

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	4
1 Теоретические и методологические основы автоматической тематической классификации	5
1.1 Сбор новостных данных	5
1.1.1 Выбор метода получения новостных данных	5
1.1.2 Подбор новостной платформы для сбора данных	5
1.2 Подготовка собранных данных	6
1.3 Математические основы тематического моделирования	7
1.3.1 Основная гипотеза тематического моделирования	7
1.3.2 Аксиоматика тематического моделирования	7
1.3.3 Задача тематического моделирования	8
1.3.4 Решение обратной задачи	9
1.3.5 Регуляризаторы в тематическом моделировании	12
1.3.6 Оценка качества моделей	15
2 Практико-технологические основы автоматической тематической классификации	18
2.1 Получение новостного массива путём веб-скраппинга	18
2.1.1 Выбор инструментов получения новостных данных	18
2.1.2 Реализация алгоритма сбора новостных данных	18
2.2 Подготовка новостного массива	22
2.2.1 Выбор инструментов для подготовки данных	22
2.2.2 Удаление лишних пробелов и переносов строк	23
2.2.3 Разделение строк на русские и английские фрагменты	24
2.2.4 Обработка двоеточий и временных меток	25
2.2.5 Токенизация, лемматизация и удаление стоп-слов по словарю	26
2.2.6 Удаление стоп-слов с помощью метрики tfidf	27
2.3 Количественные характеристики обработанного и необработанного датасета	30
2.4 Вычисление тематической модели	32
2.4.1 Функциональности классов <code>My_BigARTM_model</code> и <code>Hyperparameter_optimizer</code>	32
2.4.2 Преобразование новостного массива в приемлемый для BigARTM формат	33

2.4.3	Удобное добавление регуляризаторов	34
2.4.4	Вычисление когерентности	35
2.4.5	Вычисление тематической модели и формирование гра- фиков метрик	36
2.4.6	Подбор гиперпараметров для тематического моделирования	38
2.5	Результаты тематического моделирования	40
ЗАКЛЮЧЕНИЕ		43
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		43
Приложение А	Листинг вебскраппера	43
Приложение Б	Листинг обработчика новостного массива	46
Приложение В	Количественные характеристики подготовленного и непод- готовленного новостного массива	50
Приложение Г	Полный код класса <code>My_BigARTM_model</code>	55
Приложение Д	Полный код класса <code>Hyperparameter_optimizer</code>	63

ВВЕДЕНИЕ

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

Решением этой проблемы может стать тематическая классификация. Хотя многие сайты и порталы предлагают рубрикацию контента, её точность часто оказывается низкой: теги присваиваются некорректно или поверхностно. Это приводит к ошибкам в поиске и анализе информации.

Для устранения этих недостатков необходим механизм, обеспечивающий точную тематическую классификацию данных с возможностью автоматической разметки новостей. Одним из инструментов для реализации такого подхода являются тематические модели в сочетании с алгоритмами 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 характеристики 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 Сбор новостных данных

1.1.1 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 изучении, а также обеспечивает баланс между скоростью получения данных и минимальными требованиями к стороннему участию.

1.1.2 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.

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

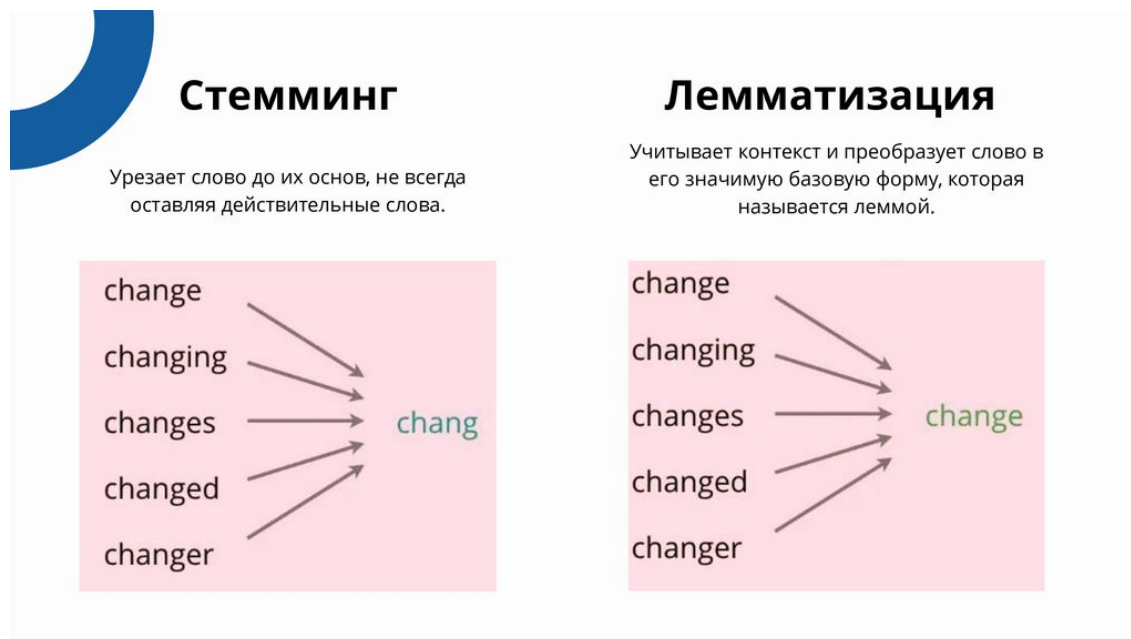


Рисунок 1 – Иллюстрация разницы между стеммингом и лемматизацией

1.3 Математические основы тематического моделирования

1.3.1 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 встречаемости слов.

1.3.2 Аксиоматика тематического моделирования

Каждый текст можно количественно охарактеризовать. Ниже приведены основные количественные характеристики, используемые 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 термина w в документе d ;
- $n_{td} = \sum_w n_{dwt}$ — частота термов темы t в документе d ;
- $n_t = \sum_{d,w} n_{dwt}$ — частота термов темы t в коллекции;
- $n_{dw} = \sum_t n_{dwt}$ — частота термина w в документе d ;
- $n_W = \sum_d n_{dw}$ — частота термина w в коллекции;
- $n_d = \sum_w n_{dw}$ — длина документа d ;
- $n = \sum_{d,w} n_{dw}$ — длина коллекции.

Также в тематическом моделировании используются следующие гипотезы и аксиомы:

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

1.3.3 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 $p(w|d, t)$.

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

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

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

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



Рисунок 2 – Алгоритм формирования документа

Обратную задачу можно представить в виде стохастического матричного разложения 3.

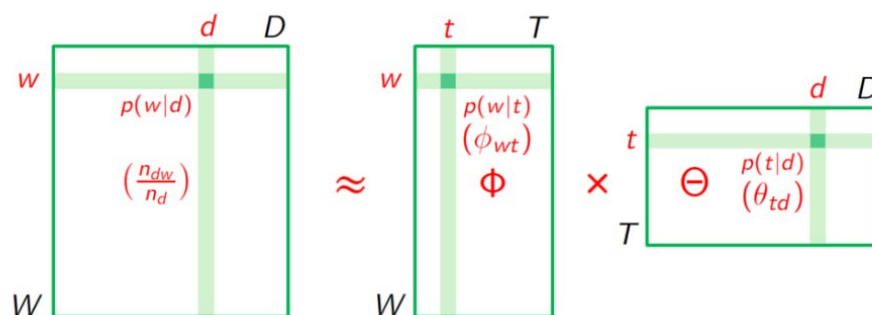


Рисунок 3 – Стохастическое матричное разложение

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

1.3.4 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 помо-

жет в этом процессе.

