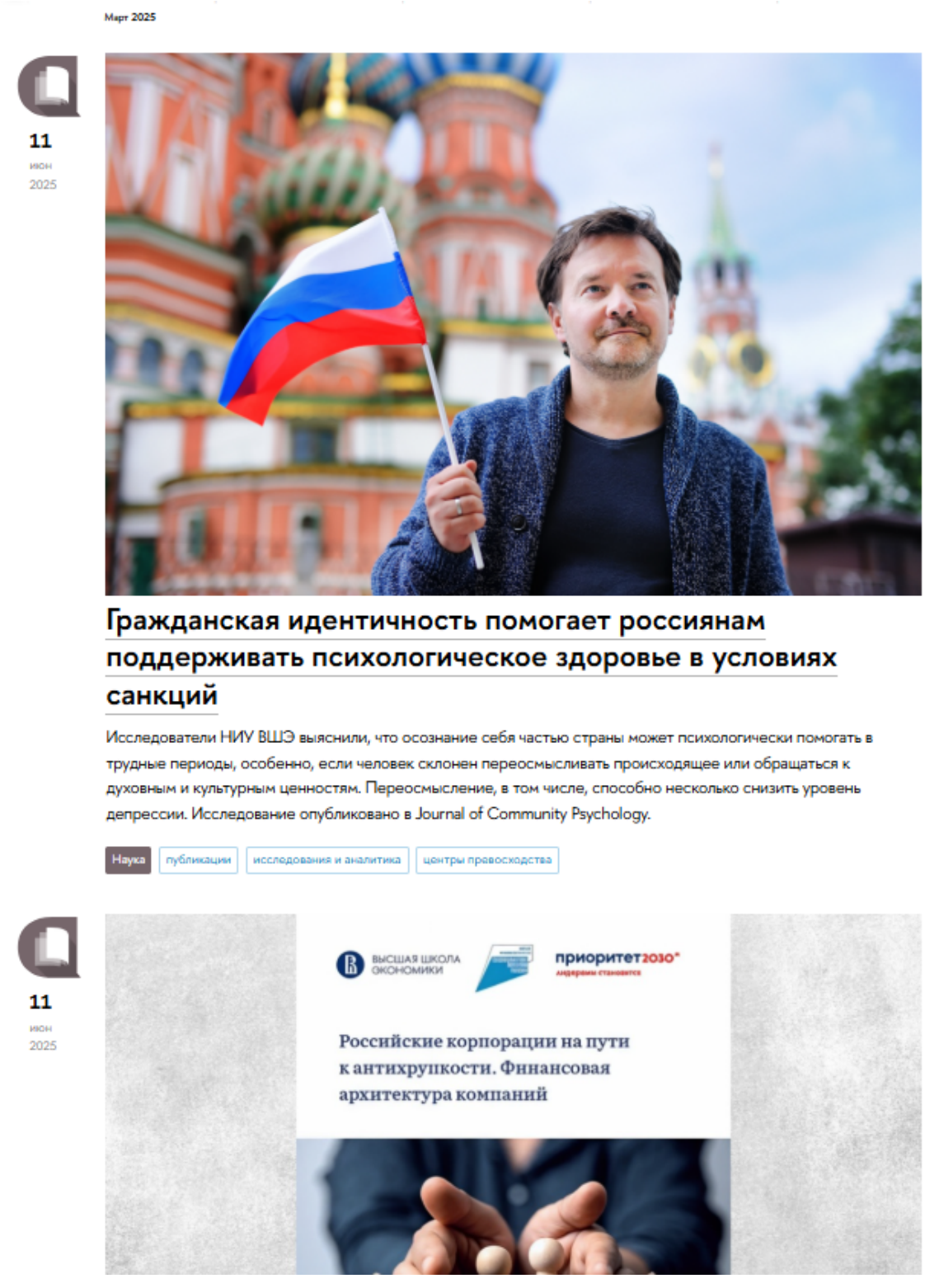

Рисунок 7 – Пример страницы новостного сайта ВШЭ

ПРИЛОЖЕНИЕ Б
Листинг вебскраппера

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
import os
import time
import threading


