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, к. 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, включа 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 машинного и глубокого обучения. Первые позволяют выявить скрытые темы в текстовых данных и подготовить разметку для обучения вторых. Алгоритмы машинного и глубокого обучения, в свою очередь, могут классифицировать новые тексты по заданным темам.

Таким образом, целью данной работы является создание механизма автоматической тематической классификации новостей с использованием методов тематического моделирования, машинного и глубокого обучения.

Для достижения цели необходимо решить следующие задачи:

1. Реализовать сбор новостных данных;
2. Разработать механизм предобработки текстовых данных;
3. Вычислить количественные характеристи данных и провести их анализ;
4. Построить тематические модели;
5. Выбрать оптимальную тематическую модель с помощью сравнительного анализа;
6. Подготовить размеченные данные для обучения моделей;
7. Обучить и сравнить эффективность различных моделей машинного и глубокого обучения;
8. Провести анализ полученных результатов.

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, геолокациям, уровням важности или эмоциональной окраске, 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, машинного 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 увеличения ресурсных затрат;
3. Снижение субъективности: исключаются человеческие ошибки, связанные с усталостью или предвзятостью, что повышает достоверность результатов;
4. Выявление скрытых паттернов: методы машинного обучения обнаруживают неочевидные связи между событиями, например, корреляцию между

экономическими новостями и колебаниями рынка.

Автоматическая классификация новостей не заменяет экспертов, но становится их основным помощником, беря на себя рутинные задачи. В разведывательном 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 машинного 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, 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 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:

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. Это критически важно для тематического моделиро-

вания, где искажение смысла слов может привести к некорректной интерпретации контекста. На рис. 1 показаны принципиальные различия между двумя подходами.

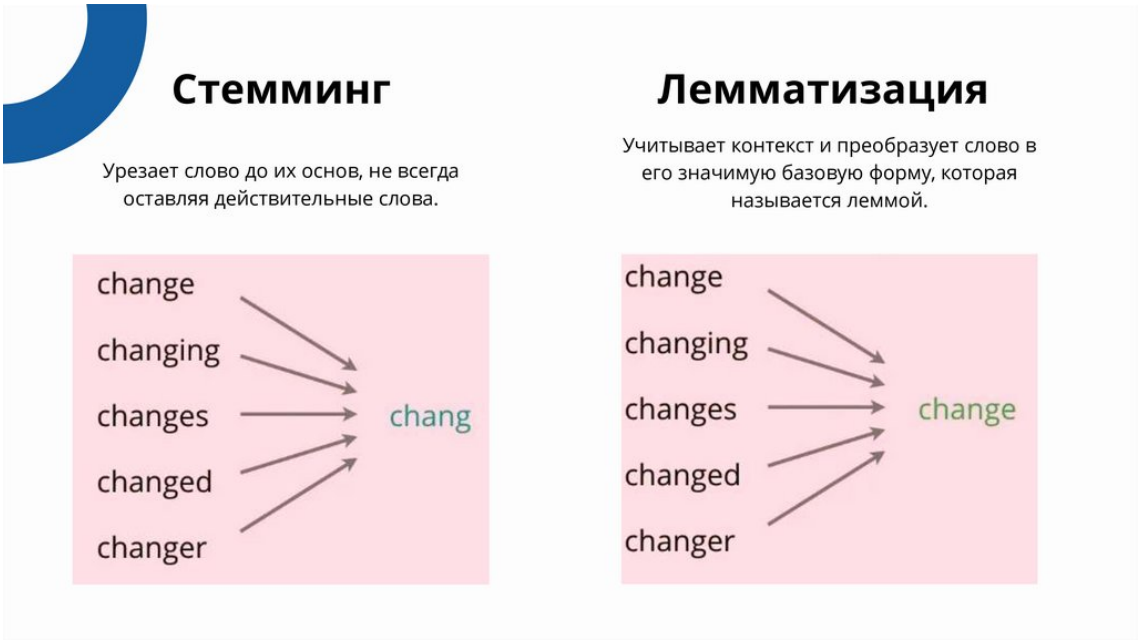

Рисунок 1 – Иллюстрация разницы между стеммингом и лемматизацией

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 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 формирует терм.

Эта гипотеза позволяет проводить тематическую 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:

— 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:

— 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:

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

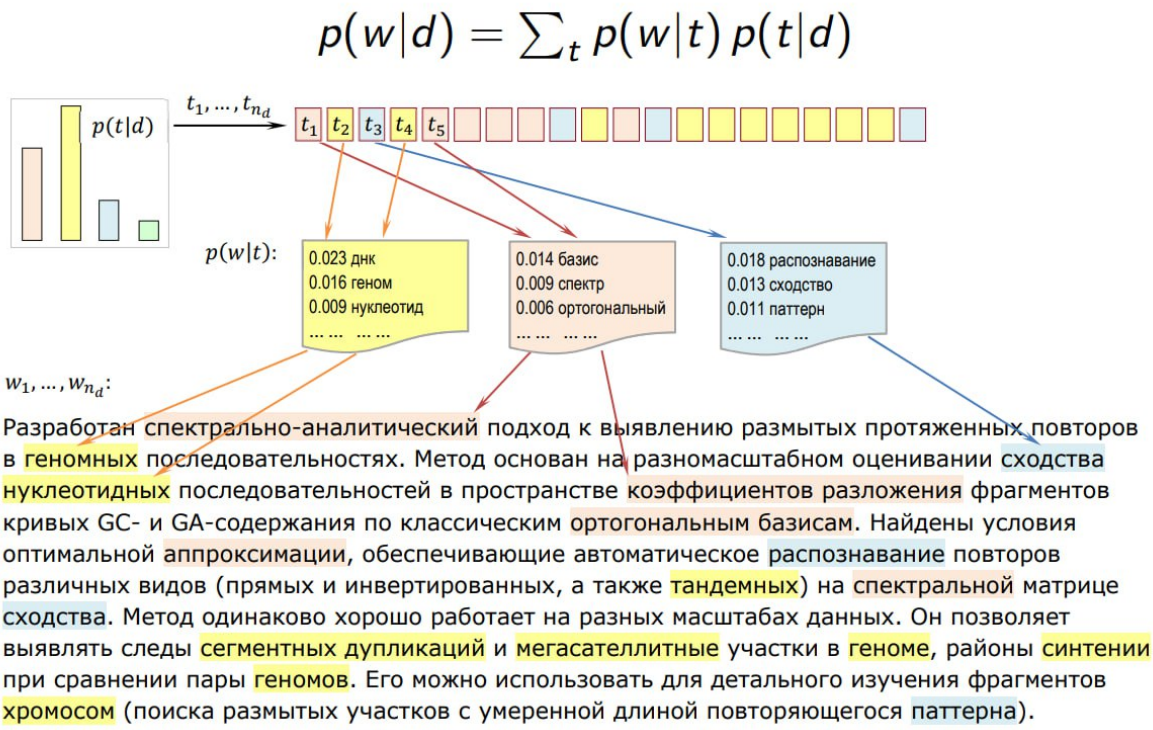

Рисунок 2 – Алгоритм формирования документа

Тогда вероятность появления слова в документе можно описать по формуле полной вероятности:

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

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.

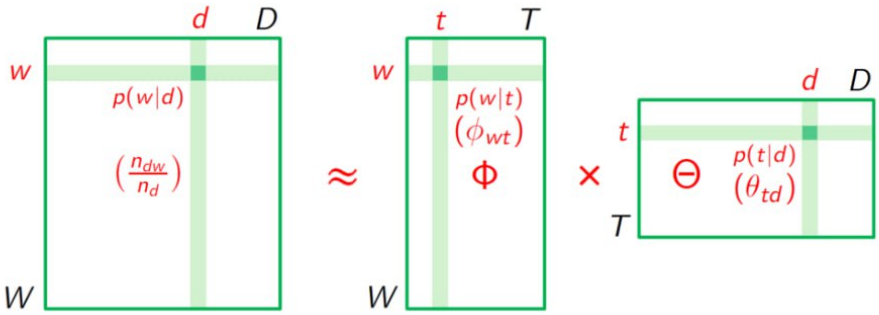

Рисунок 3 – Стохастическое матричное разложение

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.

Введём операцию нормировки вектора:

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

Лемма о максимизации функции на единичных симплексах:

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, 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.

$$\sum_{d \in D} \sum_{w \in d} n_{dw} \ln p(w|d) \underbrace{p(d)}_{const} = n_{dw} \to max \tag{7}$$

Данная задача эквивалентна задаче максимизации функционала

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

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.

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

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

Метод ARTM (аддитивная регуляризация тематических моделей) 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:

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

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.

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 (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:

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.

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 специ-

фичными и описываются относительно небольшим набором терминов, которые не встречаются в других темах.

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 сглаживания:

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

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.

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

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:

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

Таким образом был получен модифицированный EM-алгоритм, декоррелирующий темы.

1.4.6 Оценка качества моделей

После построения модели, очевидно, нужно оценить её качество.

Перечислим основные критерии оценки качества тематических моделей:

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.

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

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.

Когерентность (согласованность) темы t по k топовым словам:

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

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. 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 разре-

женность 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.

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;

— **Эффективным**: 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.

Данный метод работает следующим образом:

1. Создаётся словарь слов 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:

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. Частично решает проблему семантической значимости, но сохраняет другие недостатки BoW.

Эмбеддинги слов: Семантические векторные представления слов:

— 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:

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. Это позволяет учитывать:

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

Недостатки:

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

Преимущества:

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

Трансформеры Обрабатывают всю последовательность слов одновременно благодаря механизму внимания (attention).

Ключевые особенности:

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

Недостатки:

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

Преимущества:

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.

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 машинным обучением. Среди Python-библиотек ключевыми являются:

- 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.

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

2.1.2 Реализация алгоритма сбора новостных данных

библиотек requests и BeautifulSoup4 без привлечения Selenium.

Алгоритм сбора данных включает следующие этапы:

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.
2. Реализация базовых функций (листинг 1):
   - Получение HTML-кода страницы через requests.get();
   - Сохранение сырых данных для последующей обработки.

```
1 def __getPage__(url: str, file_name: str) -> None:
```

```
    # получение html кода страницы с помощью библиотеки
        requests
    r = requests.get(url=url)
    # сохранение полученного кода в текстовый файл
    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):
   - Парсинг сохранённого HTML через BeautifulSoup4;
   - Поиск элементов по тегам и CSS-классам (find(), find_all());
   - Извлечение текстового содержимого (text, get()).

