

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

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

Кафедра математической кибернетики и компьютерных наук

ТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ НОВОСТЕЙ
КУРСОВАЯ РАБОТА

студента 3 курса 351 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 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Математические основы тематического моделирования	5
1.1 Основная гипотеза тематического моделирования	5
1.2 Аксиоматика тематического моделирования	5
1.3 Задача тематического моделирования	6
1.4 Решение обратной задачи	7
1.4.1 Лемма о максимизации функции на единичных симплексах	7
1.4.2 Сведение обратной задачи к задаче максимизации функ- ционала	8
1.4.3 Аддитивная регуляризация тематических моделей	9
1.4.4 E-M алгоритм	9
1.5 Регуляризаторы в тематическом моделировании	10
1.5.1 Дивергенция Кульбака-Лейблера	10
1.5.2 Регуляризатор сглаживания	11
1.5.3 Регуляризатор разреживания	12
1.5.4 Регуляризатор декоррелирования тем	12
1.6 Оценка качества моделей тематического моделирования	13
1.6.1 Правдоподобия и перплексия	14
1.6.2 Когерентность	14
1.6.3 Разреженность	15
1.6.4 Чистота темы	15
1.6.5 Контрастность темы	15
2 Тематическое моделирование новостей	16
2.1 Предобработка текстов	16
2.1.1 Токенизация, перевод в нижний регистр и удаление неал- фавитных символов	16
2.1.2 Удаление стоп-слов	17
2.1.3 Лемматизация	18
2.1.4 Создание N-грамм	19
2.2 Статистика по данным	19
2.2.1 Создание тематической модели с помощью библиотеки BigARTM	20
2.3 PLSA (модель без регуляризаторов)	23

2.4	LDA (модель с регуляризатором сглаживания)	25
2.5	Модель с регуляризатором разреживания	26
2.6	Модель с регуляризатором декоррелирования	29
2.7	Выбор лучшей модели	31
ЗАКЛЮЧЕНИЕ		33
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		34
Приложение А	Код программы подготовки данных	36
Приложение Б	Код программы PLSA модели	39
Приложение В	Код программы LDA модели	42
Приложение Г	Код программы модели с регуляризатором разреживания	45
Приложение Д	Код программы модели с регуляризатором декоррелирования	50
Приложение Е	Ссылка на ноутбук с программой	55

ВВЕДЕНИЕ

С ростом объёмов информации в современном мире умение классифицировать и структурировать данные становится необходимым для их эффективного поиска и изучения. Физически невозможно найти нужные сведения, просто перебирая все ресурсы подряд, поэтому возникает острая потребность в тематическом поиске и классификации данных.

Тематическое моделирование призвано решить эту проблему. Оно позволяет быстро и эффективно автоматически разбивать большие объёмы информации по темам, упрощая процесс поиска и анализа данных.

1 Математические основы тематического моделирования

1.1 Основная гипотеза тематического моделирования

Тематическое моделирование — это метод анализа текстовых данных, который позволяет выявлять 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].

1.2 Аксиоматика тематического моделирования

Каждый текст можно количественно охарактеризовать. Вот 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: $p(w|d, t) = p(w|t)$.

Вышеперечисленные характеристики, гипотезы и аксиомы составляют основу тематического моделирования и являются достаточными для построения тематической модели. [1–4].

1.3 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 генерируем терм $p(w|d, t)$.

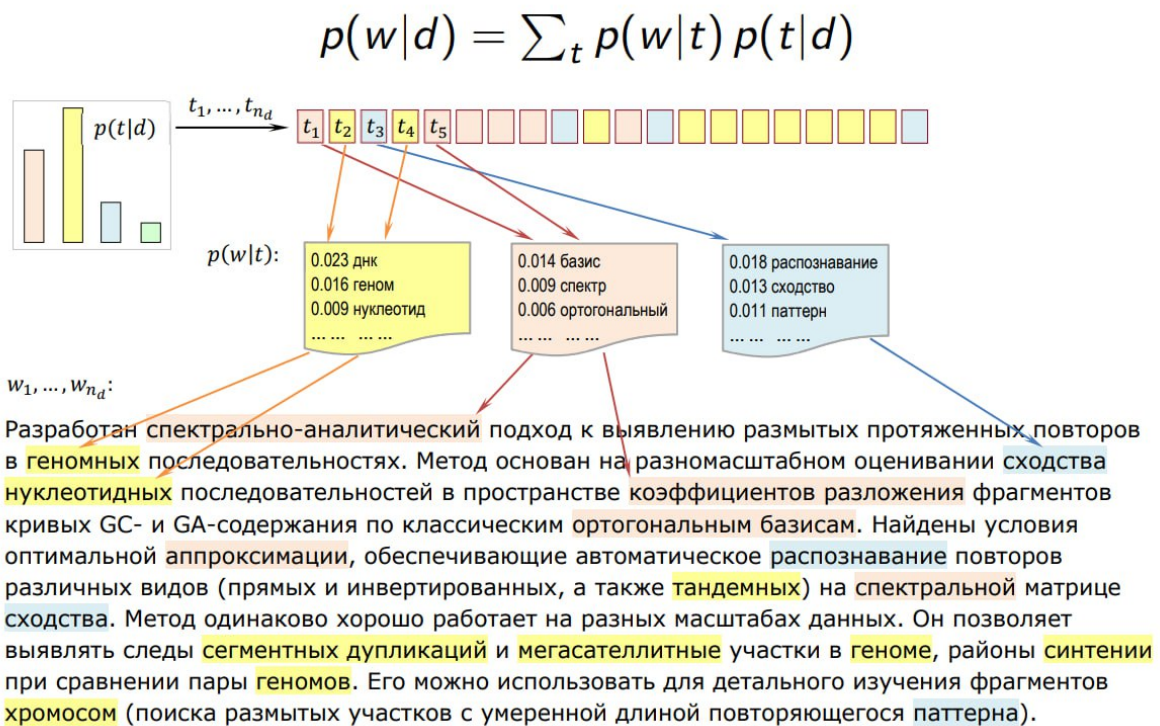


Рисунок 1 – Алгоритм формирования документа

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

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

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

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

Обратную задачу можно представить в виде стохастического матричного разложения 2.

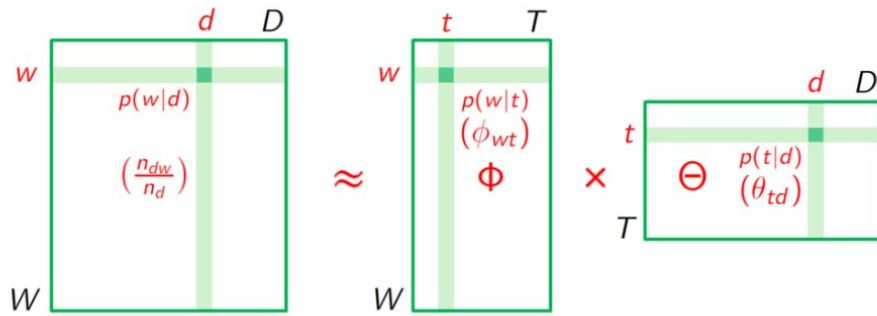


Рисунок 2 – Стохастическое матричное разложение

Таким образом, тематическое моделирование ищет величину $p(w|d)$ [1–4].

1.4 Решение обратной задачи

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

1.4.1 Лемма о максимизации функции на единичных симплексах

Перед тем как перейти к решению обратной задачи, сформулируем лемму, которая поможет нам в этом процессе.

Введём операцию нормировки вектора:

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

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 [1–3].

1.4.2 Сведение обратной задачи к задаче 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.

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

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 функции [1–4].

1.4.3 Аддитивная регуляризация тематических моделей

Задача [?] не соответствует критериям корректно поставленной задачи 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 $t\tau_i$, $i = 1, \dots, k$.

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 [1–3].

1.4.4 Е-М алгоритм

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

Воспользуемся леммой о максимизации функции на единичных симплексах 1.4.1 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].

1.5 Регуляризаторы в тематическом моделировании

В этом разделе будут рассмотрены некоторые возможные варианты регуляризаторов.

1.5.1 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 [1, 5, 6].

1.5.2 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 1.5.1 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-флгоритм 11 в соответствии с полученной формулой:

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

Таким образом был получен модифицированный ЕМ-алгоритм соответствующий модели LDA [1, 4–6].

1.5.3 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 1.5.1 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:

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

Перепишем ЕМ-алгоритм 11 в соответствии с полученной формулой:

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

Таким образом был получен модифицированный ЕМ-алгоритм, разреживающий матрицы Φ и Θ [1, 4–6].

1.5.4 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, 5, 6].

1.6 Оценка качества моделей тематического моделирования

После обучения 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) Статистический тест условной независимости.

Поскольку оценка по внешним критериям невозможна в рамках данной работы, сосредоточимся на внутренних критериях оценки, которые можно вычислять автоматически [1, 7, 8].

1.6.1 Правдоподобия 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 [1, 4, 7, 8].

1.6.2 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} , $PMI(u, v) = \ln \frac{|D|N_{uv}}{N_u N_v}$ — пото-
чечная взаимная информация, N_{uv} — число документов, в которых слова u, v хотя бы один раз встречаются рядом (расстояние определяется отдельно), N_u — число документов, в которых u встретился хотя бы один раз.