def __loading_bar_and_info__(
    start: bool, number_of_steps: int, total_steps: int,
           number_of_thread: int
) -> None:
    '''Вывод инфомрации о прогрессе выполнения программы.
    start - нужно ли вывести начальную строку;
    number_page - количество спаршенных страниц;
    total_pages - всего стнраниц, которые нужно спарсить;
    miss_count - число новостей, которые не удалось спарсить;
    whitour_whole_content - число новостей, у которых не получило
           сь полностью спарсить контент.'''
    done = int(number_of_steps / total_steps * 100) if int(
        number_of_steps / total_steps * 100
    ) < 100 or number_of_steps == total_steps else 99
    stars = int(
        40 / 100 * done
    ) if int(20 / 100 * done) < 20 or number_of_steps ==
           total_steps else 39
    tires = 40 - stars

    if start:
        stars = 0
        tires = 40
        done = 0

    print("thread{0} <".format(number_of_thread), end="")
    for i in range(stars):
        print("*", end="")

    for i in range(tires):
        print("-", end="")
```

```
37      print("> {0}% ||| {1} / {2}".format(done, number_of_steps,
            total_steps))
38
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      '''Получиние полного контента новости.
52      url - ссылка но 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",
            class_="main").find(
61          "div", class_="post__text"
62      ).text.strip()
63
64      return content
65
66
67  def __parse_page__(page_file_name: str, news_container:
        pd.DataFrame) -> None:
68      '''Парсинг информации с новостной страницы: ссылка на новость
            + короткая информация о ней.
69      page_file_name - имя файла, в который сохранён код страницы;
70      news_container - таблица, в которую заносится информация о но
            вости.
71      Функция также возвращает количество новостей, которые не удал
            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",
                        class_="post-meta__year").text.strip()
93          except:
94              news_year = ""
95
96          news_date = news_day + "." + news_month + "." + news_year
97
98          try:
99              news_name = news.find("h2",
100                                    class_="first_child").find("a").text.st
101         except:
102             news_name = ""
103
104         try:
105             news_short_content = news.find("p",
                        class_="first_child"
106                                            ).find_next_sibling("p").text.s
107         except:
108             news_short_content = ""
```

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

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])
```

```
                ) if not (token.is_stop) and not
                        (token.is_punct) and
                    len(self.__processing_token__(token.lemma_)) > 1
                ]
            else:
                tokens += [
                    self.__processing_token__(token.lemma_)
                    for token in self.nlp_ru(
                        self.__remove_extra_spaces_and_line_breaks__(part[1])
                    ) if not (token.is_stop) and not
                            (token.is_punct) and
                    len(self.__processing_token__(token.lemma_)) > 1
                ]
        return " ".join(tokens)
```