```
# получение html кода страницы из файла
with open(page_file_name, encoding="utf-8") as file:
        src = file.read()
# преобразование html кода в классы
soup = BeautifulSoup(src, "lxml")
# переход к содержимому новости, которое находится
# в теге div с классом post
news = soup.find("div", class_="post")
try:
    # получение текста ссылки из соответствующего тега
    link = news.find("h2",
            class_="first_child").find("a").get("href")
    # не все ссылки в теге сохранены полностью, данный
    # код добавляет обрезанную часть
    if not link.startswith("https://"):
        link = 'https://www.hse.ru' + link
except:
    link = ""
try:
    # получение краткого описания новости из соответствующег
        о тега
    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):

— Рекурсивное использование get_page() для целевых URL;

— Анализ структуры контентной страницы.

```
def __parse_news__(url: str) -> str:
    # получаем html код страницы по ссылке на новость
    news_file_name = "news.html"
    __getPage__(url, news_file_name)
    # и сразу загружаем его из файла
    with open(news_file_name, encoding="utf-8") as file:
        src = file.read()
    # преобразуем html код к классам и сразу получаем всё текстовое содержание
    # новости. Это возможно так как весь контент новости содержится
    # в теге post__text
    content = BeautifulSoup(src, "lxml").find("div", class_="main").find(
        "div", class_="post__text"
    ).text.strip()
    # возвращаем полученное содержание новости в виде строки
    return content
```

Листинг 3: Функция извлечения полного текста новости

5. Обработка страницы целиком (листинг 4):

— 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:
    # скрытый фрагмент получения html кода страницы
    for i in range(10):
        # скрытый фрагмент получения краткой информации о новости
        try: # получение полного содержания новости
            if link.startswith("https://www.hse.ru/news/"):
                news_content = __parse_news__(link)
        except:
            news_content = ""
        # сохранение содержимого новости, если она не пустое
        if len(
            news_day + news_month + news_year + news_name +
```

```
                    news_short_content +
13              news_content
14          ) > 0:
15              news_container.loc[len(news_container.index)] =
                    [
16                  link, news_date, news_name,
                        news_short_content, news_content]
17          # переход к следующей новости
18          news = news.find_next_sibling("div", class_="post")
```

Листинг 4: Обработка новостной страницы

6. Масштабирование на все страницы (листинг 5):
   — Динамическое формирование URL через модификацию параметров;
   — Пакетная обработка через цикл с изменяемым индексом страницы.

```
1  def __crawling_pages__(start: int, end: int,
       news_container: pd.DataFrame, num_of_thread: int) ->
       pd.DataFrame:
2      page_file_name = "page.html"
3      for i in range(start, end + 1):
4          try:
5              __getPage__("https://www.hse.ru/news/page{0}.html".forma
                   page_file_name)
6              __parse_page__(page_file_name, news_container)
7          except:
8              continue
```

Листинг 5: Функция обработки всего архива новостей

7. Оптимизация производительности (листинг 6):
   — Реализация многопоточности через стандартные средства Python;
   — Создание изолированных DataFrame для каждого потока;
   — Агрегация результатов после завершения параллельных задач.

```
1  def crawling_pages(off_pc: bool, pages: int) -> None:
2      columns = ["url", "date", "title", "summary", "content"]
3      # создание контейнеров под каждый из потоков
4      news_container1 = pd.DataFrame(columns=columns)
5      news_container2 = pd.DataFrame(columns=columns)
6      # создание потоков
7      thread1 = threading.Thread(target=__crawling_pages__,
           args=(0, pages // 2, news_container1, 1))
```

```
8       thread2 = threading.Thread(target=__crawling_pages__,
                args=(pages // 2, pages, news_container2, 2))
9       # запуск потоков
10      thread1.start()
11      thread2.start()
12      # ожидание завершения работы потоков
13      thread1.join()
14      thread2.join()
15      # объединение содержимого контейнеров потоков в один
16      try:
17          news = pd.concat([news_container1,
                    news_container2], ignore_index=True)
18          news.to_excel("./news.xlsx")
19      except:
20          print("Не получилось!")
```

Листинг 6: Многопоточная реализация парсера

Полная реализация веб-скрапера доступна в приложении А.

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.

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. Определение стоп-слов в обоих случаях происходит по заранее заданным словарям, поэтому разницы здесь нет. Однако SpaCy обеспечивает не только более точную лемматизацию, но и лаконичный интерфейс, что упрощает её использование.

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:

$$tfidf(w,d) = \frac{n_{wd}}{n_d} \cdot \log\left(\frac{|D|}{|\{d \in D : w \in d\}|}\right), \tag{27}$$

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. 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:

```python
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).

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:

```
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 Обработка двоеточий и временных меток

Библиотека BigARTM интерпретирует двоеточие как служебный символ, что может привести к ошибкам обработки текстовых данных. Для устранения проблемы требуется предварительная нормализация символа.

Стратегия обработки:

1. Сохранение смысла в временных обозначениях: замена шаблонов времени (например, "12:30") на текстовый маркер "time";
2. Удаление избыточных символов: устранение всех других двоеточий, не входящих в временные конструкции

Алгоритм реализует контекстно-зависимую обработку: анализ окружения символа определяет его замену или удаление.

Реализация функции приведена в листинге 9:

```
def __time_processing__(self, text: str) -> str:
```

```
    if re.match(r"\d{2}:\d{2}", text):
        return "time"
    else:
        return text.replace(":", "")

def __processing_token__(self, token_lemma: str) -> str:
    return self.__time_processing__(
        self.__remove_extra_spaces_and_line_breaks__(token_lemma)
    )
```

Листинг 9: Функция нормализации двоеточий в тексте

2.2.5 Токенизация, лемматизация и удаление стоп-слов по словарю

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

- Токенизацию;
- Лемматизацию;
- Идентификацию стоп-слов

Принцип работы:

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:

```
result = " ".join(
    [
        token.lemma_
        for token in
                self.nlp_en(self.__processing_token__(russian_str))
                if
        not (token.is_stop) and not (token.is_punct) and
                len(token.lemma_) > 1
    ]
```

```
)
```

Листинг 10: Обработка строки русского языка средствами SpaCy

Полный алгоритм предобработки, объединяющий нормализацию пробелов, токенизацию и фильтрацию, реализован в листинге 11:

```
def __processing_cell__(self, cell: str) -> str:
    parts = self.__split_into_en_and_ru__(cell)
    tokens = []
    for part in parts:
        if part[0]:
            tokens += [
                token.lemma_
                for token in
                        self.nlp_en(self.__processing_token__(part[1]))
                if not (token.is_stop) and not (token.is_punct)
                        and
                len(token.lemma_) > 1
            ]
        else:
            tokens += [
                token.lemma_
                for token in
                        self.nlp_ru(self.__processing_token__(part[1]))
                if not (token.is_stop) and not (token.is_punct)
                        and
                len(token.lemma_) > 1
            ]
    return " ".join(tokens)
```

Листинг 11: Комплексная обработка текста: нормализация, токенизация, лемматизация, фильтрация стоп-слов

2.2.6 Удаление стоп-слов с помощью метрики tfidf

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.

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:

```
def
        calc_tfidf_corpus_without_zero_score_tokens_and_tfidf_dictiona
        -> None:
    texts = []
    self.original_tokens = []
    for row in range(self.p_data.shape[0]):
        words = []
        for column in self.processing_columns:
            for word in self.p_data.loc[row,
                    column].split(" "):
                words.append(word)
        self.original_tokens.append(words)
        texts.append(words)
    dictionary = gensim.corpora.Dictionary(texts)
    corpus = [dictionary.doc2bow(text) for text in texts]
    tfidf = gensim.models.TfidfModel(corpus)
    self.tfidf_corpus = tfidf[corpus]
    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;

б) Нормализация структуры данных для последующего анализа;

Соответствующая доработка реализована в листинге 13:

```
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 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:

```
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 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_and_tfidf_dictio
    self.add_in_tfidf_corpus_zero_score_tokens()
    self.calc_threshold_for_tfidf_stop_words(tfidf_percent_treshold)
    for row, doc in zip(range(self.p_data.shape[0]),
            self.tfidf_corpus):
```