Введём операцию нормировки вектора:

$$p_i = \left(x_i \right) = \frac{\max x_i, 0}{\sum_{k \in I} \max x_k, 0} \quad (2)$$

Лемма о максимизации функции на единичных симплексах:

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

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

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

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

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

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

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

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

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

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

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

Прологарифмировав правдоподобие, перейдём к задаче максимизации логарифма правдоподобия.

$$\sum_{d \in D} \sum_{w \in d} n_{dw} \ln p(w|d) \xrightarrow{const} n_{dw} \rightarrow \max \quad (7)$$

Данная задача эквивалентна задаче максимизации функционала

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

при ограничениях неотрицательности и нормировки

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

Таким образом, обратная задача сводится к задаче максимизации функционала.

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 τ_i , $i = 1, \dots, k$.

Преобразуем задачу к ARTM виду:

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

при ограничениях неотрицательности и нормировки 9.

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

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

Воспользуемся леммой о максимизации функции на единичных симплексах 1.3.4 и перепишем задачу.

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

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

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

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

1.3.5 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 оцениваться вложенность матриц.

Определим KL-дивергенцию:

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

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

Свойства KL-дивергенции:

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

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

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

Теперь можно перейти к рассмотрению регуляризаторов.

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

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

Определим регуляризатор сглаживания:

Пусть распределения ϕ_{wt} близки к заданному распределению β_w и пусть распределения θ_{td} близки к заданному распределению α_t . Тогда в форме KL-дивергенции 1.3.5 выразим задачу сглаживания:

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

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

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

Перепишем ЕМ-алгоритм 11 в соответствии с полученной формулой:

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

Таким образом был получен модифицированный ЕМ-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 специфичными и описываются относительно небольшим набором терминов, которые не встречаются в других темах.

Определим регуляризатор разреживания:

Пусть распределения ϕ_{wt} далеки от заданного распределения β_w и пусть распределения θ_{td} далеки от заданного распределения α_t . Тогда в форме KL-дивергенции 1.3.5 выразим задачу сглаживания:

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

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

$$R(\Phi, \Theta) = -\beta_0 \sum_{t \in T} \sum_{w \in W} \beta_w \ln \phi_{wt} - \alpha_0 \sum_{d \in D} \sum_{t \in T} \alpha_t \ln \theta_{td} \rightarrow \max. \quad (17)$$

Перепишем ЕМ-алгоритм 11 в соответствии с полученной формулой:

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

Таким образом был получен модифицированный ЕМ-алгоритм, разрежи-

вающий 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 :

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

Перепишем ЕМ-алгоритм 11 в соответствии с полученной формулой:

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

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

1.3.6 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.

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

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

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

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

Когерентность (согласованность) темы t по k топовым словам:

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

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

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

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

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

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

Чистота темы: Чистота темы:

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

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

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

Контрастность темы: Контрастность темы:

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

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

2 Практико-технологические основы автоматической тематической классификации

2.1 Получение новостного массива путём веб-скраппинга

2.1.1 Выбор инструментов получения новостных данных

Для веб-скраппинга доступны библиотеки на разных языках, однако выбор логично сделать в пользу Python — наиболее популярного языка для обработки данных и работы с 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:
```

```

2      # получение html кода страницы с помощью библиотеки
      requests
3      r = requests.get(url=url)
4      # сохранение полученного кода в текстовый файл
5      with open(file_name, "w", encoding="utf-8") as file:
6          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()).

```

1  # получение html кода страницы из файла
2  with open(page_file_name, encoding="utf-8") as file:
3      src = file.read()
4  # преобразование html кода в классы
5  soup = BeautifulSoup(src, "lxml")
6  # переход к содержимому новости, которое находится
7  # в теге div с классом post
8  news = soup.find("div", class_="post")
9  try:
10     # получение текста ссылки из соответствующего тега
11     link = news.find("h2",
12                     class_="first_child").find("a").get("href")
13     # не все ссылки в теге сохранены полностью, данный
14     # код добавляет обрезанную часть
15     if not link.startswith("https://"):
16         link = 'https://www.hse.ru' + link
17 except:
18     link = ""
19 try:
20     # получение краткого описания новости из соответствующег
21     # о тега
22     news_short_content = news.find("p",
23                                     class_="first_child").find_next_sibling("p").text.strip()
24 except:
25     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.

```

1 def __parse_news__(url: str) -> str:
2     # получаем html код страницы по ссылке на новость
3     news_file_name = "news.html"
4     __getPage__(url, news_file_name)
5     # и сразу загружаем его из файла
6     with open(news_file_name, encoding="utf-8") as file:
7         src = file.read()
8     # преобразуем html код к классам и сразу получаем всё те-
9     # кстовое содержание
10    # новости. Это возможно так как весь контент новости сод-
11    #ержится
12    # в теге post__text
13    content = BeautifulSoup(src, "lxml").find("div",
14        class_="main").find(
15        "div", class_="post__text"
16    ).text.strip()
17    # возвращаем полученное содержание новости в виде строки
18    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.

```

1 def __parse_page__(page_file_name: str, news_container:
2     pd.DataFrame) -> None:
3     # скрытый фрагмент получения html кода страницы
4     for i in range(10):
5         # скрытый фрагмент получения краткой информации о но-
6         # вости
7         try: # получение полного содержания новости
8             if link.startswith("https://www.hse.ru/news/"):
9                 news_content = __parse_news__(link)
10            except:
11                news_content = ""
12            # сохранение содержимого новости, если она не пустое
13            if len(
14                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".format
                page_file_name )
6             __parse_page__(page_file_name , news_container)
7         except:
8             continue

```

Листинг 5: Функция обработки всего архива новостей

7. Оптимизация производительности (листинг 6):

- 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, Rymorphy3, SpaCy и Gensim.

Сделаем выбор между связкой NLTK + Rymorphy3 и SpaCy. Обе группы библиотек позволяют проводить лемматизацию и удаление стоп-слов, но реализуют это по-разному. NLTK и Rymorphy3 приводят слова к начальной форме без учёта контекста, тогда как SpaCy — нейросетевой инструмент, анализирующий окружение терминов. 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:

$$tfidf(w, d) = \frac{n_{wd}}{n_d} \cdot \log \left(\frac{|D|}{|\{d \in D : w \in d\}|} \right), \quad (25)$$

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 библиотеки Gensim.

