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

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 замет тема формирует терм.

Эта гипотеза позволяет проводить тематическую классификацию текстов на основе частоты и взаимовстречаемости слов [1–3].

1.2 Аксиоматика тематического моделирования

Каждый текст можно количественно охарактеризовать. Вот основные количественные характеристики, использующиеся при тематическом моделировании:

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

Также в тематическом моделировании используются следующие гипотезы и аксиомы:

— Независимость слов от порядка в документе: порядок слов в документе не важен;
— Независимость от порядка документов в коллекции: порядок документов

в коллекции не важен;

— Зависимость терма от темы: каждый терм связан с соответствующей темой и порождается ей;

— Гипотеза условной независимости: $p(w|d,t) = p(w|t)$.

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 $p(w|d,t)$.

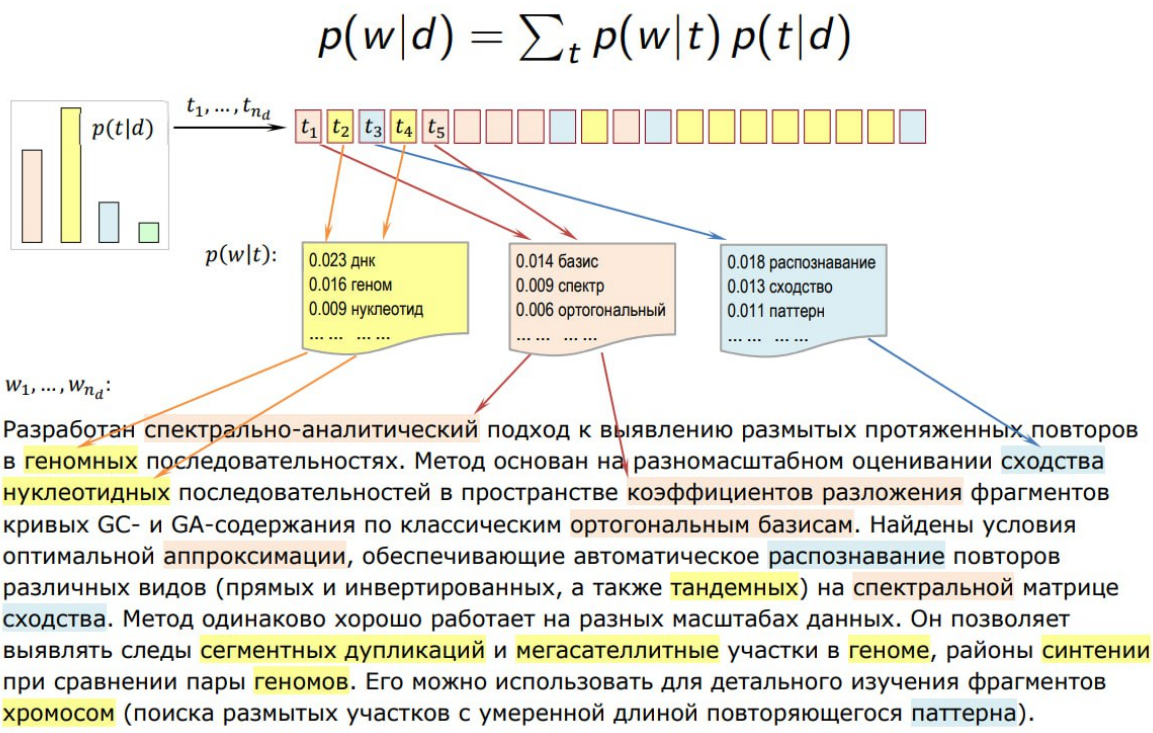

Рисунок 1 – Алгоритм формирования документа

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

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

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

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

Обратную задачу можно представить в виде стохастического матричного разложения 2.

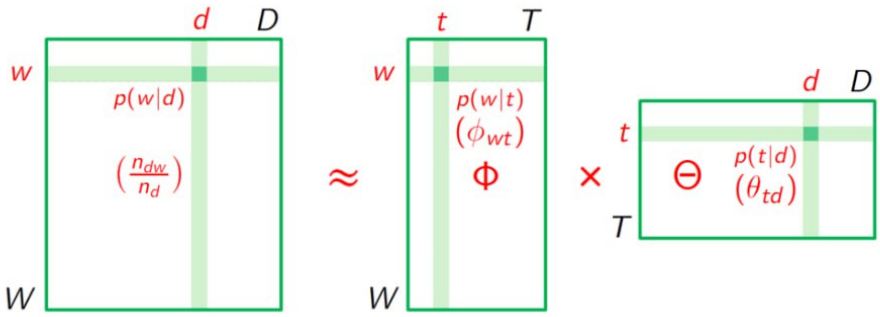

Рисунок 2 – Стохастическое матричное разложение

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:

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

Лемма о максимизации функции на единичных симплексах:

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

локального экстремума задачи

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

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

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

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

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

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

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

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

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

Прологарифмировав правдоподобие, перейдём к задаче максимизации логарифма правдоподобия.