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

ПРИЛОЖЕНИЕ Г

Листинг обработчика новостного массива

```
class Text_preparer:
    def __init__(self, additional_stop_words_path: str = ""):
        '''Инициализация.\n
        additional_stop_words: пользовательский список стоп
                -слов.'''
        self.nlp_en = spacy.load("en_core_web_sm")
        self.nlp_ru = spacy.load("ru_core_news_sm")

        self.tfidf_corpus = None
        self.tfidf_dictionary = None

    def __first_is_en__(self, cell: str) -> bool:
        '''Определяет начинается строка с символа русского алфави
                та или
        английского алфавита.\n
        cell: строка.\n
        Возвращает true, если строка начинается с символа английс
                кого алфавита.
        '''
        index_first_en = re.search(r"[a-zA-Z]", cell)
        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,
            str)]:
26          '''Разделяет строку на части, в которых содержатся символ
                ы принадлежащие
27          только русскому или английскому алфавиту (то есть в строк
                е с русскими
28          символами не будет символов английского языка и наоборот
                , остальные символы
29          не удаляются).\т
30          cell: строка.\n
31          Возвращает массив кортежей
32          (True(если начинается с символа английского алфавита), по
                дстрока).
33          '''
34          parts = []
35          is_en = self.__first_is_en__(cell)
36          part = ""
37          for symb in cell:
38              if is_en == (symb in string.ascii_letters) or not
                    (symb.isalpha()):
39                  part += symb
40              else:
41                  parts.append((is_en, part))
42                  part = symb
43                  is_en = not (is_en)
44
45          if part:
46              parts.append((is_en, part))
47
48          return parts
49
50      def __remove_extra_spaces_and_line_breaks__(self, text: str)
            -> str:
51          '''Удаляет из строки лишние пробелы и переносы строки.\n
52          text: строка.\n
53          Возвращает строку, с удалёнными лишними пробелами и перено
```

```
            сами строк.
54          '''
55          processed = ""
56
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
                        symb <= "Z"):
79                  num_letters += 1
80              if ("а" <= symb and symb <= "я") or ("А" <= symb and
                        symb <= "Я"):
81                  num_letters += 1
82
83          return num_letters
84
85      def __strip_non_letters__(self, text: str) -> str:
86          return
                re.sub(r"^[^a-zA-Za-яА-ЯёЁ]+|[^a-zA-Za-яА-ЯёЁ]+$",
                "", text)
87
88      def __processing_token__(self, token: str) -> str:
89          new_token = self.__strip_non_letters__(token)
```