```
6           tfidf_stop_words = [word for word, tfidf_value in
                doc if tfidf_value <
                self.threshold_for_tfidf_stop_words]
7           for column in self.processing_columns:
8               words_without_tfidf_stop_words = []
9               for word in self.p_data.loc[row,
                        column].split(" "):
10                  if word in tfidf_stop_words:
11                      continue
12                  words_without_tfidf_stop_words.append(word)
13              self.p_data.loc[row, column] = "
                        ".join(words_without_tfidf_stop_words)
```

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

Полная реализация обработчика данных доступна в приложении Б.

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 В.

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. 4-5).

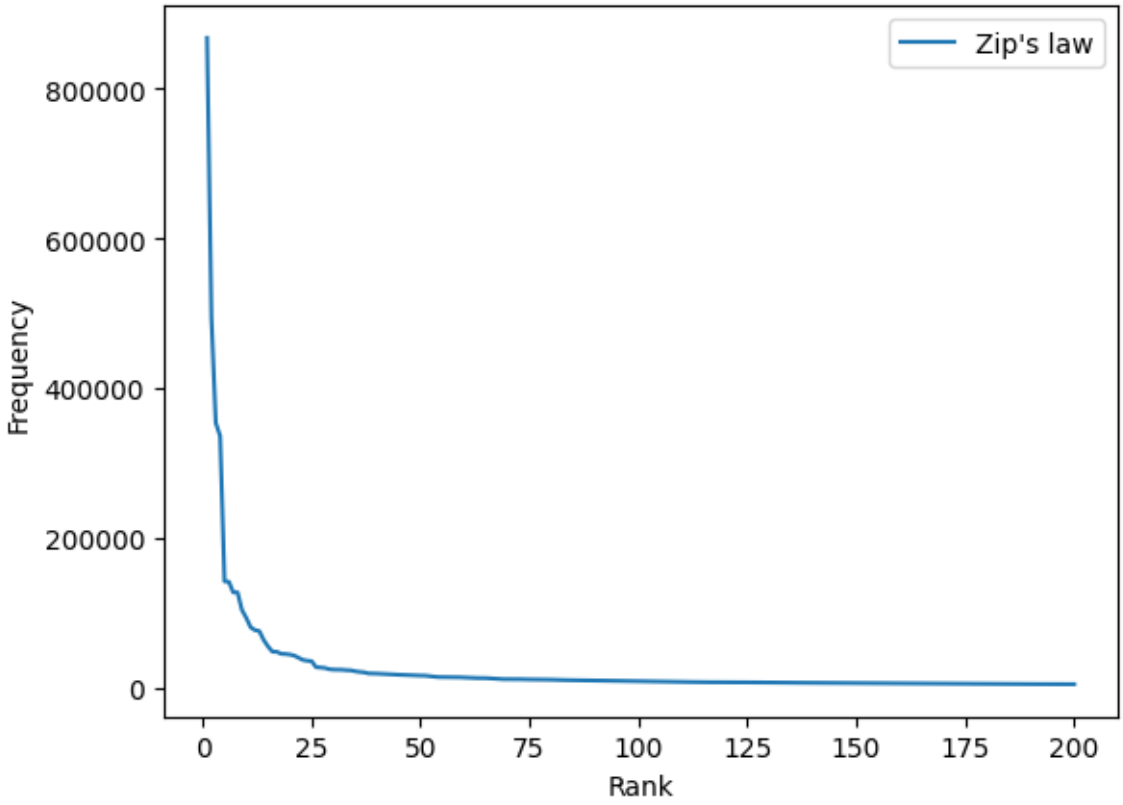

Рисунок 4 – Распределение частот слов по закону Ципфа (исходные данные)

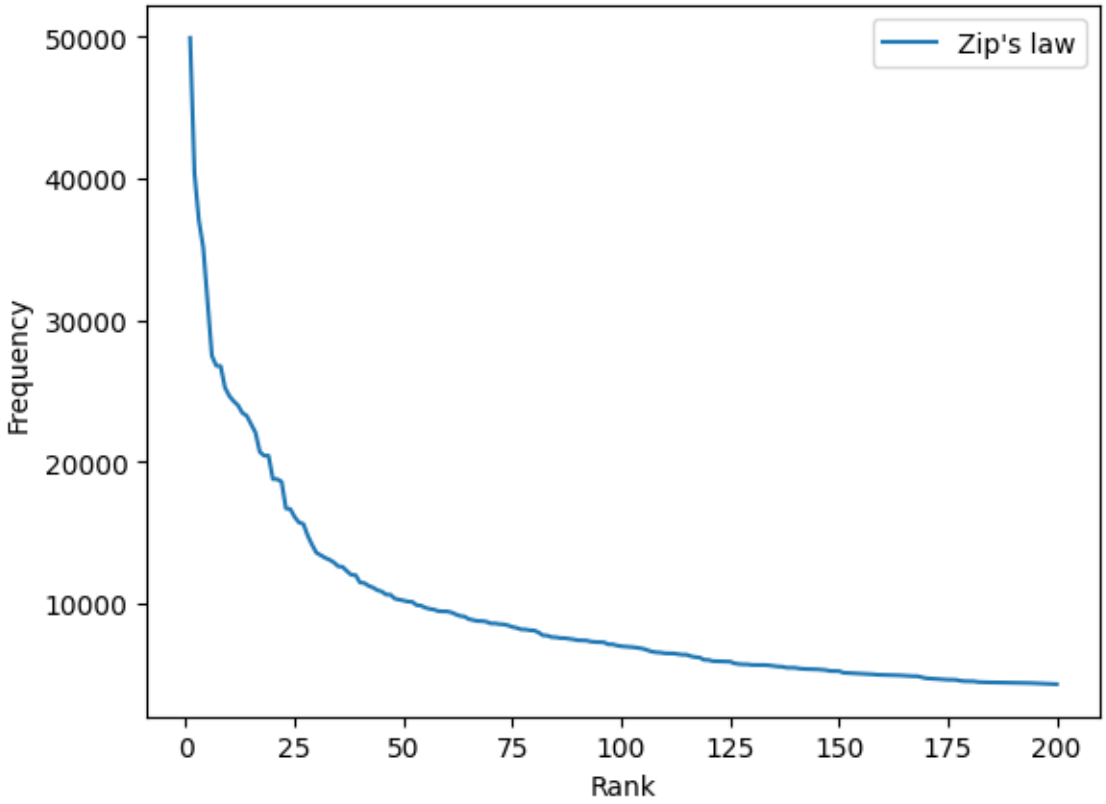

Рисунок 5 – Распределение частот слов по закону Ципфа (обработанные данные)

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»), а шумовые слова требуют фильтрации;
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. Анализатор качества — реализация расчёта когерентности и визуализации метрик;
2. Препроцессинг-обёртка — автоматизация преобразования данных и управления регуляризацией.

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;
— Автоматизацию преобразования данных в требуемый формат;
— Визуализацию динамики метрик качества через графики.

Интеграцию 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:

— Инкапсулирует логику оптимизации;
— Обеспечивает удобное сохранение настроенных моделей;
— Поддерживает различные конфигурации предобработки данных.

Такое разделение обеспечивает модульность архитектуры и упрощает дальнейшее расширение системы.

Следующим этапом работы является последовательная реализация обоих компонентов.

2.4.4 Преобразование новостного массива в приемлемый для BigARTM формат

Модель BigARTM поддерживает ограниченный набор форматов данных, включая Vowpal Wabbit. 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 16:

```
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 16: 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 17:

```
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 17: 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 сложного синтаксиса, что затрудняет массовое использование. Для упрощения процесса предложен двухуровневый подход:

1. Базовая функция — добавляет регуляризатор по имени и значению гиперпараметра;
2. Обёрточная функция — применяет первый метод для пакетного 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 18):

```
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)
            )
        else:
            print(
                "Регуляризатора {0} нет! Проверьте корректность назва
                    ния!".
                format(name)
            )
```

Листинг 18: Функция добавления одиночного регуляризатора

Реализация пакетной обработки (листинг 19):

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

Листинг 19: 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. Для восполнения этого функционала предлагается интеграция с библиотекой Gensim, предоставляющей методы вычисления различных видов когерентности.

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 20:

```
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 20: 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, 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;