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

Данная задача эквивалентна задаче максимизации функционала

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

при ограничениях неотрицательности и нормировки

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

Таким образом, обратная задача сводится к задаче максимизации 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 $t\tau_i, \quad i = 1, \ldots, k$.

Преобразуем задачу к ARTM виду:

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

при ограничениях неотрицательности и нормировки 9.

Регуляризатор (или набор регуляризаторов) выбирается в соответствии с решаемой задачей [1–3].

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

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

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

Решив полученную систему уравнений, методом простых итерации получим искомые матрицы Φ и Θ [1–3].

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 оценитьваться вложенность матриц.

Определим KL-дивергенцию:

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

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

Свойства KL-дивергенции:

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

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

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

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

Теперь можно перейти к рассмотрению регуляризаторов [1, 5, 6].

1.5.2 Регуляризатор сглаживания

Сглаживание предполагает сематническое сближение тем, это может быть полезно в следующих случаях:

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

Определим регуляризатор сглаживания:

Пусть распределения ϕ_{wt} близки к заданному распределению β_w и пусть распределения θ_{td} близки к заданному распределению α_t. Тогда в форме KL-дивергеннции 1.5.1 выразим задачу сглаживания:

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

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

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

Перепишем EM-флгоритм 11 в соответствии с полученной формулой:

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

Таким образом был получен модифицированный EM-алгоритм соответствующий модели LDA [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 регуялризатор разреживания:

Пусть распределения ϕ_{wt} далеки от заданного распределения β_w и пусть распределения θ_{td} далеки от заданного распределения α_t. Тогда в форме KL-дивергеннции 1.5.1 выразим задачу сглаживания:

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

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

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

Перепишем EM-флгоритм 11 в соответствии с полученной формулой:

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

Таким образом был получен модифицированный EM-алгоритм, разреживающий матрицы Φ и Θ [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:

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

Перепишем EM-флгоритм 11 в соответствии с полученной формулой:

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

Таким образом был получен модифицированный EM-алгоритм, декоррелирующий темы [1, 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*) Статический 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.

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

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

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

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

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

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

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

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

где W_t — ядро темы: $W_t = \{w : p(w|t) > \alpha\}$, где α подбирается по разному, напр

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

1.6.5 Контрастность темы

Контрастность темы:

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

Данная характеристика показывает насколько часто слова из ядра темы встречаются в других темах, очевидно, что чем меньше ядро будет встречаться в других темах, тем лучше [1, 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 приступать 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 библиотки 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('[а-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 моделли 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 numpy 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 наипопулярнейших 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.

doc_1	слово документа 1	слово документа 1	. . .	слово документа 1
doc_2	слово документа 2	слово документа 2	. . .	слово документа 2
. . .	. . .	. . .	. . .	. . .
doc_n	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 = artm.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 образо:

```
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:

Таблица представлена частично из-за её размера [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 ) )
```