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

Сама когерентность берёт самые часто встречающиеся слова из тем, и вычисляет для каждой пары из них насколько они часто встречаются, соответ-

ственно, чем выше будет значение взаимовстречаемости, тем лучше [1, 7, 8].

1.6.3 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 [1, 4, 7, 8].

1.6.4 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, напр

Данная характеристика показывает как вероятноотносится ядро темы к фоновым словам темы, следовательно, чем больше вероятность ядра, тем лучше [1, 4, 7, 8].

1.6.5 Контрастность темы

Контрастность темы:

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

Данная характеристика показывает насколько часто слова из ядра темы встречаются в других темах, очевидно, что чем меньше ядро будет встречаться в других темах, тем лучше [1, 4, 7, 8].

2 Тематическое моделирование новостей

В данном разделе будет выполнено тематическое моделирование новостей новостного сайта ВШЭ.

Датасет был получен методами парсинга с помощью языка python и библиотек `beuatifulsoap4` и `selenium`.

2.1 Предобработка текстов

Перед любым моделированием данные нужно подготовить. Вот стандартный набор предобработки текстов для тематического моделирования:

- токенизация;
- перевод текста в нижний регистр;
- удаление неалфавитных символов;
- удаление стоп слов;
- лемматизация;
- создание n-грамм.

После выполнения вышеописанных операций можно будет приступить к самому тематическому моделированию [9–11].

2.1.1 Токенизация, перевод в нижний регистр и удаление неалфавитных символов

Токенизация — это разделение текста на составные части — токены (предложения и слова).

Провести токенизацию можно с помощью средств языка python, библиотека `nlTK`. За токенизацию отвечают команды:

```
# разделить текст на предложения
nlTK.sent_tokenize(<sentences>)
# разделить предложение на слова
nlTK.word_tokenize(<sentence>)
```

После того как текст поделен на слова, нужно перевести все слова в нижний регистр, так как семантическое значение слов, чаще всего, не зависит от регистра. Перевод в нижний регистр можно с помощью стандартных средств языка python:

```
# перевести текст в нижний регистр
<text>.lower()
```


После перевода в нижний регистр нужно удалить все семантически незначимые символы, в данном случае будем рассматривать в качестве таких символов все символы, не совпадающие с символами русского и английского алфавитов. Чтобы провести удаление неалфавитных символов достаточно средств языка python:

```
new_word = ''
# перебираем символы некоторого слова
for symbol in word:
    # если символ принадлежит русскому или английскому алфавитам
    if ( symbol >= 'a' and symbol <= 'z'
        or symbol >= 'а' and symbol <= 'я' ):
        # добавляем символ в новое слово
        new_word += symbol
```

Таким образом, получим разбитый на слова текст, не содержащий неалфавитных символов [9–11].

2.1.2 Удаление стоп-слов

Стоп-слова — это слова, которые не несут смысловой нагрузки в рамках, некоторой темы.

Любой текст содержит большое количество слов общей тематики — стоп-слов. Такие слова, для улучшения качества модели, можно удалить, так как такие слова не несут семантической нагрузки, то будут только сбивать модель.

Чтобы удалить стоп-слова можно воспользоваться библиотеки nltk языка python:

```
new_words = []
# перебираем список слов
for word in words:
    # проверяем какому алфавиту принадлежат символы слова
    if re.match(' [а-я]', word):
        # если слово не принадлежит списку стоп слов
        if word not in (stopwords.words(' russian ')):
            # добавляем слово в новый список слов
            new_words.append(word)
```

```

elif re.match( '[a-z]', word):
    # если слово не принадлежит списку стоп слов
    if word not in stopwords.words( 'english '):
        # добавляем слово в новый список слов
        new_words.append(word)

```

Таким образом, получим список слов, в котором будет отсутствовать большинство стоп-слов [9–11].

2.1.3 Лемматизация

Лемматизация — процесс приведения слова к его начальной форме.

Так как семантическое значение слова для темы не зависит от его формы и падежа, то перед обучением модели важно привести все слова в начальную форму, сделать это можно с помощью библиотек `nltk` и `pymorphy2` языка `python`:

```

# создаём лемматизаторы
lemm_nltk = WordNetLemmatizer()
lemm_pymorphy2 = pymorphy2.MorphAnalyzer()

new_words = []
# перебираем список слов
for word in words:
    # проверяем какому алфавиту принадлежат символы слова
    if re.match( '[a-я]', word):
        # лемматизируем слово на русском и добавляем его
        # в новый список слов
        new_words.append(lemm_pymorphy2.parse(word)[0].normal_form)
    elif re.match( '[a-z]', word):
        # лемматизируем слово на английском и добавляем его
        # в новый список слов
        new_words.append(lemm_nltk.lemmatize(word))

```

Таким образом, получим список слов, приведённых к их начальной форме [9–11].

2.1.4 Создание N-грамм

N-грамма — это склеивание слов в словосочетание, слов может быть несколько.

Часто слова в теме встречаются в парах или тройках подряд, тогда, если склеить слова в N-грамм, то качество и интерпретируемость модели может вырасти.

Сделать N-граммы можно средствами библиотеки `nltk` языка `python`:

```
n_gramms = []  
# перебираем предложения и составляем список n-грамм  
for sentence in sentences:  
    # делаем n граммы и добавляем их в список n-грамм  
    n_gramms.append(sentence.split(' '), <n>)
```

Таким образом, получим список n-грамм, составленный из начального списка слов [9–11].

2.2 Статистика по данным

Чтобы корректнее строить тематические модели нужно знать количественные характеристики данных, получить такие данные можно удобно с помощью библиотек `pandas` и `pumpru` языка `python`.

Перечислим некоторые количественные характеристики, характеризующие наш датасет (перед вычислениями проводилась предобработка данных, исключая лемматизацию):

- количество новостей в датасете: 15768;
- средняя длина документа (в словах): 34.6;
- медианная длина документа (в словах): 29;
- двадцать наиболее популярных слов датасета:
 1. вшэ: 11437;
 2. ниу: 5559;
 3. экономики: 4783;
 4. россии: 2955;
 5. высшей: 2498;
 6. школы: 2293;
 7. гувшэ: 2107;
 8. вышки: 2100;

9. года: 2070;
10. развития: 2065;
11. исследований: 1876;
12. образования: 1858;
13. году: 1737;
14. программы: 1644;
15. студентов: 1481;
16. факультета: 1428;
17. университета: 1399;
18. института: 1307;
19. школа: 1303;
20. рамках: 1286.

По этим данным можно сделать следующие выводы:

- общий объём данных весьма не велик, что может усложнить построение тематической модели;
- короткая медианная длина документов тоже приведёт к снижению качества модели, так как тематическое моделирование происходит на текстах большей длины;
- среди 20 наиболее популярных слов датасета явно присутствуют слова общей лексики (стоп-слова), которые необходимо будет удалить на этапе удаления стоп-слов.

Программу вычисляющую количественные характеристики датасета можно найти в приложениях [10, 12, 13].

2.2.1 Создание тематической модели с помощью библиотеки BigARTM

Блок тематических моделей уже реализован в библиотеке BigARTM, которую можно использовать на языке python.

Модели BigARTM для своей работы требуют особого типа данных — `vowpal_wabbit`. Данный тип данных представляет из себя следующую конструкцию.

<code>doc_1</code>	слово документа 1	слово документа 1	...	слово документа 1
<code>doc_2</code>	слово документа 2	слово документа 2	...	слово документа 2
...
<code>doc_n</code>	слово документа n	слово документа n	...	слово документа n

Преобразовать excel таблицу с новостями к данному формату можно с помощью стандартных средств языка python и библиотеки pandas:

```
# считываем excel таблицу в pandas DataFrame
data = pd.read_excel('news.xlsx')
# открываем файл для записи vowpal_wabbit файла
f = open(<path>, 'w')
# проходимся по строкам DataFrame
for string in range(data.shape[0]):
    # записываем отдельную новость в файл как отдельный документ
    f.write( 'doc_{0}'.format(string)
            + data.loc[string, 'title']
            + ' '
            + data.loc[string, 'content']
            + '\n')
# после записи закрываем файл
f.close()
```

Чтобы передать данные из vowpal_wabbit файла на обучение необходимо создать батчи, они удобно будут постепенно загружаться в оперативную память по мере необходимости и передаваться на обучение, кроме того батчи автоматически вычисляют для себя словарь, который также необходим при обучении. Создать батчи можно следующим образом:

```
# data_path - путь к vowpal_wabbit файлу
# data_format - формат загружаемого файла - vowpal_wabbit
# batch_size - количество документов в одном батче
# target_folder - папка, в которую батчи сохраняются
bv = arlm.BatchVectorizer( data_path = 'vw.txt',
                          data_format = 'vowpal_wabbit',
                          batch_size=3000,
                          target_folder='batches' )
```

Наконец, можно создать саму модель, делается это следующим образом:

```
# num_topics - количество тем
# num_document_passes - количество проходов
```

```

# по каждому документу (новости)
# dictionary - словарь
# class_ids - веса для модальностей
# создание модели
model = artm.ARTM( num_topics=7,
                  num_document_passes=3,
                  dictionary=bv.dictionary,
                  class_ids={ '@default_class': 1.0})

# добавление метрик
model.scores.add( artm.PerplexityScore( name= 'perplexity',
                                       dictionary=bv.dictionary ) )

# сохранения топа слов для каждой темы
model.scores.add(artm.TopTokensScore(name= 'top-tokens', num_tokens=10))

# добавление регуляризаторов, например, декоррелятора
# tau - коэффициент регуляризации
model.regularizers.add( artm.DecorrelatorPhiRegularizer( name= 'decorrelator',
                                                         tau=2e7 ) )

```

Метрик качества, а также регуляризаторов можно добавить сразу несколько.