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

2.2.2 Удаление лишних пробелов и переносов строк

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

Алгоритм функции включает три этапа:

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

Реализация функции представлена в листинге 7:

```

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

```

Листинг 7: Функция нормализации пробелов и переносов строк

2.2.3 Разделение строк на русские и английские фрагменты

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

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:

```
1 def __first_is_en__(self, cell: str) -> bool:
2     index_first_en = re.search(r"[a-zA-Z]", cell)
3     index_first_ru = re.search(r"[a-яА-Я]", cell)
4     return True if index_first_en and (not(index_first_ru)
5         or index_first_en.start() <
6         index_first_ru.start()) else False
7
8 def __split_into_en_and_ru__(self, cell: str) -> list[(bool,
9     str)]:
10    parts = []
11    is_en = self.__first_is_en__(cell)
12    part = ""
13    for symb in cell:
14        if is_en == (symb in string.ascii_letters) or not
15            (symb.isalpha()):
16            part += symb
17        else:
18            parts.append((is_en, part))
19            part = symb
20            is_en = not (is_en)
21    if part:
22        parts.append((is_en, part))
23    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:

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

```

2         if re.match(r"\d{2}:\d{2}", text):
3             return "time"
4         else:
5             return text.replace(":", "")
6
7     def __processing_token__(self, token_lemma: str) -> str:
8         return self.__time_processing__(
9             self.__remove_extra_spaces_and_line_breaks__(token_lemma)
10        )

```

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:

```

1 result = " ".join(
2     [
3         token.lemma_
4         for token in
5             self.nlp_en(self.__processing_token__(russian_str))
6             if
7             not (token.is_stop) and not (token.is_punct) and
8             len(token.lemma_) > 1
9     ]
10 )

```

7)

Листинг 10: Обработка строки русского языка средствами SpaCy

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

```
1 def __processing_cell__(self, cell: str) -> str:
2     parts = self.__split_into_en_and_ru__(cell)
3     tokens = []
4     for part in parts:
5         if part[0]:
6             tokens += [
7                 token.lemma_
8                 for token in
9                     self.nlp_en(self.__processing_token__(part[1]))
10                    if not (token.is_stop) and not (token.is_punct)
11                    and
12                    len(token.lemma_) > 1
13            ]
14        else:
15            tokens += [
16                token.lemma_
17                for token in
18                    self.nlp_ru(self.__processing_token__(part[1]))
19                    if not (token.is_stop) and not (token.is_punct)
20                    and
21                    len(token.lemma_) > 1
22            ]
23    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:

```

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

```

Листинг 12: Вычисление TF-IDF метрик для текстового корпуса

2. Коррекция словаря:

- а) Добавление терминов с нулевым TF-IDF, исключённых Gensim по умолчанию;
- б) Нормализация структуры данных для последующего анализа;

Соответствующая доработка реализована в листинге 13:

```

1  def add_in_tfidf_corpus_zero_score_tokens(self) -> None:
2      full_corpus = []
3      for doc_idx, doc in enumerate(self.tfidf_corpus):
4          original_words = self.original_tokens[doc_idx]
5          term_weights = {self.tfidf_dictionary.get(term_id):
              weight for term_id, weight in doc}
6          full_doc = []
7          for word in original_words:
8              if word in term_weights:
9                  weight = term_weights[word]

```

```

10         else :
11             weight = 0.0
12             full_doc.append((word, weight))
13             full_corpus.append(full_doc)
14         self.tfidf_corpus = full_corpus

```

Листинг 13: Дополнение словаря нулевыми TF-IDF значениями

3. Определение порога отсеечения:

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

Логика расчёта границы показана в листинге 14:

```

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

```

1  def del_tfidf_stop_words(self, tfidf_percent_threshold) ->
2      None:
3      self.calc_tfidf_corpus_without_zero_score_tokens_and_tfidf_dictionary()
4      self.add_in_tfidf_corpus_zero_score_tokens()
5      self.calc_threshold_for_tfidf_stop_words(tfidf_percent_threshold)
6      for row, doc in zip(range(self.p_data.shape[0]),
7                          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**).

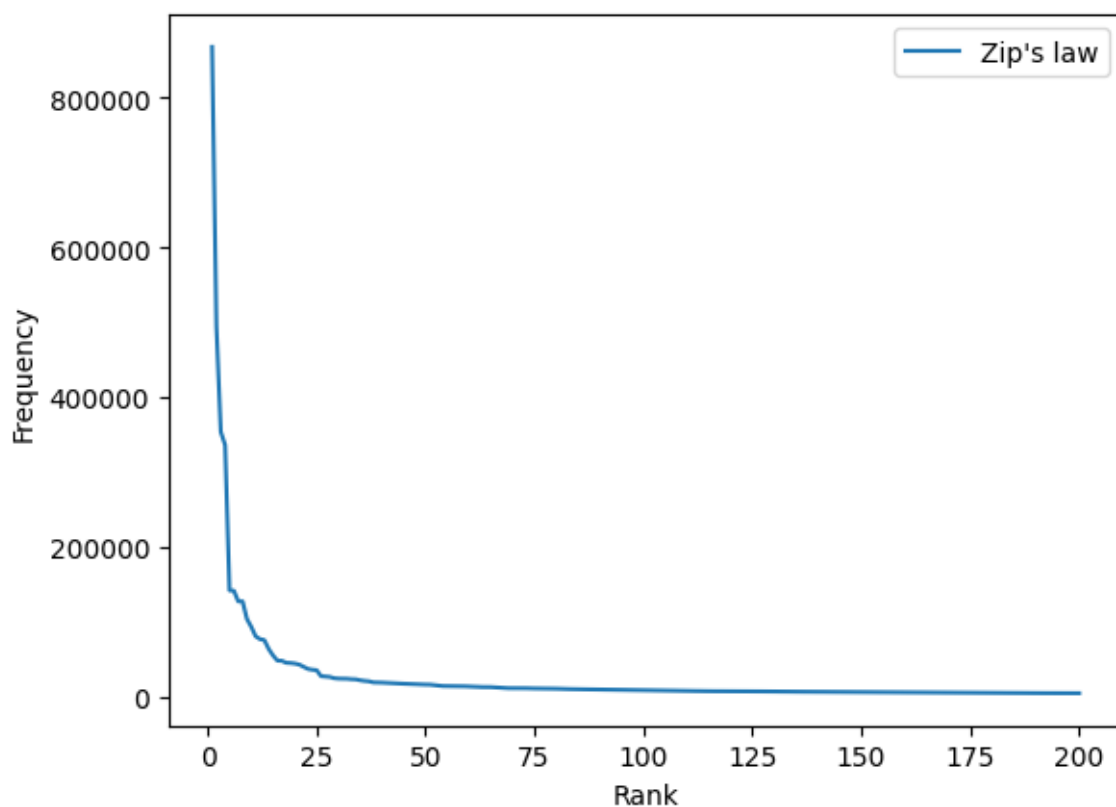


Рисунок 4 – Распределение частот слов по закону Ципфа (исходные данные)