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

```
122
123             return " ".join(tokens)
124
125         def
                __calc_tfidf_corpus_without_zero_score_tokens_and_tfidf_dictiona
126             self
127         ) -> None:
128             '''Вычисление tfidf метрики для слов документов + tfidf с
                    ловаря.'''
129             texts = []
130             self.original_tokens = []
131
132             for row in range(self.p_data.shape[0]):
133                 words = []
134                 for column in self.processing_columns:
135                     for word in self.p_data.loc[row, column].split("
                            "):
136                         words.append(word)
137                 self.original_tokens.append(words)
138                 texts.append(words)
139
140             dictionary = gensim.corpora.Dictionary(texts)
141             corpus = [dictionary.doc2bow(text) for text in texts]
142             tfidf = gensim.models.TfidfModel(corpus)
143
144             self.tfidf_corpus = tfidf[corpus]
145             self.tfidf_dictionary = dictionary
146
147         def __add_in_tfidf_corpus_zero_score_tokens__(self) -> None:
148             '''Добавление слов в tfidf_corpus, которые были исключены
                    gensim при
149             подсчёте метрики tfidf (gensim не добавляет слова, которы
                    е встречаются
150             во всех документах или которые имеют 0 метрику tfidf в
                    tfidf_corpus).'''
151             full_corpus = []
152
153             for doc_idx, doc in enumerate(self.tfidf_corpus):
154                 original_words = self.original_tokens[doc_idx]
155                 term_weights = {
156                     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
                ение tfidf
176         которых меньше, считаются стоп-словами.\n
177         tfidf_percent_threshold: процент от всех слов, которые бу
                дут считаться
178         стоп-словами. То есть берём списко всех значений tfidf, с
                ортируем их и
179         значение, которое отсекает от остальной базы 1 процент са
                мых низких
180         значений и будет threshold.'''
181         all_tfidf_values = []
182         for doc in self.tfidf_corpus:
183             for _, tfidf_value in doc:
184                 all_tfidf_values.append(tfidf_value)
185 
186         self.threshold_for_tfidf_stop_words = np.percentile(
187             all_tfidf_values, tfidf_percent_treshold
188         )
189 
190     def del_tfidf_stop_words(self, tfidf_percent_treshold) ->
            None:
191         '''Удаляет стоп-слова на основе посчитанного
                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]),
                        self.tfidf_corpus):
199                 tfidf_stop_words = [
200                     word for word, tfidf_value in doc
201                     if tfidf_value <
                                self.threshold_for_tfidf_stop_words
202                 ]
203
204                 for column in self.processing_columns:
205                     words_without_tfidf_stop_words = []
206                     for word in self.p_data.loc[row, column].split("
                                "):
207                         if word in tfidf_stop_words:
208                             continue
209                         words_without_tfidf_stop_words.append(word)
210                     self.p_data.loc[
211                         row, column] = "
                                    ".join(words_without_tfidf_stop_words)
212
213     def __calc_num_docs_for_words__(self) -> None:
214         self.num_docs_for_words = dict()
215
216         for row in range(self.p_data.shape[0]):
217             for column in self.processing_columns:
218                 words = self.p_data.loc[row, column].split(" ")
219
220                 for word in words:
221                     if word in self.num_docs_for_words.keys():
222                         self.num_docs_for_words[word] += 1
223                     else:
224                         self.num_docs_for_words[word] = 1
225
226     def __count_num_words__(self, doc: str) -> int:
227         return len(doc.split(" "))
228
```

```
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
                pandas DataFrame.\n
252         standart_processing: bool - говорит нужно ли делать станд
                артную обработку;\n
253         tfidf_processing: bool - говорит нужно ли удалять стоп
                -слова на основе
254         tfidf;\n
255         tfidf_percent_treshold: int - какой процент минимальных з
                начений tfidf
256         отсеивать.'''
257         self.p_data = self.data.copy(deep=True)
258         self.p_data.fillna("", inplace=True)
259 
260         if standart_processing:
261             for row in range(self.p_data.shape[0]):
262                 for column in self.processing_columns:
263                     cell = self.p_data.loc[row, column]
264 
265                     if len(cell) > 0:
```

```
                        self.p_data.loc[
                            row, column] =
                                self.__processing_cell__(cell)

            self.__del_docs_with_low_num_words__()
            self.__calc_up_and_down_threshold__()
            self.__calc_num_docs_for_words__()

            for row in range(self.p_data.shape[0]):
                for column in self.processing_columns:
                    words = self.p_data.loc[row, column].split("
                        ")
                    new_words = []

                    for word in words:
                        if self.num_docs_for_words[
                            word
                        ] >= self.down_threshold and
                                self.num_docs_for_words[
                            word] <= self.up_threshold:
                            new_words.append(word)

                    self.p_data.loc[row, column] = "
                        ".join(new_words)

            self.__del_docs_with_low_num_words__()

        if tfidf_processing:
            self.p_data = self.data
            self.p_data = self.p_data.fillna("")
            self.p_data = self.p_data.astype(str)
            self.del_tfidf_stop_words(tfidf_percent_treshold)
            self.__del_docs_with_low_num_words__()

    def add_data(self, data: pd.DataFrame) -> None:
        self.data = data.copy(deep=True)

    def get_data(self) -> pd.DataFrame:
        return self.data

    def add_processing_columns(self, processing_columns:
```

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

Листинг 34: Полный код подготовки новостного массива

ПРИЛОЖЕНИЕ Д

Листинг функции вычисления тематической модели с пошаговым рассчётом 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,
                   num_collection_passes=1
9          )
10         self.__calc_metrics__()
11         self.perplexity_by_epoch.append(self.perplexity)
12         self.coherence_by_epoch.append(self.coherence)
13         self.topic_purities_by_epoch.append(self.topic_purities)
14
15         if epoch > 0:
16             change_perplexity_by_percent = abs(
17                 self.perplexity_by_epoch[epoch - 1] -
18                 self.perplexity_by_epoch[epoch]
19             ) / (self.perplexity_by_epoch[epoch - 1] +
                    self.epsilon) * 100
20             change_coherence_by_percent = \
21                 abs( self.coherence_by_epoch[epoch - 1] - \
22                      self.coherence_by_epoch[epoch] ) / \
23                    ( self.coherence_by_epoch[epoch - 1] + \
```