— 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 21:

```
def calc_model(self):
    self.perplexity_by_epoch = []
    self.coherence_by_epoch = []
    self.topic_purities_by_epoch = []
    for epoch in range(self.num_collection_passes):
        self.model.fit_offline(
            batch_vectorizer=self.batches,
                num_collection_passes=1
        )
        self.__calc_metrics__()
        self.perplexity_by_epoch.append(self.perplexity)
        self.coherence_by_epoch.append(self.coherence)
        self.topic_purities_by_epoch.append(self.topic_purities)
        if epoch > 0:
            change_perplexity_by_percent = abs(
                self.perplexity_by_epoch[epoch - 1] -
                self.perplexity_by_epoch[epoch]
            ) / (self.perplexity_by_epoch[epoch - 1] +
                self.epsilon) * 100
            change_coherence_by_percent =
                    abs(self.coherence_by_epoch[epoch - 1] -
                    self.coherence_by_epoch[epoch]) / \
                                            (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_ep
                                                    - 1] +
                                                self.epsilon)
                                                * 100
```

```
23                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:
24                    break
```

Листинг 21: Функция обучения модели с пошаговым расчётом метрик

Пример визуализации для метрики когерентности (листинг 22):

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

Листинг 22: Функция построения графика динамики когерентности

Для других метрик применяется аналогичная логика с заменой целевого показателя.

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

2.4.8 Подбор гиперпараметров для тематического моделирования

Для интеллектуального подбора гиперпараметров целесообразно использовать библиотеку Optuna, которая предоставляет:

— 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):

```
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 24):

```
def __select_best_trial__(self, study, weights):
    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]))
```

Листинг 24: Функция выбора оптимальной конфигурации

4. Финализация модели:
   - Обучение на лучших гиперпараметрах;
   - Возврат оптимизированной модели.

Завершающий этап (листинг 25):

```
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 25: Обучение модели с оптимальными параметрами

Полная реализация класса Hyperparameter_optimizer доступна в приложении Д.

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 3 пр.	-	-
С tfidf 4 пр.	-	-
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 3 пр.	-	-	-	-	-
С tfidf 4 пр.	-	-	-	-	-

Данные	topics	cols	docs	tau phi	tau theta
C tfidf 5 пр.	8	6	6	1.139	-1.981
C tfidf 6 пр.	8	6	7	0.954	-1.353
C tfidf 7 пр.	8	5	5	0.942	-0.102
C tfidf 8 пр.	6	7	7	1.757	-1.222
C tfidf 9 пр.	8	6	7	-0.449	-0.365
C tfidf 10 пр.	8	5	6	-0.184	-1.826
C 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 26: 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 27.

```
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 27: 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 28):

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

Листинг 28: 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 29):

```
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 29: Функция токенизации текста

Отдельно преобразуются текстовые метки классов в числовые индексы (Листинг 30):

```
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 30: 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 31.

```
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 31: 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)]
)
self.trainer.train()
```

Листинг 32: Код класса обучения

Таким образом, была реализована основная функциональнось для обучения тематического классификатора. Полный код можно увидеть в соответствующем приложении E.

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
TF-IDF порог 3-10%	не проводилось	

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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А

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="")
```

```
33
34      for i in range(tires):
35          print("-", end="")
36      print("> {0}% ||| {1} / {2}".format(done, number_of_steps,
            total_steps))
37
38  def __getPage__(url: str, file_name: str) -> None:
39      '''Получение html файла страницы.
40      url - ссылка на страницу;
41      file_name - имя файла, в который будет сохранена страница.'''
42      r = requests.get(url=url)
43
44      with open(file_name, "w", encoding="utf-8") as file:
45          file.write(r.text)
46
47  def __parse_news__(url: str) -> str:
48      '''Получчиние полного контента новости.
49      url - ссылка но новость.
50      Функция возвращает полный текст новости.'''
51      news_file_name = "news.html"
52      __getPage__(url, news_file_name)
53
54      with open(news_file_name, encoding="utf-8") as file:
55          src = file.read()
56
57      content = BeautifulSoup(src, "lxml").find("div",
            class_="main").find(
58          "div", class_="post__text"
59      ).text.strip()
60
61      return content
62
63  def __parse_page__(page_file_name: str, news_container:
        pd.DataFrame) -> None:
64      '''Парсинг информации с новостной страницы: ссылка на новость
            + короткая информация о ней.
65      page_file_name - имя файла, в который сохранён код страницы;
66      news_container - таблица, в которую заносится информация о но
            вости.
67      Функция также возвращает количество новостей, которые не удал
            ось спарсить
```

```
68      и количество новостей, полный контент которых спарсить не уда
            лось.'''
69      with open(page_file_name, encoding="utf-8") as file:
70          src = file.read()
71
72      soup = BeautifulSoup(src, "lxml")
73
74      news = soup.find("div", class_="post")
75      for i in range(10):
76          try:
77              news_day = news.find("div",
                        class_="post-meta__day").text.strip()
78          except:
79              news_day = ""
80
81          try:
82              news_month = news.find("div",
83                                     class_="post-meta__month").text.strip(
84          except:
85              news_month = ""
86
87          try:
88              news_year = news.find("div",
                        class_="post-meta__year").text.strip()
89          except:
90              news_year = ""
91
92          news_date = news_day + "." + news_month + "." + news_year
93
94          try:
95              news_name = news.find("h2",
96                                    class_="first_child").find("a").text.st
97          except:
98              news_name = ""
99
100         try:
101             news_short_content = news.find("p",
                        class_="first_child"
102                                            ).find_next_sibling("p").text.s
103         except:
104             news_short_content = ""
```

```
105
106             try:
107                 link = news.find("h2",
                            class_="first_child").find("a").get("href")
108                 if not link.startswith("https://"):
109                     link = 'https://www.hse.ru' + link
110             except:
111                 link = ""
112
113             try:
114                 if link.startswith("https://www.hse.ru/news/"):
115                     news_content = __parse_news__(link)
116             except:
117                 news_content = ""
118
119             if len(
120                 news_day + news_month + news_year + news_name +
                        news_short_content +
121                 news_content
122             ) > 0:
123                 news_container.loc[len(news_container.index)] = [
124                     link, news_date, news_name, news_short_content,
                            news_content
125                 ]
126
127             news = news.find_next_sibling("div", class_="post")
```