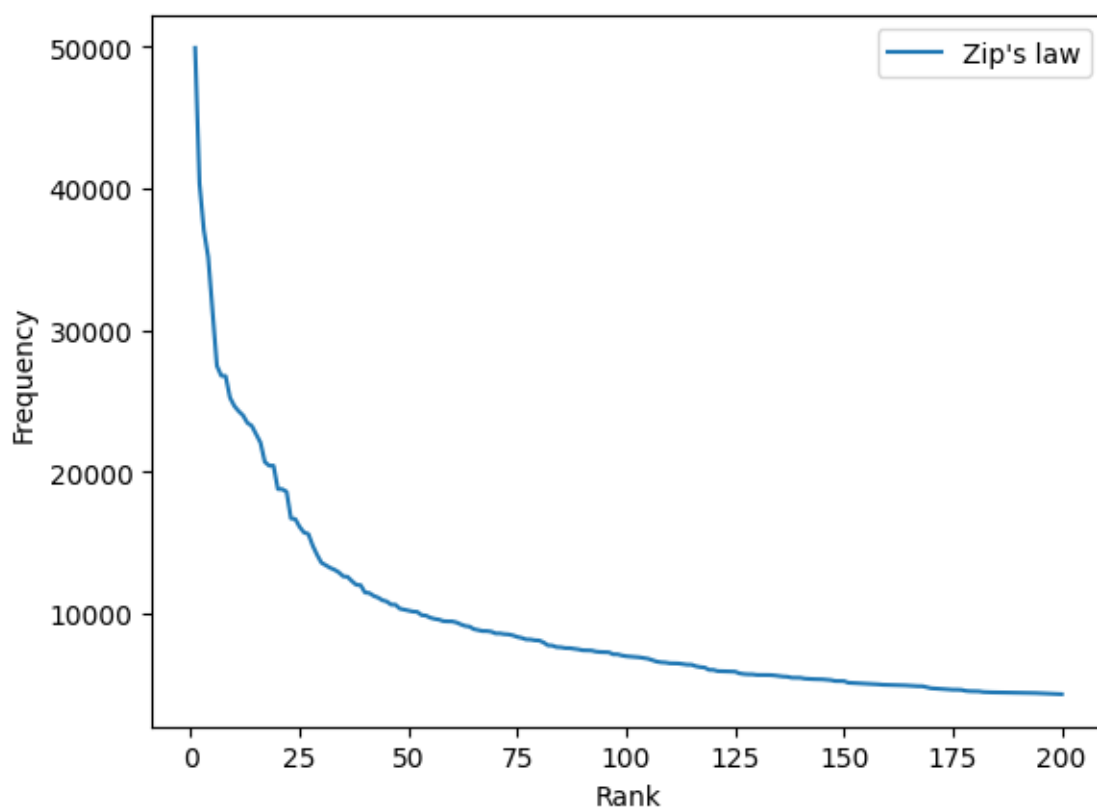


Рисунок 5 – Распределение частот слов по закону Ципфа (обработанные данные)

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

Тематическое моделирование с помощью библиотеки BigARTM достаточно удобно, однако имеет ряд недостатков:

1. Отсутствие такой встроенной метрики как когерентность;
2. Громоздкое добавление регуляризаторов;
3. Громоздкое преобразование данных для вычислений в нужный формат;
4. Отсутствие интерфейса для визуального отслеживания значения метрик качества тематических моделей;
5. Отсутствие возможности подбора оптимальных гиперпараметров;

Среди описанных выше изъянов наиболее существенным является первый, остальные являются скорее неудобствами, которые тем не менее могут сделать код громоздким и нечитабельным.

Чтобы решить описанные выше проблемы можно реализовать два отдельных класса, которые будут добавлять необходимую функциональность.

2.4.1 Функциональности классов `My_BigARTM_model` и `Hyperparameter_optimizer`

В данном классе разумно добавить вычисление когерентности, удобное добавление регуляризаторов и преобразование данных для вычислений в нужный формат, а также создание графиков, визуализирующих изменение метрик для их удобного отслеживания.

Добавлять в класс `My_BigARTM_model` функциональность по подбору гиперпараметров будет излишним, так как это сделает код слишком громоздким и нелогичным, поэтому она будет вынесена в отдельный класс

Hyperparameter_optimizer. Это позволит сделать код более простым и читаемым, а также удобно сохранять модели с оптимально подобранными параметрами для различных типов подготовки данных.

Теперь можно приступить к планомерной реализации обоих классов.

2.4.2 Преобразование новостного массива в приемлемый для BigARTM формат

BigARTM модель умеет работать только с несколькими форматами данных, например, vowpal_wabbit, описание этого формата можно увидеть ниже.

С pandas DataFrame BigARTM работать не умеет, поэтому новостной массив нужно будет преобразовать. Разумно будет это сделать с помощью отдельной функции.

Алгоритм у данной функции будет следующий:

1. Получаем строку из pandas DataFrame;
2. Объединяем ячейки строки в единый текст;
3. Записываем полученную текстовую строку с меткой, что это отдельный документ в соответствующий файл;
4. Повторяем описанные выше действия, пока не будет пройден весь новостной массив.

Реализация соответствующей функции выглядит следующим образом 16.

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

Листинг 16: Функция преобразования новостного массива к vowpal_wabbit формату

После того как данные преобразованы к нужному формату, нужно их разделить на батчи и вычислить словарь, делается это с помощью функций библиотеки BigARTM. Код реализации соответствующей функции представлен в следующем листинге 17.

```
1 def __make_batches__(self) -> None:
2     self.batches = artm.BatchVectorizer(
```

```

3         data_path=self.path_vw,
4         data_format="vowpal_wabbit",
5         batch_size=self.batch_size,
6         target_folder=self.dir_batches
7     )
8     self.dictionary = self.batches.dictionary

```

Листинг 17: Функция вычисления батчей и словаря

Теперь данные можно передавать для вычисления тематической модели.

2.4.3 Удобное добавление регуляризаторов

Модель BigARTM предоставляет большое количество регуляризаторов для использования, однако их добавление в тематическую модель достаточно громоздко и неудобно для массового использования. Поэтому есть необходимость добавления регуляризатора лишь по одному переданному имени и гиперпараметру, минуя трудный синтаксис BigARTM.

Решить данную проблему с точки зрения читабельности кода лучше с помощью двух функций: первая будет добавлять один регуляризатор, а вторая, вызывая первую, будет добавлять сразу несколько регуляризаторов.

Фрагмент реализации функции, добавляющей 1 регуляризатор представлен в следующем листинге 18.

```

1 def add_regularizer(self, name: str, tau: float = 0.0) -> None:
2     if name == "SmoothSparseThetaRegularizer":
3         self.model.regularizers.add(
4             artm.SmoothSparseThetaRegularizer(name=name, tau=tau)
5         )
6         self.user_regularizers[name] = tau
7     elif name == "SmoothSparsePhiRegularizer":
8         self.model.regularizers.add(
9             artm.SmoothSparsePhiRegularizer(name=name, tau=tau)
10        )
11    else:
12        print(
13            "Регуляризатора {0} нет! Проверьте корректность названия!".
14            format(name)
15        )

```

Листинг 18: Фрагмент функции добавляющей 1 регуляризатор

Реализация функции, добавляющей несколько регуляризаторов, выглядит следующим образом 19.

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

Листинг 19: Функция добавляющая несколько регуляризаторов

Таким образом, были добавлены инструменты для удобной работы с BigARTM регуляризаторами.

2.4.4 Вычисление когерентности

Библиотека BigARTM обладает несколькими встроенными метриками качества, однако одной из таких наиважнейших метрик, как когерентность она не обладает. Исправить это можно, реализовав соответствующую функцию на базе библиотеки Gensim (данная библиотека позволяет вычислять различные виды когерентности).

Чтобы вычислить когерентность с помощью библиотеки Gensim необходимо выполнить следующие действия:

1. Получить темы в виде списка ядер тем;
2. Получить документы в виде двумерного списка слов, в котором каждая строка соответствует набору токенов одного документа;
3. Передать вычисленные данные для вычисления когерентности.