```
                        self.epsilon ) * 100
                change_topics_purity_by_percent = \
                    abs( self.topic_purities_by_epoch[epoch - 1] - \
                         self.topic_purities_by_epoch[epoch]) / \
                        ( self.topic_purities_by_epoch[epoch - 1] + \
                          self.epsilon ) * 100

                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:
                    break
```

Листинг 35: Функция вычисления тематической модели с пошаговым рассчётом метрик

ПРИЛОЖЕНИЕ Е

Полный код класса My_BigARTM_model

```
class My_BigARTM_model():
    def __init__(
        self,
        data: pd.DataFrame = pd.DataFrame(),
        num_topics: int = 1,
        num_document_passes: int = 1,
        class_ids: dict[str, float] = {"@default_class": 1.0},
        num_processors: int = 8,
        path_vw: str = "./vw.txt",
        batch_size: int = 1000,
        dir_batches: str = "./batches",
        num_top_tokens: int = 10,
        regularizers: dict[str, float] = {},
        num_collection_passes: int = 1,
        plateau_perplexity: float = 0.1,
        plateau_coherence: float = 0.1,
        plateau_topics_purity: float = 0.1,
        epsilon: float = 0.0000001
    ):
        self.data = data.copy(deep=True)
        self.num_topics = num_topics
        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
                        будет строиться модель"
44              )
45          else:
46              self.__make_vowpal_wabbit__()
47              self.__make_batches__()
48              self.__make_model__()
49
50          if self.user_regularizers:
51              self.add_regularizers(self.user_regularizers)
52
53      def __make_vowpal_wabbit__(self) -> None:
54          f = open(self.path_vw, "w")
55
56          for row in range(self.data.shape[0]):
57              string = ""
58              for column in self.data.columns:
59                  string += str(self.data.loc[row, column]) + " "
60
61              f.write("doc_{0} ".format(row) + string.strip() +
                        "\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',
                        dictionary=self.dictionary)
88          )
89          self.model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_sc
90          self.model.scores.add(
91              artm.SparsityThetaScore(name='sparsity_theta_score')
92          )
93          self.model.scores.add(
94              artm.TopTokensScore(
95                  name="top_tokens", num_tokens=self.num_top_tokens
96              )
97          )
98
99      def __calc_coherence__(self) -> None:
100         topics = []
101         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
                    topics]
154         ) / len(topics)
155
156     def __calc_metrics__(self) -> None:
157         self.perplexity =
                self.model.score_tracker['perplexity'].last_value
158         self.sparsity_phi_score =
                self.model.score_tracker['sparsity_phi_score'
159                                                         ].last_value
160         self.sparsity_theta_score = self.model.score_tracker[
161             'sparsity_theta_score'].last_value
162         self.top_tokens =
                self.model.score_tracker['top_tokens'].last_tokens
163         self.__calc_coherence__()
164         self.__calc_phi__()
165         self.__calc_topics_purities__()
166
167     def add_data(self, data: pd.DataFrame) -> None:
168         self.data = data
169
170         self.__make_vowpal_wabbit__()
171         self.__make_batches__()
172         self.__make_model__()
173
174     def add_regularizer(self, name: str, tau: float = 0.0) ->
            None:
175         if name == "SmoothSparseThetaRegularizer":
```

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

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

```
                        self.coherence_by_epoch[epoch]) / \
245                                                 (self.coherence_by_epoch[ep
                                                            - 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_
                                                                - 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:
                        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 =
            "./drive/MyDrive/model") -> None:
```

```
311             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
                   study.best_trials
4      ]
5      metrics = np.array([item[1] for item in params_and_metrics])
6      scaled_metrics = np.zeros_like(metrics)
7      for i in range(metrics.shape[1]):
8          scaler = RobustScaler()
9          scaled_column = scaler.fit_transform(metrics[:,
                   i].reshape(-1, 1)
10                                                 ).flatten()
11         if weights[i] < 0:
12             scaled_column = -scaled_column
13         scaled_metrics[:, i] = scaled_column
14     scaled_params_and_metrics = [
15         (item[0], item[1], scaled_metrics[i].tolist())
16         for i, item in enumerate(params_and_metrics)
17     ]
18     return min(scaled_params_and_metrics, key=lambda trial:
           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] =
                                         ("num_document_passes",
                                         3, 7),
```

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

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