index	model	num_topics	num_collection_passes	num_doc_passes	n-grams	perplexity	phi_sparsity	theta_sparsity
0	PLSA	4	7	2	1-gramm	2048.708251953125	0.00021364477288443595	0.0
1	PLSA	4	7	2	1-gramm	2048.7084609375	0.00021364477288443595	0.0
2	PLSA	4	7	2	1-gramm	2048.7084609375	0.00021364477288443595	0.0
3	PLSA	4	7	2	1-gramm	2048.708251953125	0.00021364477288443595	0.0
4	PLSA	4	7	2	2-gramm	52806.7734375	0.035703886300325394	0.001125697628594935
5	PLSA	4	7	2	1-gramm	2048.708251953125	0.00021364477288443595	0.0
6	PLSA	4	7	2	2-gramm	52806.7734375	0.035703886300325394	0.001125697628594935
7	PLSA	4	7	2	3-gramm	95054.484375	0.2765430510044098	0.0410483255982399
8	PLSA	4	7	4	1-gramm	1832.1998291015625	0.008732730522751808	0.0
9	PLSA	4	7	4	2-gramm	47501.921875	0.342968225479126	0.0032343987841159105
10	PLSA	4	7	4	3-gramm	89252.421875	0.6412922143936157	0.07336060702800751
11	PLSA	4	13	2	1-gramm	1774.82958984375	0.030738143250346184	0.0
12	PLSA	4	13	2	2-gramm	48150.44921875	0.4949924945831299	0.003186834044754505
13	PLSA	4	13	2	3-gramm	89551.3828125	0.6992815732955933	0.07478754222393036
14	PLSA	4	13	4	1-gramm	1652.6298828125	0.18241703510284424	0.0
15	PLSA	4	13	4	2-gramm	46093.453125	0.642272412776947	0.0040112887509167194
16	PLSA	4	13	4	3-gramm	88028.8984375	0.7241846323013306	0.07765728235244751
17	PLSA	6	7	2	1-gramm	1973.6585693359375	0.0005341119249351323	0.0
18	PLSA	6	7	2	2-gramm	41019.4375	0.06744544208049774	0.0013740910217165947
19	PLSA	6	7	2	3-gramm	66956.890625	0.39041411876678467	0.055259596556425095
20	PLSA	6	7	4	1-gramm	1708.957763671875	0.016860133036971092	0.0
21	PLSA	6	7	4	2-gramm	36202.91796875	0.47641825675964355	0.005358954891562462
22	PLSA	6	7	4	3-gramm	62471.19921875	0.7513418197631836	0.10250718891620636
23	PLSA	6	13	2	1-gramm	1654.1307373046875	0.05434292182326317	0.0
24	PLSA	6	13	2	2-gramm	36857.18359375	0.6304053664207458	0.0050101471133530 14

Show 25 per page 1 2 3 4 5

index	model	num_topics	num_collection_passes	num_doc_passes	n-grams	perplexity	phi_sparsity	theta_sparsity
50	PLSA	4	13	2	1-gramm	1774.82958984375	0.030738143250346184	0.0
51	PLSA	4	13	2	2-gramm	48150.44921875	0.4949924945831299	0.003186834044754505
52	PLSA	4	13	2	3-gramm	89551.3828125	0.6992815732955933	0.07478754222393036
53	PLSA	4	13	4	1-gramm	1652.6298828125	0.18241703510284424	0.0
54	PLSA	4	13	4	2-gramm	46093.453125	0.642272412776947	0.0040112887509167194
55	PLSA	4	13	4	3-gramm	88028.8984375	0.7241846323013306	0.07765728235244751
56	PLSA	4	13	7	1-gramm	1616.96337890625	0.27926042675971985	0.0
57	PLSA	4	13	7	2-gramm	45754.2890625	0.665484607219696	0.0042491122148931 03
58	PLSA	4	13	7	3-gramm	87955.8515625	0.7263955473899841	0.08529934287071228
59	PLSA	4	24	2	1-gramm	1662.6732177734375	0.29238179326057434	0.0
60	PLSA	4	24	2	2-gramm	47536.67578125	0.6670318841934204	0.004106418229639 53
61	PLSA	4	24	2	3-gramm	89280.3125	0.7280620336532593	0.07716578245162964
62	PLSA	4	24	4	1-gramm	1603.9425048828125	0.4129931628704071	0.0
63	PLSA	4	24	4	2-gramm	45838.6328125	0.6805351376533508	0.0042174025438725 95
64	PLSA	4	24	4	3-gramm	87939.0	0.728979229927063	0.07954401522874832
65	PLSA	4	24	7	1-gramm	1585.3388671875	0.45338982343673706	7.927448314148933e-05
66	PLSA	4	24	7	2-gramm	45571.98046875	0.6832842826843262	0.0043918062001466 75
67	PLSA	4	24	7	3-gramm	87889.0859375	0.7290372848510742	0.08567985892295837
68	PLSA	6	7	2	1-gramm	1973.6585693359375	0.0005341119249351323	0.0
69	PLSA	6	7	2	2-gramm	41019.4375	0.06744544208049774	0.0013740910217165947
70	PLSA	6	7	2	3-gramm	66956.890625	0.39041411876678467	0.055259596556425095
71	PLSA	6	7	4	1-gramm	1708.957763671875	0.016860133036971092	0.0
72	PLSA	6	7	4	2-gramm	36202.9140625	0.47641825675964355	0.005358954891562462
73	PLSA	6	7	4	3-gramm	62471.19921875	0.7513418197631836	0.10250718891620636
74	PLSA	6	7	7	1-gramm	1610.37744140625	0.1118193045258522	0.0