После создания модели её нужно обучить, сделать это можно следующим образом:

```

for _ in range(<num_passes>):
    model.fit_offline(bv, num_collection_passes=1)

```

Чтобы оценить модель можно запросить значение метрик и список слов для тем:

```

# запрашиваем последнее значение перплексии
perplexity = model.score_tracker[ 'perplexity' ].last_value
# запрашиваем массив самых популярных слов для каждой темы
top_tokens = model.score_tracker[ 'top-tokens' ].last_value

```

Таким образом, получим обученную тематическую модель [9, 10, 12, 14, 15].

2.3 PLSA (модель без регуляризаторов)

Модели PLSA соответствует EM-алгоритм без регуляризаторов 11. Данную модель можно создать средствами библиотеки BigARTM следующим образом:

```
model = artm.ARTM( num_topics=param1,
                   num_document_passes=param3,
                   dictionary=bv.dictionary,
                   class_ids={ '@default_class': 1.0} )

model.scores.add( artm.PerplexityScore( name= 'perplexity',
                                       dictionary=bv.dictionary ) )
model.scores.add(artm.SparsityPhiScore(name= 'sparsity_phi_score' ))
model.scores.add(artm.SparsityThetaScore(name= 'sparsity_theta_score' ))
model.scores.add(artm.TopTokensScore(name= 'top-tokens', num_tokens=10))
```

Для оценки качества модели выбраны такие характеристика как перплексия и разреженность (по матрицам Φ и Θ).

На место параметров модели (param1, param3) в функции создания и обучения (обучение будет происходить на простых словах, биграммах и триграммах), которую можно увидеть в приложениях, будут подставляться значения из некоторого набора, затем модель будет обучаться param2 раз. После обучения будет выведена соответствующая таблица с результатами по моделям.

В результате получаем следующую таблицу:

	model	num_topics	num_collection_passes	num_doc_passes	n-grams	perplexity	phi_sparsity	theta_sparsity
0	PLSA	6	10	2	1-gramm	1775.065796	0.006955	0.000000
1	PLSA	6	10	2	2-gramm	37625.476562	0.403637	0.003351
2	PLSA	6	10	2	3-gramm	63038.718750	0.730458	0.095848
3	PLSA	6	10	4	1-gramm	1572.347656	0.127451	0.000000
4	PLSA	6	10	4	2-gramm	35219.101562	0.695063	0.006131
5	PLSA	6	10	4	3-gramm	61639.851562	0.804921	0.107327
6	PLSA	6	15	2	1-gramm	1604.354980	0.116235	0.000000
7	PLSA	6	15	2	2-gramm	36661.375000	0.695785	0.005644
8	PLSA	6	15	2	3-gramm	62521.285156	0.807227	0.105424
9	PLSA	6	15	4	1-gramm	1487.769897	0.327868	0.000000
10	PLSA	6	15	4	2-gramm	34896.421875	0.760885	0.006902
11	PLSA	6	15	4	3-gramm	61521.820312	0.813839	0.110266
12	PLSA	6	24	2	1-gramm	1519.599487	0.381967	0.000000
13	PLSA	6	24	2	2-gramm	36354.816406	0.767654	0.006511
14	PLSA	6	24	2	3-gramm	62446.078125	0.815319	0.108595
15	PLSA	6	24	4	1-gramm	1445.069336	0.509252	0.000127
16	PLSA	6	24	4	2-gramm	34762.117188	0.775528	0.007515
17	PLSA	6	24	4	3-gramm	61484.753906	0.815943	0.111914
18	PLSA	8	10	2	1-gramm	1673.471313	0.011083	0.000000
19	PLSA	8	10	2	2-gramm	30920.789062	0.496010	0.004344
20	PLSA	8	10	2	3-gramm	48795.281250	0.792697	0.116018
21	PLSA	8	10	4	1-gramm	1462.351074	0.172950	0.000000
22	PLSA	8	10	4	2-gramm	28756.289062	0.761618	0.008221
23	PLSA	8	10	4	3-gramm	47652.964844	0.850665	0.130533
24	PLSA	8	15	2	1-gramm	1499.914062	0.161026	0.000000
25	PLSA	8	15	2	2-gramm	30063.355469	0.763374	0.007515
26	PLSA	8	15	2	3-gramm	48310.164062	0.852704	0.128615
27	PLSA	8	15	4	1-gramm	1377.463989	0.392327	0.000024
28	PLSA	8	15	4	2-gramm	28477.035156	0.813935	0.009243
29	PLSA	8	15	4	3-gramm	47542.296875	0.858260	0.135100
30	PLSA	8	24	2	1-gramm	1410.077759	0.451992	0.000016
31	PLSA	8	24	2	2-gramm	29788.367188	0.819743	0.009132
32	PLSA	8	24	2	3-gramm	48231.464844	0.859705	0.133617
33	PLSA	8	24	4	1-gramm	1329.057007	0.571340	0.000262
34	PLSA	8	24	4	2-gramm	28339.804688	0.825783	0.010100
35	PLSA	8	24	4	3-gramm	47509.140625	0.860247	0.137953

Рисунок 3 – Результат работы модели PLSA

Наилучшие значения перплексии достигаются при 8 темах, 24 проходах по коллекции и 4 проходах по каждому документу. Скорее всего модель без регуляризаторов не сильно подходит для тематического моделирования новостей, так как темы, скорее всего семантически близки друг к другу, поэтому их стоит разреживать.

Вариант с N-граммами не прошёл, скорее всего, из-за топорности их создания библиотекой nltk [1, 9, 10, 14, 15].

2.4 LDA (модель с регуляризатором сглаживания)

Модели LDA соответствует EM-алгоритм с регуляризатором сглаживания 15. Создать модель можно следующим образом:

```
model = artm.ARTM( num_topics=param1,
                  num_document_passes=param3,
                  dictionary=bv.dictionary,
                  class_ids={ '@default_class': 1.0 } )
model.regularizers.add( artm.SmoothSparsePhiRegularizer( name='smooth',
                                                         tau=tau ) )

model.scores.add( artm.PerplexityScore( name='perplexity',
                                       dictionary=bv.dictionary ) )
model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_score'))
model.scores.add(artm.SparsityThetaScore(name='sparsity_theta_score'))
model.scores.add(artm.TopTokensScore(name='top-tokens', num_tokens=10))
```

Для оценки качества модели выбраны такие же характеристики как и у модели PLSA.

На место параметров модели (param1, param3, tau > 0) в функции создания и обучения, которую можно увидеть в приложениях, будут подставляться значения из некоторого набора, затем модель будет обучаться param2 раз. После обучения будет выведена соответствующая таблица с результатами по моделям.

В результате получаем следующую таблицу:

	model	num_topics	num_collection_passes	num_doc_passes	tau	n-grams	perplexity	phi_sparsity	theta_sparsity
0	LDA	6	10	7	0.5	1-gramm	1710.850220	0.0	0.0
1	LDA	6	10	7	0.5	2-gramm	67566.757812	0.0	0.0
2	LDA	6	10	7	0.5	3-gramm	153792.531250	0.0	0.0
3	LDA	6	15	7	0.5	1-gramm	1641.057129	0.0	0.0
4	LDA	6	15	7	0.5	2-gramm	65737.039062	0.0	0.0
5	LDA	6	15	7	0.5	3-gramm	152687.625000	0.0	0.0
6	LDA	6	24	7	0.5	1-gramm	1600.610229	0.0	0.0
7	LDA	6	24	7	0.5	2-gramm	64311.207031	0.0	0.0
8	LDA	6	24	7	0.5	3-gramm	152265.625000	0.0	0.0
9	LDA	7	10	7	0.5	1-gramm	1693.715332	0.0	0.0
10	LDA	7	10	7	0.5	2-gramm	67011.507812	0.0	0.0
11	LDA	7	10	7	0.5	3-gramm	151199.500000	0.0	0.0
12	LDA	7	15	7	0.5	1-gramm	1619.568848	0.0	0.0
13	LDA	7	15	7	0.5	2-gramm	65138.832031	0.0	0.0
14	LDA	7	15	7	0.5	3-gramm	149755.437500	0.0	0.0
15	LDA	7	24	7	0.5	1-gramm	1577.376587	0.0	0.0
16	LDA	7	24	7	0.5	2-gramm	63728.996094	0.0	0.0
17	LDA	7	24	7	0.5	3-gramm	149215.875000	0.0	0.0
18	LDA	8	10	7	0.5	1-gramm	1674.479980	0.0	0.0
19	LDA	8	10	7	0.5	2-gramm	66994.335938	0.0	0.0
20	LDA	8	10	7	0.5	3-gramm	150138.484375	0.0	0.0
21	LDA	8	15	7	0.5	1-gramm	1589.150024	0.0	0.0
22	LDA	8	15	7	0.5	2-gramm	64756.683594	0.0	0.0
23	LDA	8	15	7	0.5	3-gramm	147781.640625	0.0	0.0
24	LDA	8	24	7	0.5	1-gramm	1552.519653	0.0	0.0
25	LDA	8	24	7	0.5	2-gramm	63383.753906	0.0	0.0
26	LDA	8	24	7	0.5	3-gramm	147131.609375	0.0	0.0

Рисунок 4 – Результат работы модели LDA

Модель LDA, ожидаемо, показывает не лучшие результаты в виду особенностей коллекции новостей (семантическая близость новстей не нуждается в регуляризаторе сглаживания). [1, 9, 10, 14, 15].