```
            ."""
        params_and_metrics = [
            (trial.params, trial.values) for trial in
                    study.best_trials
        ]
        metrics = np.array([item[1] for item in
                params_and_metrics])

        scaled_metrics = np.zeros_like(metrics)
        for i in range(metrics.shape[1]):
            scaler = RobustScaler()
            scaled_column = scaler.fit_transform(metrics[:,
                    i].reshape(-1, 1)
                                                 ).flatten()
            if weights[i] < 0:
                scaled_column = -scaled_column
            scaled_metrics[:, i] = scaled_column

        scaled_params_and_metrics = [
            (item[0], item[1], scaled_metrics[i].tolist())
            for i, item in enumerate(params_and_metrics)
        ]

        return min(scaled_params_and_metrics, key=lambda trial:
                sum(trial[2]))

    def optimizer(self):
        study = optuna.create_study(
            directions=["minimize", "maximize", "maximize"]
        )
        study.optimize(self.__objective__,
                n_trials=self.n_trials)
        best_trial = self.__select_best_trial__(study,
                weights=[1, -1, -1])
        best_params = best_trial[0]

        # Извлечение основных параметров
        num_topics = best_params[self.num_topics[0]]
        num_document_passes =
                best_params[self.num_document_passes[0]]
        num_collection_passes =
```

```
                best_params[self.num_collection_passes[0]]

        # Извлечение параметров регуляризаторов
        regularizers_params =
                self.__extract_regularizers_params__(best_params)

        print("Лучшие параметры:")
        print(f"Количество тем: {num_topics}")
        print(f"Проходы по документу: {num_document_passes}")
        print(f"Проходы по коллекции: {num_collection_passes}")

        for reg_name, tau_value in regularizers_params.items():
            print(f"{reg_name}: {tau_value:.4f}")

        # Создание финальной модели
        final_model = My_BigARTM_model(
            data=self.data,
            num_topics=num_topics,
            num_document_passes=num_document_passes,
            num_collection_passes=num_collection_passes,
            regularizers=regularizers_params,
            class_ids=self.class_ids
        )
        final_model.calc_model()
        self.model = final_model

    # Остальные методы без изменений
    def get_model(self) -> My_BigARTM_model:
        return self.model

    def save_model(self, path_model: str =
            "./drive/MyDrive/model") -> None:
        self.model.model.dump_artm_model(path_model)

    def save_phi(self, path_phi: str =
            "./drive/MyDrive/phi.xlsx") -> None:
        self.model.model.get_phi().to_excel(path_phi)

    def save_theta(
        self, path_theta: str = "./drive/MyDrive/theta.xlsx"
    ) -> None:
```