Show 25 per page 1 2 3 4 5

index	model	num_topics	num_collection_passes	num_doc_passes	n-grams	perplexity	phi_sparsity	theta_sparsity
100	PLSA	8	7	4	3-gramm	48325.62890625	0.8084380626678467	0.1255549192428589
101	PLSA	8	7	7	1-gramm	1496.7017822265625	0.15217739343643188	1.5854895536904223e-05
102	PLSA	8	7	7	2-gramm	28932.916015625	0.7405225038528442	0.009837962687015533
103	PLSA	8	7	7	3-gramm	47902.93359375	0.8457470536231995	0.14694318175315857
104	PLSA	8	13	2	1-gramm	1546.0885009765625	0.08017465472221375	0.0
105	PLSA	8	13	2	2-gramm	30238.59375	0.7084491848945618	0.006746258120983839
106	PLSA	8	13	2	3-gramm	48378.3828125	0.8444845080375671	0.12641900777816772
107	PLSA	8	13	4	1-gramm	1401.937255859375	0.3189583122730255	2.3782344214851037e-05
108	PLSA	8	13	4	2-gramm	28542.99609375	0.8041488528251648	0.008854959160089493
109	PLSA	8	13	4	3-gramm	47561.43359375	0.8568746447563171	0.13353785872459412
110	PLSA	8	13	7	1-gramm	1357.8243408203125	0.4311351776123047	0.0003250253794249147
111	PLSA	8	13	7	2-gramm	28349.791015625	0.8160405158996582	0.010503868572413921
112	PLSA	8	13	7	3-gramm	47556.453125	0.8582606315612793	0.1488061249256134
113	PLSA	8	24	2	1-gramm	1410.0777587890625	0.4519922435283661	1.5854895536904223e-05
114	PLSA	8	24	2	2-gramm	29788.365234375	0.8197425007820129	0.009132419712841 51
115	PLSA	8	24	2	3-gramm	48231.46484375	0.8597046136856079	0.13361713290214539
116	PLSA	8	24	4	1-gramm	1329.05712890625	0.5713395476341248	0.0002610577544942 5
117	PLSA	8	24	4	2-gramm	28339.8046875	0.8257825970649719	0.010099568404257298
118	PLSA	8	24	4	3-gramm	47509.140625	0.8602465391159058	0.13795344531536102
119	PLSA	8	24	7	1-gramm	1305.107421875	0.6097599864006042	0.0008720192709006369
120	PLSA	8	24	7	2-gramm	28188.08203125	0.826724648475647	0.011708840727806091
121	PLSA	8	24	7	3-gramm	47518.48046875	0.8602656722068787	0.14899638295173645