Реализация соответствующей функции выглядит следующим образом 20.

```
1 def __calc_coherence__(self) -> None:
2     last_tokens =
        self.model.score_tracker["top_tokens"].last_tokens
3     valid_topics = [tokens for tokens in last_tokens.values() if
        tokens]
4     texts = []
5     for row in range(self.data.shape[0]):
6         words = []
7         for column in self.data.columns:
8             cell_content = self.data.loc[row, column]
9             if isinstance(cell_content, str) and
                cell_content.strip():
10                words += cell_content.split()
```

```

11         if words:
12             texts.append(words)
13         dictionary = Dictionary(texts)
14         coherence_model = CoherenceModel(
15             topics=valid_topics,
16             texts=texts,
17             dictionary=dictionary,
18             coherence="c_v"
19         )
20         self.coherence = coherence_model.get_coherence()

```

Листинг 20: Функция вычисление когерентности

2.4.5 Вычисление тематической модели и формирование графиков метрик

Сама библиотека BigARTM не предоставляет возможности отслеживать процесс изменения метрик при обучении, особенно невстроенных метрик, поэтому данный функционал придётся реализовать отдельно.

Чтобы получить графики изменения метрик нужно их вычислять каждую эпоху формирования тематической модели, за это при её создании отвечает параметр `num_collection_passes`. Однако если мы зададим его отличным от единицы, то получим значение метрик уже после полного вычисления. Тогда необходимо данный параметр передавать не в модель, а в цикл, который будет вычислять модель только для одного прохода по коллекции, а после этого переходить к вычислению значения метрик в текущую эпоху. Таким образом, получим значение метрик за каждую эпоху.

Реализация соответствующей функции представлена в следующем листинге 21.

```

1 def calc_model(self):
2     self.perplexity_by_epoch = []
3     self.coherence_by_epoch = []
4     self.topic_purities_by_epoch = []
5     for epoch in range(self.num_collection_passes):
6         self.model.fit_offline(
7             batch_vectorizer=self.batches,
8             num_collection_passes=1
9         )
10    self.__calc_metrics__()

```

```

10     self.perplexity_by_epoch.append(self.perplexity)
11     self.coherence_by_epoch.append(self.coherence)
12     self.topic_purities_by_epoch.append(self.topic_purities)
13     if epoch > 0:
14         change_perplexity_by_percent = abs(
15             self.perplexity_by_epoch[epoch - 1] -
16             self.perplexity_by_epoch[epoch]
17         ) / (self.perplexity_by_epoch[epoch - 1] +
18             self.epsilon) * 100
19         change_coherence_by_percent =
20             abs(self.coherence_by_epoch[epoch - 1] -
21                 self.coherence_by_epoch[epoch]) / \
22                 (self.coherence_by_epoch[epoch
23                     - 1] +
24                     self.epsilon)
25                     * 100
26         change_topics_purity_by_percent = abs(
27             self.topic_purities_by_epoch[epoch - 1] -
28             self.topic_purities_by_epoch[epoch]) / \
29                 (self.topic_purities_by_epoch
30                     - 1] +
31                     self.epsilon)
32                     * 100
33
34         if change_perplexity_by_percent <
35             self.plateau_perplexity and
36             change_coherence_by_percent <
37             self.plateau_coherence and
38             change_topics_purity_by_percent <
39             self.plateau_topics_purity:
40             break

```

Листинг 21: Функция вычисление модели и метрик качества

После этого остаётся только вычислить соответствующие графики с помощью библиотеки `matplotlib`. Функция построения графика изменения когерентности выглядит следующим образом.

```

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 вычисление графика изменения когерентности

Для остальных метрик код будет аналогичным.

Таким образом, основная функциональность класса `My_BigARTM_model` была реализована. Полный код можно увидеть в соответствующем приложении Г.

2.4.6 Подбор гиперпараметров для тематического моделирования

Реализовать подбор гиперпараметров удобно с помощью библиотеки `optuna`, у неё достаточно простой и удобный интерфейс, а также есть возможно более интеллектуального подбора, не по сетке параметров, а спомощью байесовской оптимизации, что позволяет заметно сократить число попыток на подборку большого числа параметров.

Для работы с `optuna` требуется функция, которая будет производить нужные вычисления и возвращать в качестве результата метрики качества. Также именно в этой функции задаются диапазоны значений гиперпараметров с помощью методов `trial.suggest_int` и `trial.suggest_float`. Ключевые фрагменты соответствующей функции представлены в следующем листинге 23.

```

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

```

```

15     return model.get_perplexity(), model.get_coherence(
16         ), model.get_topic_purities()

```

Листинг 23: Функция вычисления тематической модели для подбора гиперпараметров

Теперь получившуюся функцию можно вызвать для произведения вычислений с помощью метода `study.optimize`, на выходе он вернёт набор попыток, в каждой из которых будут содержаться выбранные гиперпараметры и полученные при обучении метрики качества.

Следующим шагом станет выбор из попыток той, чьи параметры были оптимальными. Для этого нужно будет отмасштабировать метрики и выбрать попытку по минимальной сумме метрик (чем меньше значение, тем качественнее модель). Реализация соответствующей функции представлена в следующем листинге 24.

```

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

```

Листинг 24: Функция вычисления лучшей попытки

Осталось только по полученным оптимальным гиперпараметрам обучить модель и вернуть её в качестве результата. Сделать это можно следующим

образом 25.

```
1 def optimizer(self):
2     study = optuna.create_study(
3         directions=["minimize", "maximize", "maximize"])
4     study.optimize(self.__objective__, n_trials=self.n_trials)
5     best_trial = self.__select_best_trial__(study, weights=[1,
6         -1, -1])
7     best_params = best_trial[0]
8     num_topics = best_params["num_topics"]
9     # скрытые остальные параметры ...
10    # скрытый фрагмент создания финальной модели
11    final_model.calc_model()
12    self.model = final_model
```

Листинг 25: Функция вычисления тематической модели с лучшими параметрами

Таким образом, был реализован основной функционал класса Hyperparameter_optimizer. Посмотреть его код полностью можно в соответствующем приложении Д.

2.5 Результаты тематического моделирования

В рамках данной работы было проведено моделирование со всеми представленными выше подготовленными данными. Для каждого новостного массива были подобраны оптимальные гиперпараметры, с которыми были вычислены финальные модели.

Всего тематических моделей получилось 13, значение когерентности и перплексии для них можно увидеть в соответствующих таблицах.

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

Данные	perplexity	coherence
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.	-	-	-	-	-
С tfidf 5 пр.	8	6	6	1.139	-1.981
С tfidf 6 пр.	8	6	7	0.954	-1.353
С tfidf 7 пр.	8	5	5	0.942	-0.102
С tfidf 8 пр.	6	7	7	1.757	-1.222
С tfidf 9 пр.	8	6	7	-0.449	-0.365
С tfidf 10 пр.	8	5	6	-0.184	-1.826
С tfidf 10 пр. с add.	8	5	6	0.385	-1.165

По ним можно сказать, что наилучшим качеством для тематического моделирования обладает подготовка данных с удалением низкочастотных слов, но без удаления стоп-слов с помощью метрики tfidf. Объясняться это может следующим:

1. Подбор гиперпараметров прошёл недостаточно полно, что не позволило в полной мере выбрать оптимальные гиперпараметры;
2. Рассмотренно недостаточно вариантов подготовки данных (из-за ограниченности времени не были рассмотрены варианты с tfidf удалением стоп-слов и удалением низкочастотных слов);
3. Удаление стоп-слов с помощью метрики tfidf некорректно.

На данный момент можно сказать, что наиболее вероятны первые две причины, для подтверждения третьей не хватает данных.

Также по результатам можно сказать, что высокий процент порога для tfidf метрики негативно влияет на обучение. Связано это, вероятно, с тем, что начинают удаляться уже не только стоп-слова и порог нужно понизить.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А

Листинг вебскраппера