```
147                 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	6545045	-	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

```
class TopicClassifier:
    def __init__(
        self,
        data_path: str,
        columns: List[str],
        maximum_sequence_length: int = 200,
        output_dir: str = "./model"
    ):
        try:
            self.data = pd.read_excel(data_path)
            self.data = self.data.fillna("")
            self.data = self.data.astype(str)
        except FileNotFoundError:
            raise ValueError(f"File {data_path} not found!")

        self.model_name = "nikitast/multilang-classifier-roberta"
        self.columns = columns
        self.maximum_sequence_length = maximum_sequence_length
        self.output_dir = output_dir
        self.device = torch.device("cuda" if
              torch.cuda.is_available() else "cpu")

        self.topic2id: Dict[str, int] = {}
        self.id2topic: Dict[int, str] = {}
        self.num_labels: int = 0
        self.tokenizer = None
        self.model = None
        self.trainer = None
        self.evaluation_results: Dict[str, float] = {}

    def __prepare_data__(self):
        self.data['text'] = self.data[self.columns].apply(
            lambda x: ' '.join(x.dropna().astype(str)), axis=1
        )

        unique_topics = self.data['topic'].unique()
        self.topic2id = {topic: i for i, topic in
              enumerate(unique_topics)}
        self.id2topic = {i: topic for i, topic in
```

```
                enumerate(unique_topics)}

        self.num_labels = len(self.topic2id)
        if self.num_labels < 2:
            raise ValueError("At least 2 classes required for
                classification")

        self.data['label'] =
                self.data['topic'].map(self.topic2id)

    def __load_model__(self):
        self.tokenizer =
                AutoTokenizer.from_pretrained(self.model_name)
        self.model =
                AutoModelForSequenceClassification.from_pretrained(
            self.model_name,
            num_labels=self.num_labels,
            problem_type="single_label_classification",
            ignore_mismatched_sizes=True
        ).to(self.device)

    def __tokenize_data__(self, df: pd.DataFrame) -> Dataset:
        dataset = Dataset.from_pandas(df[['text', 'label']])

        def tokenize_function(examples):
            return self.tokenizer(
                examples["text"],
                padding="max_length",
                truncation=True,
                max_length=self.maximum_sequence_length
            )

        return dataset.map(tokenize_function, batched=True)

    def __compute_metrics__(self, eval_pred) -> Dict[str, float]:
        accuracy_metric = evaluate.load("accuracy")
        logits, labels = eval_pred
        predictions = np.argmax(logits, axis=-1)

        metrics = {
            "accuracy":
```

```
                accuracy_metric.compute(predictions=predictions,
                references=labels)["accuracy"],
            "f1_micro": f1_score(labels, predictions,
                average="micro"),
            "f1_macro": f1_score(labels, predictions,
                average="macro"),
            "f1_weighted": f1_score(labels, predictions,
                average="weighted"),
        }

        try:
            if logits.shape[1] == 2:
                metrics["roc_auc"] = roc_auc_score(labels,
                    logits[:, 1])
            else:
                metrics["roc_auc"] = roc_auc_score(
                    labels, logits, multi_class="ovo",
                        average="macro"
                )
        except ValueError:
            metrics["roc_auc"] = float("nan")

        return metrics

    def __print_final_metrics__(self):
        if not self.evaluation_results:
            raise ValueError("Model not evaluated yet. Call
                train_model() first")

        print("\n" + "="*50)
        print("Final Model Evaluation Metrics:")
        print("-"*50)
        for metric, value in self.evaluation_results.items():
            if metric not in ["eval_loss", "epoch"]:
                print(f"{metric.upper():<15}: {value:.4f}")
        print("="*50 + "\n")

    def train_model(self):
        self.__prepare_data__()
        train_df, val_df = train_test_split(
            self.data,
```

```
            test_size=0.2,
            random_state=42,
            stratify=self.data['topic']
        )

        self.__load_model__()

        train_dataset = self.__tokenize_data__(train_df)
        val_dataset = self.__tokenize_data__(val_df)

        training_args = TrainingArguments(
            output_dir=self.output_dir,
            eval_strategy="epoch",
            save_strategy="epoch",
            learning_rate=2e-5,
            lr_scheduler_type="linear",
            warmup_steps=100,
            per_device_train_batch_size=32,
            per_device_eval_batch_size=32,
            num_train_epochs=10,
            weight_decay=0.01,
            load_best_model_at_end=True,
            metric_for_best_model="accuracy",
            logging_dir='./logs',
            logging_steps=10,
            report_to="none",
            save_total_limit=1
        )

        self.trainer = Trainer(
            model=self.model,
            args=training_args,
            train_dataset=train_dataset,
            eval_dataset=val_dataset,
            compute_metrics=self.__compute_metrics__,
            callbacks=[EarlyStoppingCallback(early_stopping_patience=3)]
        )

        self.trainer.train()

        self.evaluation_results = self.trainer.evaluate()
```

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

Листинг 39: Полный код обучения модели классификатора