2.5 Модель с регуляризатором разреживания

В данном случае модели соответствует ЕМ-алгоритм с регуляризатором разреживания 18. Создать модель можно следующим образом:

```
model = artm.ARTM( num_topics=param1,
                   num_document_passes=param3,
```

```

dictionary=bv.dictionary,
class_ids={ '@default_class': 1.0} )
model.regularizers.add( artm.SmoothSparsePhiRegularizer( name='smooth',
                                                         tau=tau ) )

model.scores.add( artm.PerplexityScore( name='perplexity',
                                       dictionary=bv.dictionary ) )
model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_score'))
model.scores.add(artm.SparsityThetaScore(name='sparsity_theta_score'))
model.scores.add(artm.TopTokensScore(name='top-tokens', num_tokens=10))

```

Характеристики для оценки качества используются всё те же.

На место параметров модели (param1, param3, $\tau < 0$) в функции создания и обучения, которую можно увидеть в приложениях, будут подставляться значения из некоторого набора, затем модель будет обучаться param2 раз. После обучения будет выведена соответствующая таблица с результатами по моделям.

В результате получаем следующую таблицу:

	model	num_topics	num_collection_passes	num_doc_passes	tau	n-grams	perplexity	phi_sparsity	theta_sparsity
0	SPARSE	6	10	2	-0.5	1-gramm	1231.503052	0.640000	0.071823
1	SPARSE	6	10	2	-0.5	2-gramm	1538.760498	0.951833	0.300651
2	SPARSE	6	10	2	-0.5	3-gramm	188.688263	0.979778	0.531911
3	SPARSE	6	10	3	-0.5	1-gramm	1157.054565	0.659647	0.200438
4	SPARSE	6	10	3	-0.5	2-gramm	1777.264771	0.944017	0.381501
5	SPARSE	6	10	3	-0.5	3-gramm	241.116043	0.972879	0.563029
6	SPARSE	6	15	2	-0.5	1-gramm	1158.813599	0.715806	0.114388
7	SPARSE	6	15	2	-0.5	2-gramm	1527.366699	0.954007	0.305016
8	SPARSE	6	15	2	-0.5	3-gramm	188.333817	0.980124	0.529828
9	SPARSE	6	15	3	-0.5	1-gramm	1114.064941	0.714879	0.229537
10	SPARSE	6	15	3	-0.5	2-gramm	1767.818481	0.946003	0.377769
11	SPARSE	6	15	3	-0.5	3-gramm	240.820450	0.973188	0.560767
12	SPARSE	7	10	2	-0.5	1-gramm	1182.924194	0.676651	0.106681
13	SPARSE	7	10	2	-0.5	2-gramm	1137.191650	0.963955	0.363358
14	SPARSE	7	10	2	-0.5	3-gramm	142.102112	0.986896	0.605548
15	SPARSE	7	10	3	-0.5	1-gramm	1112.731445	0.694456	0.250580
16	SPARSE	7	10	3	-0.5	2-gramm	1311.708008	0.958015	0.436861
17	SPARSE	7	10	3	-0.5	3-gramm	172.531143	0.982427	0.627401
18	SPARSE	7	15	2	-0.5	1-gramm	1110.098633	0.747720	0.154853
19	SPARSE	7	15	2	-0.5	2-gramm	1128.847412	0.965883	0.365596
20	SPARSE	7	15	2	-0.5	3-gramm	141.829971	0.987146	0.602504
21	SPARSE	7	15	3	-0.5	1-gramm	1071.746948	0.744674	0.278140
22	SPARSE	7	15	3	-0.5	2-gramm	1304.456909	0.959750	0.431452
23	SPARSE	7	15	3	-0.5	3-gramm	172.244049	0.982699	0.624737
24	SPARSE	8	10	2	-0.5	1-gramm	1145.132202	0.708129	0.135718
25	SPARSE	8	10	2	-0.5	2-gramm	872.833496	0.972098	0.419695
26	SPARSE	8	10	2	-0.5	3-gramm	116.196747	0.990707	0.663424
27	SPARSE	8	10	3	-0.5	1-gramm	1074.828125	0.725828	0.293617
28	SPARSE	8	10	3	-0.5	2-gramm	1018.269531	0.966972	0.489060
29	SPARSE	8	10	3	-0.5	3-gramm	137.886215	0.987549	0.679279
30	SPARSE	8	15	2	-0.5	1-gramm	1075.168091	0.775927	0.189482
31	SPARSE	8	15	2	-0.5	2-gramm	866.262329	0.973689	0.420464
32	SPARSE	8	15	2	-0.5	3-gramm	115.994278	0.990904	0.660681
33	SPARSE	8	15	3	-0.5	1-gramm	1035.476440	0.771970	0.318969
34	SPARSE	8	15	3	-0.5	2-gramm	1012.164307	0.968415	0.484502
35	SPARSE	8	15	3	-0.5	3-gramm	137.631042	0.987740	0.676441

Рисунок 5 – Результат работы модели с регуляризатором разреживания

Модель с регуляризатором разреживания показывает второй по качеству результат по перплексии, лучшее значение достигается при 7 темах, 3 проходах

по каждому из документов и 15 проходах по всей коллекции.

Несмотря на лучшие значения перплексии при использовании N-граммов, их не стоит использовать из-за слишком большой разреженности, так как в данном случае размер ядер будет очень маленьким [1, 9, 10, 14, 15].

2.6 Модель с регуляризатором декоррелирования

В данном случае модели соответствует ЕМ-алгоритм с регуляризатором декоррелирования 20. Создать модель можно следующим образом:

```
model = artm.ARTM( num_topics=param1,
                   num_document_passes=param3,
                   dictionary=bv.dictionary,
                   class_ids={ '@default_class': 1.0 } )
model.regularizers.add( artm.DecorrelatorPhiRegularizer( name= 'decorrelator',
                                                         tau=tau ) )

model.scores.add( artm.PerplexityScore( name= 'perplexity',
                                       dictionary=bv.dictionary ) )
model.scores.add(artm.SparsityPhiScore(name= 'sparsity_phi_score' ))
model.scores.add(artm.SparsityThetaScore(name= 'sparsity_theta_score' ))
model.scores.add(artm.TopTokensScore(name= 'top-tokens', num_tokens=10))
```

Характеристики для оценки качества используются всё те же.

На место параметров модели (param1, param3, tau) в функции создания и обучения, которую можно увидеть в приложениях, будут подставляться значения из некоторого набора, затем модель будет обучаться param2 раз. После обучения будет выведена соответствующая таблица с результатами по моделям.

В результате получаем следующую таблицу:

	model	num_topics	num_collection_passes	num_doc_passes	tau	n-grams	perplexity	phi_sparsity	theta_sparsity
0	DECOR	6	24	7	1000000.0	1-gramm	997.300354	0.249426	0.001702
1	DECOR	6	24	7	1000000.0	2-gramm	30181.234375	0.767092	0.008752
2	DECOR	6	24	7	1000000.0	3-gramm	56177.351562	0.814341	0.120698
3	DECOR	6	24	7	2000000.0	1-gramm	987.056885	0.246225	0.004936
4	DECOR	6	24	7	2000000.0	2-gramm	30541.140625	0.766294	0.010771
5	DECOR	6	24	7	2000000.0	3-gramm	56174.789062	0.814333	0.120814
6	DECOR	6	24	7	10000000.0	1-gramm	898.948303	0.360353	0.052987
7	DECOR	6	24	7	10000000.0	2-gramm	28595.830078	0.766903	0.011849
8	DECOR	6	24	7	10000000.0	3-gramm	55265.882812	0.814376	0.127347
9	DECOR	6	24	7	20000000.0	1-gramm	889.529602	0.725563	0.217360
10	DECOR	6	24	7	20000000.0	2-gramm	28333.759766	0.766618	0.013181
11	DECOR	6	24	7	20000000.0	3-gramm	55354.835938	0.814413	0.134323
12	DECOR	7	24	7	1000000.0	1-gramm	965.575134	0.268023	0.003126
13	DECOR	7	24	7	1000000.0	2-gramm	27194.945312	0.796486	0.010953
14	DECOR	7	24	7	1000000.0	3-gramm	48933.242188	0.839778	0.137865
15	DECOR	7	24	7	2000000.0	1-gramm	925.523438	0.271334	0.009151
16	DECOR	7	24	7	2000000.0	2-gramm	26743.005859	0.795559	0.013581
17	DECOR	7	24	7	2000000.0	3-gramm	49241.468750	0.839727	0.144968
18	DECOR	7	24	7	10000000.0	1-gramm	855.574219	0.542649	0.139795
18	DECOR	7	24	7	10000000.0	1-gramm	855.574219	0.542649	0.139795
19	DECOR	7	24	7	10000000.0	2-gramm	25736.072266	0.795918	0.016435
20	DECOR	7	24	7	10000000.0	3-gramm	48125.343750	0.839806	0.148791
21	DECOR	7	24	7	20000000.0	1-gramm	940.037048	0.923179	0.501060
22	DECOR	7	24	7	20000000.0	2-gramm	25049.480469	0.795690	0.018274
23	DECOR	7	24	7	20000000.0	3-gramm	47833.484375	0.839776	0.159410
24	DECOR	8	24	7	1000000.0	1-gramm	936.616882	0.285579	0.009006
25	DECOR	8	24	7	1000000.0	2-gramm	24957.750000	0.818599	0.015126
26	DECOR	8	24	7	1000000.0	3-gramm	43443.000000	0.858962	0.151081
27	DECOR	8	24	7	2000000.0	1-gramm	868.931519	0.306772	0.017646
28	DECOR	8	24	7	2000000.0	2-gramm	23935.548828	0.818830	0.016854
29	DECOR	8	24	7	2000000.0	3-gramm	42724.964844	0.858977	0.159825
30	DECOR	8	24	7	10000000.0	1-gramm	838.104187	0.727616	0.268471
31	DECOR	8	24	7	10000000.0	2-gramm	23172.703125	0.818806	0.020025
32	DECOR	8	24	7	10000000.0	3-gramm	42752.050781	0.859007	0.171027
33	DECOR	8	24	7	20000000.0	1-gramm	1095.360229	0.976109	0.713098
34	DECOR	8	24	7	20000000.0	2-gramm	22251.166016	0.818686	0.021475
35	DECOR	8	24	7	20000000.0	3-gramm	42528.093750	0.858923	0.180896