```
1 import requests
2 from bs4 import BeautifulSoup
3 import pandas as pd
4 import os
5 import time
6 import threading
7
8 def __loading_bar_and_info__(
9     start: bool, number_of_steps: int, total_steps: int,
10     number_of_thread: int
11 ) -> None:
12     '''Вывод информации о прогрессе выполнения программы.
13     start - нужно ли вывести начальную строку;
14     number_page - количество спаршенных страниц;
15     total_pages - всего страниц, которые нужно спарсить;
16     miss_count - число новостей, которые не удалось спарсить;
17     whitour_whole_content - число новостей, у которых не получило
18         сь полностью спарсить контент.'''
19     done = int(number_of_steps / total_steps * 100) if int(
20         number_of_steps / total_steps * 100
21     ) < 100 or number_of_steps == total_steps else 99
22     stars = int(
23         40 / 100 * done
24     ) if int(20 / 100 * done) < 20 or number_of_steps ==
25         total_steps else 39
26     tises = 40 - stars
27
28     if start:
29         stars = 0
30         tises = 40
31         done = 0
32
33     print("thread{0} <".format(number_of_thread), end=" ")
34     for i in range(stars):
35         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     '''Получение полного контента новости.
49     url - ссылка на 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()
83         except:
84             news_month = ""
85
86         try:
87             news_year = news.find("div",
                                    class_="post-meta__year").text.strip()
88         except:
89             news_year = ""
90
91         news_date = news_day + "." + news_month + "." + news_year
92
93         try:
94             news_name = news.find("h2",
                                    class_="first_child").find("a").text.st
95         except:
96             news_name = ""
97
98         try:
99             news_short_content = news.find("p",
100                                             class_="first_child")
101                                     ).find_next_sibling("p").text.st
102         except:
103             news_short_content = ""
104

```

```

105
106     try:
107         link = news.find("h2",
108                           class_="first_child").find("a").get("href")
109         if not link.startswith("https://"):
110             link = 'https://www.hse.ru' + link
111     except:
112         link = ""
113
114     try:
115         if link.startswith("https://www.hse.ru/news/"):
116             news_content = __parse_news__(link)
117     except:
118         news_content = ""
119
120     if len(
121         news_day + news_month + news_year + news_name +
122         news_short_content +
123         news_content
124     ) > 0:
125         news_container.loc[len(news_container.index)] = [
126             link, news_date, news_name, news_short_content,
127             news_content
128
129
130     news = news.find_next_sibling("div", class_="post")

```

Листинг 26: 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,
10     number_of_thread: int
11 ) -> None:

```

```

11     '''Вывод информации о прогрессе выполнения программы.
12     start - нужно ли вывести начальную строку;
13     number_page - количество спаршенных страниц;
14     total_pages - всего страниц, 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     tiores = 40 - stars
24
25     if start:
26         stars = 0
27         tiores = 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(tiores):
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     '''Получение полного контента новости.

```

```

49     url - ссылка на 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",
58         class_="main").find(
59         "div", class_="post__text"
60     ).text.strip()
61
62     return content
63
64 def __parse_page__(page_file_name: str, news_container:
65     pd.DataFrame) -> None:
66     '''Парсинг информации с новостной страницы: ссылка на новость
67         + короткая информация о ней.
68     page_file_name - имя файла, в который сохранён код страницы;
69     news_container - таблица, в которую заносится информация о но
70     вости.
71     Функция также возвращает количество новостей, которые не удал
72     ось спарсить
73     и количество новостей, полный контент которых спарсить не уда
74     лось. '''
75     with open(page_file_name, encoding="utf-8") as file:
76         src = file.read()
77
78     soup = BeautifulSoup(src, "lxml")
79
80     news = soup.find("div", class_="post")
81     for i in range(10):
82         try:
83             news_day = news.find("div",
84                 class_="post-meta__day").text.strip()
85         except:
86             news_day = ""
87
88         try:
89             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",
89                             class_="post-meta__year").text.strip()
90 except:
91     news_year = ""
92
93 news_date = news_day + "." + news_month + "." + news_year
94
95 try:
96     news_name = news.find("h2",
97                             class_="first_child").find("a").text.strip()
98 except:
99     news_name = ""
100
101 try:
102     news_short_content = news.find("p",
103                                     class_="first_child")
104                                     ).find_next_sibling("p").text.strip()
105 except:
106     news_short_content = ""
107
108 try:
109     link = news.find("h2",
110                       class_="first_child").find("a").get("href")
111     if not link.startswith("https://"):
112         link = 'https://www.hse.ru' + link
113 except:
114     link = ""
115
116 try:
117     if link.startswith("https://www.hse.ru/news/"):
118         news_content = __parse_news__(link)
119 except:
120     news_content = ""
121
122 if len(
123     news_day + news_month + news_year + news_name +

```

```

121         news_short_content +
122         news_content
123     ) > 0:
124         news_container.loc[len(news_container.index)] = [
125             link, news_date, news_name, news_short_content,
126             news_content
127
128         ]
129
130     news = news.find_next_sibling("div", class_="post")

```

Листинг 27: 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	16545045	-	6479545	6414045	6348544
Кол. уник. ток.	278724	148677	-	148677	148677	148677
Мин. кол. ток. в док.	6	4	-	4	4	4
Модальное кол. ток. в док.	47	31	-	31	31	30
Среднее кол. ток. в док.	695	375	-	371	367	364

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

```
1 class My_BigARTM_model():
2     def __init__(
3         self,
4         data: pd.DataFrame = pd.DataFrame(),
5         num_topics: int = 1,
6         num_document_passes: int = 1,
7         class_ids: dict[str, float] = {"@default_class": 1.0},
8         num_processors: int = 8,
9         path_vw: str = "./vw.txt",
10        batch_size: int = 1000,
11        dir_batches: str = "./batches",
12        num_top_tokens: int = 10,
13        regularizers: dict[str, float] = {},
14        num_collection_passes: int = 1,
15        plateau_perplexity: float = 0.1,
16        plateau_coherence: float = 0.1,
17        plateau_topics_purity: float = 0.1,
18        epsilon: float = 0.0000001
19    ):
20        self.data = data.copy(deep=True)
21        self.num_topics = num_topics
22        self.num_document_passes = num_document_passes
23        self.class_ids = class_ids
24        self.num_processors = num_processors
25        self.path_vw = path_vw
26        self.batch_size = batch_size
27        self.dir_batches = dir_batches
28        self.num_top_tokens = num_top_tokens
29        self.user_regularizers = regularizers
30        self.num_collection_passes = num_collection_passes
31        self.epsilon = epsilon
32
33        self.perplexity_by_epoch = []
34        self.coherence_by_epoch = []
35        self.topic_purities_by_epoch = []
36
37        self.plateau_perplexity = plateau_perplexity
38        self.plateau_coherence = plateau_coherence
39        self.plateau_topics_purity = plateau_topics_purity
```

```

40
41     if data.empty:
42         print(
43             "Чтобы создать модель добавьте данные, на которых
44             будет строиться модель"
45         )
46     else:
47         self.__make_vowpal_wabbit__()
48         self.__make_batches__()
49         self.__make_model__()
50
51     if self.user_regularizers:
52         self.add_regularizers(self.user_regularizers)
53
54 def __make_vowpal_wabbit__(self) -> None:
55     f = open(self.path_vw, "w")
56
57     for row in range(self.data.shape[0]):
58         string = ""
59         for column in self.data.columns:
60             string += str(self.data.loc[row, column]) + " "
61
62         f.write("doc_{0} ".format(row) + string.strip() +
63             "\n")
64
65 def __make_batches__(self) -> None:
66     self.batches = artm.BatchVectorizer(
67         data_path=self.path_vw,
68         data_format="vowpal_wabbit",
69         batch_size=self.batch_size,
70         target_folder=self.dir_batches
71     )
72
73     self.dictionary = self.batches.dictionary
74
75 def __make_model__(self) -> None:
76     self.model = artm.ARTM(
77         cache_theta=True,
78         num_topics=self.num_topics,
79         num_document_passes=self.num_document_passes,
80         dictionary=self.dictionary,