Листинг 33: Полный код вебскраппера

ПРИЛОЖЕНИЕ Б

Листинг обработчика новостного массива

```
1  import requests
2  from bs4 import BeautifulSoup
3  import pandas as pd
4  import os
5  import time
6  import threading
7
8  def __loading_bar_and_info__(
9      start: bool, number_of_steps: int, total_steps: int,
           number_of_thread: int
10 ) -> None:
```

```
11      '''Вывод инфомрации о прогрессе выполнения программы.
12      start - нужно ли вывести начальную строку;
13      number_page - количество спаршенных страниц;
14      total_pages - всего стнраниц, которые нужно спарсить;
15      miss_count - число новостей, которые не удалось спарсить;
16      whitour_whole_content - число новостей, у которых не получило
            сь полностью спарсить контент.'''
17      done = int(number_of_steps / total_steps * 100) if int(
18          number_of_steps / total_steps * 100
19      ) < 100 or number_of_steps == total_steps else 99
20      stars = int(
21          40 / 100 * done
22      ) if int(20 / 100 * done) < 20 or number_of_steps ==
            total_steps else 39
23      tires = 40 - stars
24
25      if start:
26          stars = 0
27          tires = 40
28          done = 0
29
30      print("thread{0} <".format(number_of_thread), end="")
31      for i in range(stars):
32          print("*", end="")
33
34      for i in range(tires):
35          print("-", end="")
36      print("> {0}% ||| {1} / {2}".format(done, number_of_steps,
            total_steps))
37
38  def __getPage__(url: str, file_name: str) -> None:
39      '''Получение html файла страницы.
40      url - ссылка на страницу;
41      file_name - имя файла, в который будет сохранена страница.'''
42      r = requests.get(url=url)
43
44      with open(file_name, "w", encoding="utf-8") as file:
45          file.write(r.text)
46
47  def __parse_news__(url: str) -> str:
48      '''Получиние полного контента новости.
```

```
49      url - ссылка но новость.
50      Функция возвращает полный текст новости.'''
51      news_file_name = "news.html"
52      __getPage__(url, news_file_name)
53
54      with open(news_file_name, encoding="utf-8") as file:
55          src = file.read()
56
57      content = BeautifulSoup(src, "lxml").find("div",
            class_="main").find(
58          "div", class_="post__text"
59      ).text.strip()
60
61      return content
62
63  def __parse_page__(page_file_name: str, news_container:
        pd.DataFrame) -> None:
64      '''Парсинг информации с новостной страницы: ссылка на новость
            + короткая информация о ней.
65      page_file_name - имя файла, в который сохранён код страницы;
66      news_container - таблица, в которую заносится информация о но
            вости.
67      Функция также возвращает количество новостей, которые не удал
            ось спарсить
68      и количество новостей, полный контент которых спарсить не уда
            лось.'''
69      with open(page_file_name, encoding="utf-8") as file:
70          src = file.read()
71
72      soup = BeautifulSoup(src, "lxml")
73
74      news = soup.find("div", class_="post")
75      for i in range(10):
76          try:
77              news_day = news.find("div",
                        class_="post-meta__day").text.strip()
78          except:
79              news_day = ""
80
81          try:
82              news_month = news.find("div",
```

```
                                        class_="post-meta__month").text.strip()
        except:
            news_month = ""

        try:
            news_year = news.find("div",
                    class_="post-meta__year").text.strip()
        except:
            news_year = ""

        news_date = news_day + "." + news_month + "." + news_year

        try:
            news_name = news.find("h2",
                                  class_="first_child").find("a").text.st
        except:
            news_name = ""

        try:
            news_short_content = news.find("p",
                    class_="first_child"
                                           ).find_next_sibling("p").text.s
        except:
            news_short_content = ""

        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:
            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 +
121             news_content
122         ) > 0:
123             news_container.loc[len(news_container.index)] = [
124                 link, news_date, news_name, news_short_content,
                        news_content
125             ]
126
127         news = news.find_next_sibling("div", class_="post")
```

Листинг 34: Полный код подготовки новостного массива

ПРИЛОЖЕНИЕ В

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

Продолжение следует...

Продолжение таблицы

Характеристика	**Неподгот.**	**Стоп-слова**	**+Низкочаст.**	**TF-IDF 1%**	**TF-IDF 2%**	**TF-IDF 3%**
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
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
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

Продолжение следует...

Продолжение таблицы

Характеристика	**TF-IDF 4%**	**TF-IDF 5%**	**TF-IDF 6%**	**TF-IDF 7%**	**TF-IDF 8%**	**TF-IDF 9%**
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	-

Продолжение следует...