Рисунок 6 – Результат работы модели с регуляризатором сглаживания

Модель с регуляризатором декорреляции дала наилучший результат среди моделей, обусловлено это особенностями данных (семантическая близость тем).

Лучшее значение достигается при 8 темах, 24 проходах по всей коллекции и 7 проходах по каждому документу [1, 9, 10, 14, 15].


2.7 Выбор лучшей модели

Выберем по одной модели из каждого класса, обладающей наибольшим значением перплексии в своём классе, и повторно обучим их.

После этого посмотрим на топ слов для каждой из моделей и решим, темы какой из моделей лучше интерпретируются.

Приведём списки слов для каждой из моделей:


PLSA



```
['факультет', 'вшэ', 'наука', 'учёный', 'лаборатория', 'научный', 'профессор', 'исследование', 'кафедра', 'журнал']
['вышка', 'вшэ', 'год', 'день', 'ниу', 'лекция', 'центр', 'открытый', 'деловой', 'пройти']
['исследование', 'вшэ', 'ниу', 'социальный', 'центр', 'российский', 'рост', 'россия', 'год', 'труд']
['экономика', 'образование', 'международный', 'развитие', 'вшэ', 'высокий', 'школа', 'конференция', 'институт', 'научный']
['год', 'вшэ', 'олимпиада', 'высокий', 'участие', 'студент', 'ниу', 'школа', 'который', 'школьник']
['программа', 'школа', 'студент', 'вшэ', 'вышка', 'магистерский', 'новый', 'рассказать', 'ниу', 'курс']
['россия', 'политика', 'мировой', 'новый', 'экономика', 'вопрос', 'бизнес', 'экономический', 'страна', 'кризис']
['проект', 'вшэ', 'конкурс', 'статья', 'университет', 'школа', 'вуз', 'год', 'хороший', 'высокий']
```

Рисунок 7 – Ядра тем модели PLSA

LDA



```
['факультет', 'вшэ', 'наука', 'учёный', 'лаборатория', 'профессор', 'научный', 'исследование', 'кафедра', 'новый']
['год', 'вшэ', 'вышка', 'ниу', 'центр', 'день', 'лекция', 'открытый', 'деловой', 'бюллетень']
['исследование', 'социальный', 'вшэ', 'ниу', 'рост', 'российский', 'центр', 'россия', 'человек', 'труд']
['экономика', 'образование', 'международный', 'вшэ', 'развитие', 'высокий', 'школа', 'конференция', 'институт', 'научный']
['год', 'вшэ', 'олимпиада', 'высокий', 'участие', 'студент', 'который', 'школа', 'ниу', 'принять']
['программа', 'школа', 'студент', 'вышка', 'рассказать', 'вшэ', 'новый', 'магистерский', 'курс', 'компания']
['россия', 'политика', 'гшэ', 'мировой', 'экономика', 'экономический', 'новый', 'вопрос', 'бизнес', 'российский']
['проект', 'вшэ', 'статья', 'конкурс', 'университет', 'школа', 'год', 'студент', 'вуз', 'хороший']
```

Рисунок 8 – Ядра тем модели LDA

Модель с регуляризатором разреживания



```
['вшэ', 'факультет', 'наука', 'учёный', 'лаборатория', 'профессор', 'рассказать', 'научный', 'анализ', 'кафедра']
['вышка', 'вшэ', 'год', 'студент', 'университет', 'статья', 'ниу', 'день', 'место', 'рейтинг']
['исследование', 'вшэ', 'россия', 'российский', 'ниу', 'центр', 'общество', 'институт', 'развитие', 'год']
['вшэ', 'экономика', 'школа', 'международный', 'развитие', 'высокий', 'ниу', 'конференция', 'пройти', 'центр']
['вшэ', 'год', 'образование', 'конкурс', 'студент', 'олимпиада', 'высокий', 'вуз', 'ниу', 'получить']
['программа', 'школа', 'вышка', 'высокий', 'магистерский', 'университет', 'студент', 'экономика', 'обучение', 'первый']
['политика', 'россия', 'мировой', 'новый', 'вопрос', 'экономика', 'гшэ', 'бизнес', 'страна', 'кризис']
```

Рисунок 9 – Ядра тем модели с регуляризатором разреживания

Модель с регуляризатором декоррелирования


['анализ', 'журнал', 'будущее', 'опубликовать', 'социология', 'разный', 'компьютерный', 'данные', 'заместитель', 'прикладной']
['студенческий', 'ректор', 'деловой', 'этап', 'фестиваль', 'климат', 'конъюнктурный', 'кузьмин', 'промышленный', 'клуб']
['книга', 'обсудить', 'стол', 'круглый', 'дать', 'гражданский', 'сектор', 'модель', 'автор', 'помощь']
['интервью', 'решение', 'форум', 'менеджмент', 'национальный', 'представитель', 'создать', 'главный', 'разработка', 'банк']
['приём', 'бюллетень', 'кампус', 'выпуск', 'всероссийский', 'впервые', 'поступление', 'регистрация', 'поступать', 'документ']
['встреча', 'набор', 'карьера', 'неделя', 'путь', 'большой', 'открывать', 'слушатель', 'дополнительный', 'аспирантура']
['вызов', 'хотеть', 'известный', 'партнёрство', 'министр', 'ценность', 'регулирование', 'политология', 'образ', 'прогноз']
['город', 'финансовый', 'второй', 'практика', 'инновационный', 'среди', 'опыт', 'войти', 'культурный', 'лицей']

Рисунок 10 – Ядра тем модели с регуляризатором декоррелирования

Как видно из характеристик качества лучшей оказалась модель с регуляризатором декоррелирования, также самой интерпретируемой стала модель с декоррелированием. Результат хорошо коррелирует с тем, что в датасете темы семантически близки, поэтому их декорреляция хорошо сказалась на интерпретируемости тем и вывела данный тип модели в топ [1, 9, 10, 14, 15].

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены основные механизмы вероятностного тематического моделирования, включая ЕМ-алгоритм, методы регуляризации и аддитивная регуляризация тематических моделей. Также в процессе тематического моделирования новостей с сайта ВШЭ были закреплены практические навыки: предобработка данных с использованием библиотек `rumorphy2` и `nltk`, а также создание тематических моделей с помощью библиотеки `bigARTM`.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Вероятностное тематическое моделирование: теория регуляризации ARTM и библиотека с открытым исходным кодом BigARTM [Электронный ресурс]. — URL: <http://www.machinelearning.ru/wiki/images/d/d5/Voron17survey-artm.pdf> (Дата обращения 26.10.2023). Загл. с экр. Яз. рус.
- 2 Вероятностные тематические модели Лекция 1. Постановка задачи, оптимизация и регуляризация [Электронный ресурс]. — URL: <http://www.machinelearning.ru/wiki/images/6/6a/Voron24ptm-intro.pdf> (Дата обращения 4.11.2023). Загл. с экр. Яз. рус.
- 3 Тематическое моделирование. Лекция 1. [Электронный ресурс]. — URL: <https://youtu.be/sBdFG8Rl-i8?si=pEmqLSU8yJU2M3DH> (Дата обращения 4.11.2023). Загл. с экр. Яз. рус.
- 4 *Николаевич, Ш.* Вероятность-1 / Ш. Николаевич. — Москва: МЦНМО, 2021.
- 5 Вероятностные тематические модели Лекция 2. Онлайновый EM-алгоритм и аддитивная регуляризация [Электронный ресурс]. — URL: <http://www.machinelearning.ru/wiki/images/b/be/Voron24ptm-regular.pdf> (Дата обращения 18.11.2024). Загл. с экр. Яз. рус.
- 6 Тематическое моделирование. Лекция 2. [Электронный ресурс]. — URL: <https://youtu.be/bA6wS6j6akU?si=HhiEqyPQOo-PbyfX> (Дата обращения 18.11.2024). Загл. с экр. Яз. рус.
- 7 Вероятностные тематические модели Лекция 4. Оценивание качества тематических моделей [Электронный ресурс]. — URL: <http://www.machinelearning.ru/wiki/images/0/0b/Voron24ptm-quality.pdf> (Дата обращения 01.01.2024). Загл. с экр. Яз. рус.
- 8 Тематическое моделирование. Лекция 3. [Электронный ресурс]. — URL: <https://youtu.be/FfxuULD-TmU?si=n6hMcjbQTvFjUQsf> (Дата обращения 01.01.2024). Загл. с экр. Яз. рус.
- 9 Тематическое моделирование. Лекция 4. [Электронный ресурс]. — URL: <https://youtu.be/AIN00vWOJGw?si=EFgyJ5mBtCyu2Bro> (Дата обращения 01.02.2024). Загл. с экр. Яз. рус.