```



```

79         class_ids=self.class_ids ,
80         num_processors=8
81     )
82
83     self.__add_BigARTM_metrics__()
84
85     def __add_BigARTM_metrics__(self) -> None:
86         self.model.scores.add(
87             artm.PerplexityScore(name='perplexity',
88                                   dictionary=self.dictionary)
89         )
90         self.model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_score',
91                                                       dictionary=self.dictionary)
92         )
93         self.model.scores.add(
94             artm.SparsityThetaScore(name='sparsity_theta_score',
95                                     dictionary=self.dictionary)
96         )
97         self.model.scores.add(
98             artm.TopTokensScore(
99                 name="top_tokens", num_tokens=self.num_top_tokens
100             )
101         )
102
103     def __calc_coherence__(self) -> None:
104         topics = []
105         if "top_tokens" in self.model.score_tracker:
106             last_tokens =
107                 self.model.score_tracker["top_tokens"].last_tokens
108             topics = [last_tokens[topic] for topic in
109                       last_tokens]
110
111         valid_topics = []
112         for topic in topics:
113             if isinstance(topic, list) and len(topic) > 0:
114                 valid_topics.append(topic)
115
116         if not valid_topics:
117             self.coherence = 0.0
118             return
119
120         texts = []
121         for row in range(self.data.shape[0]):
122             words = []

```

```

117         for column in self.data.columns:
118             cell_content = self.data.loc[row, column]
119             if isinstance(cell_content, str) and
120                 cell_content.strip():
121                 words += cell_content.split()
122         if words:
123             texts.append(words)
124
125     if not texts:
126         self.coherence = 0.0
127         return
128
129     try:
130         dictionary = Dictionary(texts)
131         coherence_model = CoherenceModel(
132             topics=valid_topics,
133             texts=texts,
134             dictionary=dictionary,
135             coherence="c_v"
136         )
137         self.coherence = coherence_model.get_coherence()
138     except Exception as e:
139         print(f"Ошибка при расчете когерентности: {e}")
140         self.coherence = 0.0
141
142     def __calc_phi__(self) -> None:
143         self.phi = np.sort(self.model.get_phi(), axis=0)[::-1, :]
144
145     def __calc_theta__(self) -> None:
146         self.theta = self.model.get_theta()
147
148     def __calc_topic_purity__(self, topic: int) -> None:
149         return np.sum(self.phi[:, topic]) / self.phi.shape[0]
150
151     def __calc_topics_purities__(self) -> None:
152         topics = range(self.phi.shape[1])
153         self.topic_purities = sum(
154             [self.__calc_topic_purity__(topic) for topic in
155              topics]
156         ) / len(topics)

```

```

156 def __calc_metrics__(self) -> None:
157     self.perplexity =
158         self.model.score_tracker['perplexity'].last_value
159     self.sparsity_phi_score =
160         self.model.score_tracker['sparsity_phi_score']
161         self.model.score_tracker['sparsity_phi_score'].last_value
162     self.sparsity_theta_score = self.model.score_tracker[
163         'sparsity_theta_score'].last_value
164     self.top_tokens =
165         self.model.score_tracker['top_tokens'].last_tokens
166     self.__calc_coherence__()
167     self.__calc_phi__()
168     self.__calc_topics_purities__()
169
170 def add_data(self, data: pd.DataFrame) -> None:
171     self.data = data
172
173     self.__make_vowpal_wabbit__()
174     self.__make_batches__()
175     self.__make_model__()
176
177 def add_regularizer(self, name: str, tau: float = 0.0) ->
178     None:
179     if name == "SmoothSparseThetaRegularizer":
180         self.model.regularizers.add(
181             artm.SmoothSparseThetaRegularizer(name=name,
182                 tau=tau)
183         )
184         self.user_regularizers[name] = tau
185     elif name == "SmoothSparsePhiRegularizer":
186         self.model.regularizers.add(
187             artm.SmoothSparsePhiRegularizer(name=name,
188                 tau=tau)
189         )
190         self.user_regularizers[name] = tau

```

```

190     elif name == "LabelRegularizationPhiRegularizer":
191         self.model.regularizers.add(
192             artm.LabelRegularizationPhiRegularizer(name=name,
193                                                         tau=tau)
194         )
195         self.user_regularizers[name] = tau
196     elif name == "HierarchicalSparsityPhiRegularizer":
197         self.model.regularizers.add(
198             artm.HierarchicalSparsityPhiRegularizer(name=name,
199                                                         tau=tau)
200         )
201         self.user_regularizers[name] = tau
202     elif name == "TopicSelectionThetaRegularizer":
203         self.model.regularizers.add(
204             artm.TopicSelectionThetaRegularizer(name=name,
205                                                         tau=tau)
206         )
207         self.user_regularizers[name] = tau
208     elif name == "BitermsPhiRegularizer":
209         self.model.regularizers.add(
210             artm.BitermsPhiRegularizer(name=name, tau=tau)
211         )
212         self.user_regularizers[name] = tau
213     elif name == "BackgroundTopicsRegularizer":
214         self.model.regularizers.add(
215             artm.BackgroundTopicsRegularizer(name=name,
216                                                         tau=tau)
217         )
218         self.user_regularizers[name] = tau
219     else:
220         print(
221             "Регуляризатора {0} нет! Проверьте корректность н
222             азвания!".
223             format(name)
224         )
225
226 def add_regularizers(self, regularizers: dict[str, float])
227     -> None:
228     for regularizer in regularizers:
229         self.add_regularizer(regularizer,
230                             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,
233             num_collection_passes=1
234         )
235         self.__calc_metrics__()
236         self.perplexity_by_epoch.append(self.perplexity)
237         self.coherence_by_epoch.append(self.coherence)
238         self.topic_purities_by_epoch.append(self.topic_purities)
239
240     if epoch > 0:
241         change_perplexity_by_percent = abs(
242             self.perplexity_by_epoch[epoch - 1] -
243             self.perplexity_by_epoch[epoch]
244         ) / (self.perplexity_by_epoch[epoch - 1] +
245             self.epsilon) * 100
246
247         change_coherence_by_percent =
248             abs(self.coherence_by_epoch[epoch - 1] -
249                 self.coherence_by_epoch[epoch]) / \
250                 (self.coherence_by_epoch[epoch - 1] +
251                     self.epsilon)
252                 * 100
253
254         change_topics_purity_by_percent = abs(
255             self.topic_purities_by_epoch[epoch - 1] -
256             self.topic_purities_by_epoch[epoch])
257             / \
258             (self.topic_purities_by_epoch[epoch - 1] +
259                 self.epsilon)
260                 * 100
261
262     if change_perplexity_by_percent <
263         self.plateau_perplexity and
264         change_coherence_by_percent <