Продолжение таблицы

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

ПРИЛОЖЕНИЕ Г

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
        self.class_ids = class_ids
        self.num_processors = num_processors
        self.path_vw = path_vw
        self.batch_size = batch_size
        self.dir_batches = dir_batches
        self.num_top_tokens = num_top_tokens
        self.user_regularizers = regularizers
        self.num_collection_passes = num_collection_passes
        self.epsilon = epsilon

        self.perplexity_by_epoch = []
        self.coherence_by_epoch = []
        self.topic_purities_by_epoch = []

        self.plateau_perplexity = plateau_perplexity
        self.plateau_coherence = plateau_coherence
        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

    def get_perplexity(self) -> float:
        return self.perplexity

    def get_perplexity_by_epochs(self) -> list[float]:
        return self.perplexity_by_epoch

    def print_perplexity_by_epochs(self) -> None:
        plt.plot(
            range(len(self.perplexity_by_epoch)),
            self.perplexity_by_epoch,
            label="perplexity"
        )
        plt.title("График перплексии")
        plt.xlabel("Epoch")
        plt.ylabel("Perplexity")
        plt.legend()
        plt.show()

    def get_coherence(self) -> float:
        return self.coherence

    def get_coherence_by_epochs(self) -> list[float]:
        return self.coherence_by_epoch

    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()
```

```
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 35: Полный код класса My_BigRTM_model

ПРИЛОЖЕНИЕ Д

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),
        regularizers: dict[str, tuple[str, float, float]] = {
            "SmoothSparseThetaRegularizer": ('tau_theta', -2.0,
                    2.0),
            "SmoothSparsePhiRegularizer": ('tau_phi', -2.0, 2.0)
        },
        class_ids: dict[str, float] = {"@default_class": 1.0}
    ):
        self.data = data.copy(deep=True)
        self.n_trials = n_trials
        self.num_topics = num_topics
        self.num_document_passes = num_document_passes
        self.num_collection_passes = num_collection_passes
        self.regularizers = regularizers
        self.class_ids = class_ids

        self.robast_scaler = RobustScaler()

    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]
        )
        num_document_passes = trial.suggest_int(
            self.num_document_passes[0],
                    self.num_document_passes[1],
            self.num_document_passes[2]
        )
        num_collection_passes = trial.suggest_int(
            self.num_collection_passes[0],
                    self.num_collection_passes[1],
            self.num_collection_passes[2]
        )
        tau_theta = trial.suggest_float(
            self.regularizers["SmoothSparseThetaRegularizer"][0],
            self.regularizers["SmoothSparseThetaRegularizer"][1],
            self.regularizers["SmoothSparseThetaRegularizer"][2]
        )
        tau_phi = trial.suggest_float(
            self.regularizers["SmoothSparsePhiRegularizer"][0],
            self.regularizers["SmoothSparsePhiRegularizer"][1],
```

```
            self.regularizers["SmoothSparsePhiRegularizer"][2]
        )
        regularizers = {
            "SmoothSparseThetaRegularizer": tau_theta,
            "SmoothSparsePhiRegularizer": tau_phi
        }
        class_ids = self.class_ids

        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()

    def __select_best_trial__(self, study, weights):
        """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
```

```
84
85          scaled_params_and_metrics = [
86              (item[0], item[1], scaled_metrics[i].tolist())
87              for i, item in enumerate(params_and_metrics)
88          ]
89
90          return min(scaled_params_and_metrics, key=lambda trial:
                sum(trial[2]))
91
92      def optimizer(self):
93          study = optuna.create_study(
94              directions=["minimize", "maximize", "maximize"]
95          )
96
97          study.optimize(self.__objective__,
                n_trials=self.n_trials)
98
99          best_trial = self.__select_best_trial__(study,
                weights=[1, -1, -1])
100
101         best_params = best_trial[0]
102
103         num_topics = best_params["num_topics"]
104         num_document_passes = best_params["num_document_passes"]
105         num_collection_passes =
                best_params["num_collection_passes"]
106         tau_theta = best_params["tau_theta"]
107         tau_phi = best_params["tau_phi"]
108
109         print("best params:")
110         print(f"num topics = {num_topics}; num document passes =
                {num_document_passes};\nnum collection passes =
                {num_collection_passes}; tau theta = {tau_theta};
                tau phi = {tau_phi}.")
111
112         final_model = My_BigARTM_model(
113             data=self.data,
114             num_topics=num_topics,
115             num_document_passes=num_document_passes,
116             num_collection_passes=num_collection_passes,
117             regularizers={
```

```
                "SmoothSparseThetaRegularizer": tau_theta,
                "SmoothSparsePhiRegularizer": tau_phi
            },
            class_ids={"@default_class": 1.0}
        )
        final_model.calc_model()

        self.model = final_model

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

Листинг 36: Полный код класса Hyperparameter_optimizer

ПРИЛОЖЕНИЕ Е

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()
        self.__print_final_metrics__()

        self.model.save_pretrained(self.output_dir)
        self.tokenizer.save_pretrained(self.output_dir)

        with open(f"{self.output_dir}/id2topic.json", "w") as f:
            json.dump({str(k): v for k, v in
                    self.id2topic.items()}, f)

    def load_trained_model(self, model_path: str):
        self.tokenizer =
                AutoTokenizer.from_pretrained(model_path)
        self.model =
                AutoModelForSequenceClassification.from_pretrained(model_p
```

```
        with open(f"{model_path}/id2topic.json", "r") as f:
            self.id2topic = {int(k): v for k, v in
                    json.load(f).items()}

    def predict(self, text: str) -> str:
        self.model.eval()
        inputs = self.tokenizer(
            text,
            return_tensors="pt",
            truncation=True,
            max_length=self.maximum_sequence_length
        ).to(self.device)

        with torch.no_grad():
            logits = self.model(**inputs).logits

        predicted_id = torch.argmax(logits, dim=-1).item()
        return self.id2topic[predicted_id]
```

Листинг 37: Полный код обучения модели классификатора