- 10 *Васильев, А.* Программирование на PYTHON в примерах и задачах / А. Васильев. — Москва: Эксмо, 2021.
- 11 Тематическое моделирование средствами BigARTM. [Электронный ресурс]. — URL: <https://habr.com/ru/articles/334668/> (Дата обращения 01.02.2024). Загл. с экр. Яз. рус.
- 12 User Guide [Электронный ресурс]. — URL: https://pandas.pydata.org/docs/user_guide/index.html (Дата обращения 01.02.2024). Загл. с экр. Яз. рус.
- 13 NumPy user guide [Электронный ресурс]. — URL: <https://numpy.org/doc/stable/user/index.html> (Дата обращения 01.02.2024). Загл. с экр. Яз. рус.
- 14 BigARTM. Примеры обучения моделей на Python [Электронный ресурс]. — URL: https://github.com/bigartm/bigartm-book/blob/master/ARTM_tutorial_Fun.ipynb (Дата обращения 01.02.2024). Загл. с экр. Яз. рус.
- 15 BigARTM's documentation [Электронный ресурс]. — URL: <https://docs.bigartm.org/en/stable/index.html> (Дата обращения 01.02.2024). Загл. с экр. Яз. англ.

ПРИЛОЖЕНИЕ А

Код программы подготовки данных

```
import pandas as pd
import re

!pip install nltk
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

!pip install pymorphy2
import pymorphy2

# Загрузка данных и отсечение последней строки с несущественными столбцами (link,
↪ data, tags)
news = pd.read_excel('news.xlsx')
news = news[:-1]
news = news[['title', 'content']]

# Функция для разбиения ячеек на слова
def tokenize(cell: str) -> list[str]:
    words = []

    sentences = nltk.sent_tokenize(cell)
    for sentence in sentences:
        words += nltk.word_tokenize(sentence)

    return words

# Функция для перевода слов в нижний регистр
def convert_to_lowercase(words: list[str]) -> list[str]:
    new_words = []

    for word in words:
        new_words.append(word.lower())

    return new_words
```

```
# Функция для удаления символов, отличающихся от символов русского и английского  
↪ алфавитов
```

```
def del_non_alphs(words: list[str]) -> list[str]:  
    new_words = []  
  
    for word in words:  
        new_word = ''  
  
        for symbol in word:  
            if (symbol >= 'a' and symbol <= 'z' or symbol >= 'а' and symbol <= 'я'):  
                new_word += symbol  
  
        if (len(new_word) > 0):  
            new_words.append(new_word)  
  
    return new_words
```

```
# Функция для удаления стоп слов
```

```
def del_stop_words(words: list[str]) -> list[str]:  
    new_words = []  
  
    for word in words:  
        if re.match('[а-я]', word):  
            if word not in (stopwords.words('russian') + ['вше' + 'ни']):  
                new_words.append(word)  
        elif re.match('[a-z]', word):  
            if word not in stopwords.words('english'):  
                new_words.append(word)  
  
    return new_words
```

```
# Функция лемматизации
```

```
def lemm_words(words: list[str]) -> list[str]:  
    lemm_nltk = WordNetLemmatizer()  
    lemm_pymorphy2 = pymorphy2.MorphAnalyzer()  
  
    new_words = []  
  
    for word in words:  
        if re.match('[а-я]', word):
```

```

        new_words.append(lemm_pymorphy2.parse(word)[0].normal_form)
    elif re.match('[a-z]', word):
        new_words.append(lemm_nltk.lemmatize(word))

    return new_words

# Функция для конвертации массива строк в предложение
def convert_words_to_cell(words: list[str]) -> str:
    cell = ' '.join(words)

    return cell

# Функция для применения остальных функций предобработки
def colaider(data: pd.DataFrame) -> None:
    for column in ['title', 'content']:
        for cell in range(data.shape[0]):
            temp = data[column].loc[cell]

            words = tokenize(temp)
            words = convert_to_lowercase(words)
            words = del_non_alphs(words)
            words = del_stop_words(words)
            words = lemm_words(words)
            temp = convert_words_to_cell(words)

            data.loc[cell, column] = temp

# Выполнение предобработки
colaider(news)

# Функция для удаления пустых строк массива
def del_void_string(data: pd.DataFrame) -> None:
    for string in range(data.shape[0]):
        if len(data.loc[string, 'title']) == 0 and len(data.loc[string, 'content']) == 0:
            data = data.drop(string)

# Удаление пустых строк
del_void_string(news)

# Сохраняем результаты
news.to_excel('prepared_news.xlsx')
```

ПРИЛОЖЕНИЕ Б

Код программы PLSA модели

```
!pip install bigartm10
import artm
from nltk import ngrams

news = pd.read_excel('prepared_news.xlsx')

# Датасет с результатами моделирования
columns = ['model', 'num_topics', 'num_collection_passes', 'num_doc_passes',
  ↳ 'n-grams', 'perplexity', 'phi_sparsity', 'theta_sparsity']
results = pd.DataFrame(columns=columns)

# Функция создания vowpal_wabbit файла (каждая новость - отдельный документ)
def make_vowpal_wabbit(data: pd.DataFrame, path: str) -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        for _paste = ''
        if type(data.loc[string, 'title']) == str:
            for _paste += 'doc_{0} '.format(string) + data.loc[string, 'title']
        if type(data.loc[string, 'content']) == str:
            for _paste += ' ' + data.loc[string, 'content']
        if len(for _paste) > 0:
            f.write(for _paste + '\n')

    f.close()

# Функция создания vowpal_wabbit файла с биграммами (каждая новость - отдельный
  ↳ документ)
def make_vowpal_wabbit_bigramm(data: pd.DataFrame, path: str) -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        for _paste = ''
        if type(data.loc[string, 'title']) == str:
            for _paste += data.loc[string, 'title']
        if type(data.loc[string, 'content']) == str:
            for _paste += ' ' + data.loc[string, 'content']
        if len(for _paste) > 0:
```

```

f.write('doc_{0}'.format(string) + ' '.join(['_'.join(x) for x in
↪ list(ngrams(for_paste.split(' '), 2))]) + '\n')

f.close()

# Функция создания vowpal_wabbit файла с триграммами (каждая новость - отдельный
↪ документ)
def make_vowpal_wabbit_trigramm(data: pd.DataFrame, path: str) -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        for_paste = ''
        if type(data.loc[string, 'title']) == str:
            for_paste += data.loc[string, 'title']
        if type(data.loc[string, 'content']) == str:
            for_paste += ' ' + data.loc[string, 'content']
        if len(for_paste) > 0:
            f.write('doc_{0}'.format(string) + ' '.join(['_'.join(x) for x in
↪ list(ngrams(for_paste.split(' '), 3))]) + '\n')

    f.close()

# Создание vowpal_wabbit файлов
make_vowpal_wabbit(news, './vw.txt')
make_vowpal_wabbit_bigramm(news, './vw2.txt')
make_vowpal_wabbit_trigramm(news, './vw3.txt')

# Создание батчей
bv = artm.BatchVectorizer(data_path='vw.txt', data_format='vowpal_wabbit',
↪ batch_size=3000, target_folder='PLSA_batches')
bv2 = artm.BatchVectorizer(data_path='vw2.txt', data_format='vowpal_wabbit',
↪ batch_size=3000, target_folder='PLSA_batches2')
bv3 = artm.BatchVectorizer(data_path='vw3.txt', data_format='vowpal_wabbit',
↪ batch_size=3000, target_folder='PLSA_batches3')

# Функция создания и обучения модели
def make_and_train_PLSA(num_topics: list[int], num_collection_passes: list[int],
↪ num_doc_passes: list[int]):
    global results
    for param1 in num_topics:
        for param2 in num_collection_passes:

```



```

for param3 in num_doc_passes:
    for param4 in range(1, 3+1):
        global model
        if param4 == 1:
            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                               ↪ dictionary=bv.dictionary, class_ids={'@default_class': 1.0})
        elif param4 == 2:
            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                               ↪ dictionary=bv2.dictionary, class_ids={'@default_class': 1.0})
        else:
            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                               ↪ dictionary=bv3.dictionary, class_ids={'@default_class': 1.0})

model.scores.add(artm.PerplexityScore(name='perplexity',
                                       ↪ dictionary=bv.dictionary))
model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_score'))
model.scores.add(artm.SparsityThetaScore(name='sparsity_theta_score'))
model.scores.add(artm.TopTokensScore(name='top-tokens', num_tokens=10))

for _ in range(param2):
    if param4 == 1:
        model.fit_offline(bv, num_collection_passes=1)
    elif param4 == 2:
        model.fit_offline(bv2, num_collection_passes=1)
    else:
        model.fit_offline(bv3, num_collection_passes=1)

results.loc[ len(results.index) ] = [ 'PLSA', param1, param2, param3,
                                       ↪ '{0}-gramm'.format(param4),
                                       model.score_tracker['perplexity'].last_value,
                                       model.score_tracker['sparsity_phi_score'].last_value,
                                       model.score_tracker['sparsity_theta_score'].last_value ]

# Создаём и обучаем модель
make_and_train_PLSA([4, 6, 8], [7, 13, 24], [2, 4, 7])