```

```

                self.plateau_coherence and
                change_topics_purity_by_percent <
                self.plateau_topics_purity:
251             break
252
253     def get_perplexity(self) -> float:
254         return self.perplexity
255
256     def get_perplexity_by_epochs(self) -> list[float]:
257         return self.perplexity_by_epoch
258
259     def print_perplexity_by_epochs(self) -> None:
260         plt.plot(
261             range(len(self.perplexity_by_epoch)),
262             self.perplexity_by_epoch,
263             label="perplexity"
264         )
265         plt.title("График перплексии")
266         plt.xlabel("Epoch")
267         plt.ylabel("Perplexity")
268         plt.legend()
269         plt.show()
270
271     def get_coherence(self) -> float:
272         return self.coherence
273
274     def get_coherence_by_epochs(self) -> list[float]:
275         return self.coherence_by_epoch
276
277     def print_coherence_by_epochs(self) -> None:
278         plt.plot(
279             range(len(self.coherence_by_epoch)),
280             self.coherence_by_epoch,
281             label="coherence"
282         )
283         plt.title("График когерентности")
284         plt.xlabel("Epoch")
285         plt.ylabel("Coherence")
286         plt.legend()
287         plt.show()
288

```

```

289     def get_topic_purities(self) -> float:
290         return self.topic_purities
291
292     def get_topic_purities_by_epochs(self) -> list[float]:
293         return self.topic_purities_by_epoch
294
295     def print_topic_purities_by_epochs(self) -> None:
296         plt.plot(
297             range(len(self.topic_purities_by_epoch)),
298             self.topic_purities_by_epoch,
299             label="topic purities"
300         )
301         plt.title("График чистоты тем")
302         plt.xlabel("Epoch")
303         plt.ylabel("Topics purity")
304         plt.legend()
305         plt.show()
306
307     def get_model(self):
308         return self.model
309
310     def save_model(self, dir_model: str =
311         ". / drive / MyDrive / model") -> None:
312         self.model.dump_artm_model(dir_model)

```

Листинг 28: 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] =
9                                     ("num_document_passes",
10                                    3, 7),
11         num_collection_passes: tuple[str, int,
12                                     int] =
13                                     ("num_collection_passes",

```

```

3, 7),
11     regularizers: dict[str, tuple[str, float, float]] = {
12         "SmoothSparseThetaRegularizer": ('tau_theta', -2.0,
13         2.0),
14         "SmoothSparsePhiRegularizer": ('tau_phi', -2.0, 2.0)
15     },
16     class_ids: dict[str, float] = {"@default_class": 1.0}
17 ):
18     self.data = data.copy(deep=True)
19     self.n_trials = n_trials
20     self.num_topics = num_topics
21     self.num_document_passes = num_document_passes
22     self.num_collection_passes = num_collection_passes
23     self.regularizers = regularizers
24     self.class_ids = class_ids
25
26     self.robast_scaler = RobustScaler()
27
28 def __objective__(self, trial) -> tuple[float, float, float]:
29     num_topics = trial.suggest_int(
30         self.num_topics[0], self.num_topics[1],
31         self.num_topics[2]
32     )
33     num_document_passes = trial.suggest_int(
34         self.num_document_passes[0],
35         self.num_document_passes[1],
36         self.num_document_passes[2]
37     )
38     num_collection_passes = trial.suggest_int(
39         self.num_collection_passes[0],
40         self.num_collection_passes[1],
41         self.num_collection_passes[2]
42     )
43     tau_theta = trial.suggest_float(
44         self.regularizers["SmoothSparseThetaRegularizer"][0],
45         self.regularizers["SmoothSparseThetaRegularizer"][1],
46         self.regularizers["SmoothSparseThetaRegularizer"][2]
47     )
48     tau_phi = trial.suggest_float(
49         self.regularizers["SmoothSparsePhiRegularizer"][0],
50         self.regularizers["SmoothSparsePhiRegularizer"][1],
51         self.regularizers["SmoothSparsePhiRegularizer"][2]
52     )

```



```

47         self.regularizers["SmoothSparsePhiRegularizer"]][2]
48     )
49     regularizers = {
50         "SmoothSparseThetaRegularizer": tau_theta,
51         "SmoothSparsePhiRegularizer": tau_phi
52     }
53     class_ids = self.class_ids
54
55     model = My_BigARTM_model(
56         data=self.data,
57         num_topics=num_topics,
58         num_document_passes=num_document_passes,
59         class_ids=class_ids,
60         num_collection_passes=num_collection_passes,
61         regularizers=regularizers
62     )
63     model.calc_model()
64
65     return model.get_perplexity(), model.get_coherence(
66     ), model.get_topic_purities()
67
68     def __select_best_trial__(self, study, weights):
69         """Выбирает trial с минимальной взвешенной суммой метрик
70             ."""
71         params_and_metrics = [
72             (trial.params, trial.values) for trial in
73             study.best_trials
74         ]
75         metrics = np.array([item[1] for item in
76             params_and_metrics])
77
78         scaled_metrics = np.zeros_like(metrics)
79         for i in range(metrics.shape[1]):
80             scaler = RobustScaler()
81             scaled_column = scaler.fit_transform(metrics[:,
82                 i].reshape(-1, 1)
83                 ).flatten()
84
85             if weights[i] < 0:
86                 scaled_column = -scaled_column
87             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:
91            sum(trial[2]))
92
93 def optimizer(self):
94     study = optuna.create_study(
95         directions=["minimize", "maximize", "maximize"]
96     )
97     study.optimize(self.__objective__,
98                   n_trials=self.n_trials)
99
100     best_trial = self.__select_best_trial__(study,
101                                           weights=[1, -1, -1])
102
103     best_params = best_trial[0]
104
105     num_topics = best_params["num_topics"]
106     num_document_passes = best_params["num_document_passes"]
107     num_collection_passes =
108         best_params["num_collection_passes"]
109     tau_theta = best_params["tau_theta"]
110     tau_phi = best_params["tau_phi"]
111
112     print("best params:")
113     print(f"num topics = {num_topics}; num document passes =
114           {num_document_passes}; \num collection passes =
115           {num_collection_passes}; tau theta = {tau_theta};
116           tau phi = {tau_phi}.")
117
118     final_model = My_BigARTM_model(
119         data=self.data,
120         num_topics=num_topics,
121         num_document_passes=num_document_passes,
122         num_collection_passes=num_collection_passes,
123         regularizers={

```

```

118         "SmoothSparseThetaRegularizer": tau_theta,
119         "SmoothSparsePhiRegularizer": tau_phi
120     },
121     class_ids={"@default_class": 1.0}
122 )
123 final_model.calc_model()
124
125 self.model = final_model
126
127 def get_model(self) -> My_BigARTM_model:
128     return self.model
129
130 def save_model(self, path_model: str =
131     "./drive/MyDrive/model") -> None:
132     self.model.model.dump_artm_model(path_model)
133
134 def save_phi(self, path_phi: str =
135     "./drive/MyDrive/phi.xlsx") -> None:
136     self.model.model.get_phi().to_excel(path_phi)
137
138 def save_theta(
139     self, path_theta: str = "./drive/MyDrive/theta.xlsx"
140 ) -> None:
141     self.model.model.get_theta().T.to_excel(path_theta)

```

Листинг 29: Полный код класса Hyperparameter_optimizer