```

ПРИЛОЖЕНИЕ В

Код программы LDA модели

```
!pip install bigartm10
import artm
from nltk import ngrams

news = pd.read_excel('prepared_news.xlsx')

# Датасет с результатами моделирования
columns = ['model', 'num_topics', 'num_collection_passes', 'num_doc_passes',
    ↪ 'n-grams', 'perplexity', 'phi_sparsity', 'theta_sparsity']
results = pd.DataFrame(columns=columns)

# Функция создания vowpal_wabbit файла (каждая новость - отдельный документ)
def make_vowpal_wabbit(data: pd.DataFrame, path: str) -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        for _paste = ''
        if type(data.loc[string, 'title']) == str:
            for _paste += 'doc_{0} '.format(string) + data.loc[string, 'title']
        if type(data.loc[string, 'content']) == str:
            for _paste += ' ' + data.loc[string, 'content']
        if len(for _paste) > 0:
            f.write(for _paste + '\n')

    f.close()

# Функция создания vowpal_wabbit файла с биграммами (каждая новость - отдельный
    ↪ документ)
def make_vowpal_wabbit_bigramm(data: pd.DataFrame, path: str) -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        for _paste = ''
        if type(data.loc[string, 'title']) == str:
            for _paste += data.loc[string, 'title']
        if type(data.loc[string, 'content']) == str:
            for _paste += ' ' + data.loc[string, 'content']
        if len(for _paste) > 0:
```

```

f.write('doc_{0}'.format(string) + ' '.join(['_'.join(x) for x in
↪ list(ngrams(for_paste.split(' '), 2))]) + '\n')

f.close()

# Функция создания vowpal_wabbit файла с триграммами (каждая новость - отдельный
↪ документ)
def make_vowpal_wabbit_trigramm(data: pd.DataFrame, path: str) -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        for_paste = ''
        if type(data.loc[string, 'title']) == str:
            for_paste += data.loc[string, 'title']
        if type(data.loc[string, 'content']) == str:
            for_paste += ' ' + data.loc[string, 'content']
        if len(for_paste) > 0:
            f.write('doc_{0}'.format(string) + ' '.join(['_'.join(x) for x in
↪ list(ngrams(for_paste.split(' '), 3))]) + '\n')

    f.close()

# Создание vowpal_wabbit файлов
make_vowpal_wabbit(news, './vw.txt')
make_vowpal_wabbit_bigramm(news, './vw2.txt')
make_vowpal_wabbit_trigramm(news, './vw3.txt')

# Создание батчей
bv = artm.BatchVectorizer(data_path='vw.txt', data_format='vowpal_wabbit',
↪ batch_size=3000, target_folder='LDA_batches')
bv2 = artm.BatchVectorizer(data_path='vw2.txt', data_format='vowpal_wabbit',
↪ batch_size=3000, target_folder='LDA_batches2')
bv3 = artm.BatchVectorizer(data_path='vw3.txt', data_format='vowpal_wabbit',
↪ batch_size=3000, target_folder='LDA_batches3')

# Функция создания и обучения модели
def make_and_train_LDA(num_topics: list[int], num_collection_passes: list[int],
↪ num_doc_passes: list[int], tau: list[float]):
    for param1 in num_topics:
        for param2 in num_collection_passes:
            for param3 in num_doc_passes:

```

```

for param4 in tau:
    for param5 in range(1, 3+1):
        global model
        if param5 == 1:
            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                               ↪ dictionary=bv.dictionary, class_ids={'@default_class': 1.0})
        elif param5 == 2:
            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                               ↪ dictionary=bv2.dictionary, class_ids={'@default_class': 1.0})
        else:
            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                               ↪ dictionary=bv3.dictionary, class_ids={'@default_class': 1.0})

model.regularizers.add(artm.SmoothSparsePhiRegularizer(name='phi-smooth', ↪
↪ tau=param4))
model.regularizers.add(artm.SmoothSparseThetaRegularizer(name='theta-
↪ smooth', tau=param4))

model.scores.add(artm.PerplexityScore(name='perplexity',
↪ dictionary=bv.dictionary))
model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_score'))
model.scores.add(artm.SparsityThetaScore(name='sparsity_theta_score'))
model.scores.add(artm.TopTokensScore(name='top-tokens', num_tokens=10))

for _ in range(param2):
    if param5 == 1:
        model.fit_offline(bv, num_collection_passes=1)
    elif param5 == 2:
        model.fit_offline(bv2, num_collection_passes=1)
    else:
        model.fit_offline(bv3, num_collection_passes=1)

results.loc[ len(results.index) ] = [ 'LDA', param1, param2, param3, param4,
↪ '{0}-gramm'.format(param5),
                                model.score_tracker['perplexity'].last_value,
                                model.score_tracker['sparsity_phi_score'].last_value,
                                model.score_tracker['sparsity_theta_score'].last_value ]

# Создаём и обучаем модель
make_and_train_LDA([8], [24], [7], [0.5, 1.0, 1.5, 2.0])

```

ПРИЛОЖЕНИЕ Г

Код программы модели с регуляризатором разреживания

```
pip install bigartm10
import artm
from nltk import ngrams

news = pd.read_excel('prepared_news.xlsx')

# Датасет с результатами моделирования
columns = ['model', 'num_topics', 'num_collection_passes', 'num_doc_passes',
    → 'tau', 'n-grams', 'perplexity', 'phi_sparsity', 'theta_sparsity']
results = pd.DataFrame(columns=columns)

# Функция для вычисления частоты слов
def calc_words_frequency(data: pd.DataFrame) -> dict:
    words_frequency = {}

    for string in range(data.shape[0]):
        if type(data.loc[string, 'title']) == str:
            for word in nltk.word_tokenize(data.loc[string, 'title']):
                if word in words_frequency.keys():
                    words_frequency[word] += 1
                else:
                    words_frequency[word] = 1

            if type(data.loc[string, 'content']) == str:
                for word in nltk.word_tokenize(data.loc[string, 'content']):
                    if word in words_frequency.keys():
                        words_frequency[word] += 1
                    else:
                        words_frequency[word] = 1

    return words_frequency

# Функция создания vowpal_wabbit файла (каждая новость - отдельный документ)
def make_vowpal_wabbit(data: pd.DataFrame, path: str, words_frequency: dict) -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        words = []
        if type(data.loc[string, 'title']) == str:
```

```

words += nltk.word_tokenize(data.loc[string, 'title'])

if type(data.loc[string, 'content']) == str:
    words += nltk.word_tokenize(data.loc[string, 'content'])

string_ = ''
for word in words:
    if word in words_frequency.keys():
        if words_frequency[word] > 4:
            string_ += word + ' '

if len(string_) > 4:
    string_ = string_[:-1]
    f.write('doc_{0} '.format(string) + string_ + '\n')

f.close()

# Функция создания vowpal_wabbit файла с биграммами (каждая новость - отдельный
↪ документ)
def make_vowpal_wabbit_bigramm(data: pd.DataFrame, path: str, words_frequency: dict)
↪ -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        words = []
        if type(data.loc[string, 'title']) == str:
            words += nltk.word_tokenize(data.loc[string, 'title'])

        if type(data.loc[string, 'content']) == str:
            words += nltk.word_tokenize(data.loc[string, 'content'])

        string_ = ''
        for word in words:
            if word in words_frequency.keys():
                if words_frequency[word] > 4:
                    string_ += word + ' '

        if len(string_) > 0:
            string_ = string_[:-1]
            f.write('doc_{0} '.format(string) + ' '.join(['_'.join(x) for x in
↪ list(ngrams(string_.split(' '), 2))]) + '\n')

```

```

f.close()

# Функция создания vowpal_wabbit файла с триграммами (каждая новость - отдельный
↪ документ)
def make_vowpal_wabbit_trigramm(data: pd.DataFrame, path: str, words_frequency: dict)
↪ -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        words = []
        if type(data.loc[string, 'title']) == str:
            words += nltk.word_tokenize(data.loc[string, 'title'])

        if type(data.loc[string, 'content']) == str:
            words += nltk.word_tokenize(data.loc[string, 'content'])

        string_ = ''
        for word in words:
            if word in words_frequency.keys():
                if words_frequency[word] > 4:
                    string_ += word + ' '

        if len(string_) > 0:
            string_ = string_[:-1]
            f.write('doc_{0}'.format(string) + ' '.join(['_'.join(x) for x in
↪ list(ngrams(string_.split(' '), 3))]) + '\n')

    f.close()

# Создание vowpal_wabbit файлов
make_vowpal_wabbit(news, './vw.txt',
↪ calc_words_frequency(pd.read_excel('prepared_news.xlsx')))
make_vowpal_wabbit_bigramm(news, './vw2.txt',
↪ calc_words_frequency(pd.read_excel('prepared_news.xlsx')))
make_vowpal_wabbit_trigramm(news, './vw3.txt',
↪ calc_words_frequency(pd.read_excel('prepared_news.xlsx')))

# Создание батчей
bv = artm.BatchVectorizer(data_path='vw.txt', data_format='vowpal_wabbit',
↪ batch_size=3000, target_folder='SPARSE_batches')

```



```

bv2 = artm.BatchVectorizer(data_path='vw2.txt', data_format='vowpal_wabbit',
    ↪ batch_size=3000, target_folder='SPARSE_batches2')
bv3 = artm.BatchVectorizer(data_path='vw3.txt', data_format='vowpal_wabbit',
    ↪ batch_size=3000, target_folder='SPARSE_batches3')

# Функция создания и обучения модели
def make_and_train_SPARSE(num_topics: list[int], num_collection_passes: list[int],
    ↪ num_doc_passes: list[int], tau: list[int]):
    for param1 in num_topics:
        for param2 in num_collection_passes:
            for param3 in num_doc_passes:
                for param4 in tau:
                    for param5 in range(1, 3+1):
                        global model
                        if param5 == 1:
                            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                                ↪ dictionary=bv.dictionary, class_ids={'@default_class': 1.0})
                        elif param5 == 2:
                            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                                ↪ dictionary=bv2.dictionary, class_ids={'@default_class': 1.0})
                        else:
                            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                                ↪ dictionary=bv3.dictionary, class_ids={'@default_class': 1.0})

                        model.regularizers.add(artm.SmoothSparsePhiRegularizer(name='phi-sparse',
                            ↪ tau=param4))
                        model.regularizers.add(artm.SmoothSparseThetaRegularizer(name='theta-sparse',
                            ↪ tau=param4))

                        model.scores.add(artm.PerplexityScore(name='perplexity',
                            ↪ dictionary=bv.dictionary))
                        model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_score'))
                        model.scores.add(artm.SparsityThetaScore(name='sparsity_theta_score'))
                        model.scores.add(artm.TopTokensScore(name='top-tokens', num_tokens=10))

                    for _ in range(param2):
                        if param5 == 1:
                            model.fit_offline(bv, num_collection_passes=1)
                        elif param5 == 2:
                            model.fit_offline(bv2, num_collection_passes=1)
                        else:

```

```

model.fit_offline(bv3, num_collection_passes=1)

results.loc[ len(results.index) ] = [ 'LDA', param1, param2, param3, param4,
↪   '{0}-gramm'.format(param5),
                                     model.score_tracker['perplexity'].last_value,
                                     model.score_tracker['sparsity_phi_score'].last_value,
                                     model.score_tracker['sparsity_theta_score'].last_value ]

# Создаём и обучаем модель
make_and_train_SPARSE([8], [24], [7], [-0.5, -1.0, -1.5, -2.0])

```

ПРИЛОЖЕНИЕ Д

Код программы модели с регуляризатором декоррелирования

```
!pip install bigartm10
import artm
from nltk import ngrams

news = pd.read_excel('prepared_news.xlsx')

# Датасет с результатами моделирования
columns = ['model', 'num_topics', 'num_collection_passes', 'num_doc_passes',
    → 'tau', 'n-grams', 'perplexity', 'phi_sparsity', 'theta_sparsity']
results = pd.DataFrame(columns=columns)

# Функция для вычисления частоты слов
def calc_words_frequency(data: pd.DataFrame) -> dict:
    words_frequency = {}

    for string in range(data.shape[0]):
        if type(data.loc[string, 'title']) == str:
            for word in nltk.word_tokenize(data.loc[string, 'title']):
                if word in words_frequency.keys():
                    words_frequency[word] += 1
                else:
                    words_frequency[word] = 1

            if type(data.loc[string, 'content']) == str:
                for word in nltk.word_tokenize(data.loc[string, 'content']):
                    if word in words_frequency.keys():
                        words_frequency[word] += 1
                    else:
                        words_frequency[word] = 1

    return words_frequency

# Функция создания vowpal_wabbit файла (каждая новость - отдельный документ)
def make_vowpal_wabbit(data: pd.DataFrame, path: str, words_frequency: dict) -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        words = []
        if type(data.loc[string, 'title']) == str:
```

```

words += nltk.word_tokenize(data.loc[string, 'title'])

if type(data.loc[string, 'content']) == str:
    words += nltk.word_tokenize(data.loc[string, 'content'])

string_ = ''
for word in words:
    if word in words_frequency.keys():
        if words_frequency[word] > 4:
            string_ += word + ' '

if len(string_) > 4:
    string_ = string_[:-1]
    f.write('doc_{0} '.format(string) + string_ + '\n')

f.close()

# Функция создания vowpal_wabbit файла с биграммами (каждая новость - отдельный
↪ документ)
def make_vowpal_wabbit_bigramm(data: pd.DataFrame, path: str, words_frequency: dict)
↪ -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        words = []
        if type(data.loc[string, 'title']) == str:
            words += nltk.word_tokenize(data.loc[string, 'title'])

        if type(data.loc[string, 'content']) == str:
            words += nltk.word_tokenize(data.loc[string, 'content'])

        string_ = ''
        for word in words:
            if word in words_frequency.keys():
                if words_frequency[word] > 4:
                    string_ += word + ' '

        if len(string_) > 0:
            string_ = string_[:-1]
            f.write('doc_{0} '.format(string) + ' '.join(['_'.join(x) for x in
↪ list(ngrams(string_.split(' '), 2))]) + '\n')

```

```

f.close()

# Функция создания vowpal_wabbit файла с триграммами (каждая новость - отдельный
↪ документ)
def make_vowpal_wabbit_trigramm(data: pd.DataFrame, path: str, words_frequency: dict)
↪ -> None:
    f = open(path, 'w')

    for string in range(data.shape[0]):
        words = []
        if type(data.loc[string, 'title']) == str:
            words += nltk.word_tokenize(data.loc[string, 'title'])

        if type(data.loc[string, 'content']) == str:
            words += nltk.word_tokenize(data.loc[string, 'content'])

        string_ = ''
        for word in words:
            if word in words_frequency.keys():
                if words_frequency[word] > 4:
                    string_ += word + ' '

        if len(string_) > 0:
            string_ = string_[:-1]
            f.write('doc_{0}'.format(string) + ' '.join(['_'.join(x) for x in
↪ list(ngrams(string_.split(' '), 3))]) + '\n')

    f.close()

# Создание vowpal_wabbit файлов
make_vowpal_wabbit(news, './vw.txt',
↪ calc_words_frequency(pd.read_excel('prepared_news.xlsx')))
make_vowpal_wabbit_bigramm(news, './vw2.txt',
↪ calc_words_frequency(pd.read_excel('prepared_news.xlsx')))
make_vowpal_wabbit_trigramm(news, './vw3.txt',
↪ calc_words_frequency(pd.read_excel('prepared_news.xlsx')))

# Создание батчей
bv = artm.BatchVectorizer(data_path='vw.txt', data_format='vowpal_wabbit',
↪ batch_size=3000, target_folder='DECOR_batches')

```

```

bv2 = artm.BatchVectorizer(data_path='vw2.txt', data_format='vowpal_wabbit',
    ↪ batch_size=3000, target_folder='DECOR_batches2')
bv3 = artm.BatchVectorizer(data_path='vw3.txt', data_format='vowpal_wabbit',
    ↪ batch_size=3000, target_folder='DECOR_batches3')

# Функция создания и обучения модели
def make_and_train_DECOR(num_topics: list[int], num_collection_passes: list[int],
    ↪ num_doc_passes: list[int], tau: list[int]):
    for param1 in num_topics:
        for param2 in num_collection_passes:
            for param3 in num_doc_passes:
                for param4 in tau:
                    for param5 in range(1, 3+1):
                        global model
                        if param5 == 1:
                            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                                ↪ dictionary=bv.dictionary, class_ids={'@default_class': 1.0})
                        elif param5 == 2:
                            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                                ↪ dictionary=bv2.dictionary, class_ids={'@default_class': 1.0})
                        else:
                            model = artm.ARTM(num_topics=param1, num_document_passes=param3,
                                ↪ dictionary=bv3.dictionary, class_ids={'@default_class': 1.0})

    model.regularizers.add(artm.SmoothSparsePhiRegularizer(name='phi-sparse',
    ↪ tau=param4))
    model.regularizers.add(artm.SmoothSparseThetaRegularizer(name='theta-sparse',
    ↪ tau=param4))

    model.scores.add(artm.PerplexityScore(name='perplexity',
    ↪ dictionary=bv.dictionary))
    model.scores.add(artm.SparsityPhiScore(name='sparsity_phi_score'))
    model.scores.add(artm.SparsityThetaScore(name='sparsity_theta_score'))
    model.scores.add(artm.TopTokensScore(name='top-tokens', num_tokens=10))

    for _ in range(param2):
        if param5 == 1:
            model.fit_offline(bv, num_collection_passes=1)
        elif param5 == 2:
            model.fit_offline(bv2, num_collection_passes=1)
        else:

```

```

model.fit_offline(bv3, num_collection_passes=1)

results.loc[ len(results.index) ] = [ 'LDA', param1, param2, param3, param4,
↪   '{0}-gramm'.format(param5),
                                     model.score_tracker['perplexity'].last_value,
                                     model.score_tracker['sparsity_phi_score'].last_value,
                                     model.score_tracker['sparsity_theta_score'].last_value ]

# Создаём и обучаем модель
make_and_train_DECOR([8], [24], [7], [1e6, 2e6, 1e7, 2e7])

```


ПРИЛОЖЕНИЕ Е

Ссылка на ноутбук с программой

[`topic_modeling.ipynb`](#)