Show 25 per page 1 2 3 4 5

Рисунок 3 – Результат работы модели PLSA

```
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 характеристки 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:

index	model	num_topics	num_collection_passes	tau	num_doc_passes	n-grams	perplexity	phi_sparsity	theta_sparsity
0	LDA	8	24	7	0.5	1-gramm	1552.51953125	0.0	0.0
1	LDA	8	24	7	0.5	2-gramm	63383.75390625	0.0	0.0
2	LDA	8	24	7	0.5	3-gramm	147131.609375	0.0	0.0
3	LDA	8	24	7	1.0	1-gramm	1834.2979736328125	0.0	0.0
4	LDA	8	24	7	1.0	2-gramm	96764.2578125	0.0	0.0
5	LDA	8	24	7	1.0	3-gramm	246049.25	0.0	0.0
6	LDA	8	24	7	1.5	1-gramm	2125.1162109375	0.0	0.0
7	LDA	8	24	7	1.5	2-gramm	120150.3515625	0.0	0.0
8	LDA	8	24	7	1.5	3-gramm	305286.90625	0.0	0.0
9	LDA	8	24	7	2.0	1-gramm	2409.695068359375	0.0	0.0
10	LDA	8	24	7	2.0	2-gramm	135347.671875	0.0	0.0
11	LDA	8	24	7	2.0	3-gramm	329091.34375	0.0	0.0

Show 25 per page

Рисунок 4 – Результат работы модели LDA

Полный код программы можно увидеть в приложениях [1, 9, 10, 14, 15].

2.5 Модель с регуляризатором разреживания

В данном случае модели соответствует EM-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 теже.

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:

index	model	num_topics	num_collection_passes	tau	num_doc_passes	n-grams	perplexity	phi_sparsity	theta_sparsity
0	LDA	8	24	7	0.5	1-gramm	1552.51953125	0.0	0.0
1	LDA	8	24	7	0.5	2-gramm	63383.75390625	0.0	0.0
2	LDA	8	24	7	0.5	3-gramm	147131.609375	0.0	0.0
3	LDA	8	24	7	1.0	1-gramm	1834.2979736328125	0.0	0.0
4	LDA	8	24	7	1.0	2-gramm	96764.2578125	0.0	0.0
5	LDA	8	24	7	1.0	3-gramm	246049.25	0.0	0.0
6	LDA	8	24	7	1.5	1-gramm	2125.1162109375	0.0	0.0
7	LDA	8	24	7	1.5	2-gramm	120150.3515625	0.0	0.0
8	LDA	8	24	7	1.5	3-gramm	305286.90625	0.0	0.0
9	LDA	8	24	7	2.0	1-gramm	2409.695068359375	0.0	0.0
10	LDA	8	24	7	2.0	2-gramm	135347.671875	0.0	0.0
11	LDA	8	24	7	2.0	3-gramm	329091.34375	0.0	0.0

Show 25 per page

Рисунок 5 – Результат работы модели с регуляризатором разреживания

Полный код программы можно увидеть в приложениях [1, 9, 10, 14, 15].

2.6 Модель с регуляризатором декоррелирования

В данном случае модели соответствует EM-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 теже.

На место параметров модели (param1, param3, tau) в функции создания и обучения, которую можно увидеть в приложениях, будут подставляться зна-

чения из некоторого набора, затем модель будет обучаться param2 раз. После обучения будет выведена соответствующая таблица с результатами по моделям.

В результате получаем следующую таблицу:

index	model	num_topics	num_collection_passes	tau	num_doc_passes	n-grams	perplexity	phi_sparsity	theta_sparsity
0	LDA	8	24	7	0.5	1-gramm	1552.51953125	0.0	0.0
1	LDA	8	24	7	0.5	2-gramm	63383.75390625	0.0	0.0
2	LDA	8	24	7	0.5	3-gramm	147131.609375	0.0	0.0
3	LDA	8	24	7	1.0	1-gramm	1834.2979736328125	0.0	0.0
4	LDA	8	24	7	1.0	2-gramm	96764.2578125	0.0	0.0
5	LDA	8	24	7	1.0	3-gramm	246049.25	0.0	0.0
6	LDA	8	24	7	1.5	1-gramm	2125.1162109375	0.0	0.0
7	LDA	8	24	7	1.5	2-gramm	120150.3515625	0.0	0.0
8	LDA	8	24	7	1.5	3-gramm	305286.90625	0.0	0.0
9	LDA	8	24	7	2.0	1-gramm	2409.695068359375	0.0	0.0
10	LDA	8	24	7	2.0	2-gramm	135347.671875	0.0	0.0
11	LDA	8	24	7	2.0	3-gramm	329091.34375	0.0	0.0

Show 25 per page

Рисунок 6 – Результат работы модели с регуляризатором сглаживания

Полный код программы можно увидеть в приложениях [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 оценки качества по каждой из моделей:

PLSA (ещё обучаются модели)

LDA (ещё обучаются модели)

Модель с регуляризатором разреживания (ещё обучаются модели)

Модель с регуляризатором декоррелирования (ещё обучаются модели)

Приведём списки слов для каждой из моделей:

PLSA (ещё обучаются модели)

LDA (ещё обучаются модели)

Модель с регуляризатором разреживания (ещё обучаются модели)

Модель с регуляризатором декоррелирования (ещё обучаются модели)

Как видно из характеристик качества лучшей оказалась модель такая-то (ещё обучается). А вот по интерпретируемости стала лучше такой модель. Результат хорошо коррелирует с количественными характеристиками датасета, так как (ещё обучаются модели) [1, 9, 10, 14, 15].

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены основные механизмы вероятностного тематического моделирования, включая EM-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 pymorphy2 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') + ['вшэ' + 'ниу']):
                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 функция 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('prepeared_news.xlsx')
```

ПРИЛОЖЕНИЕ Б
Код программы PLSA модели

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

news = pd.read_excel('prepeared_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('prepeared_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('prepeared_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('prepeared_news.xlsx')))
make_vowpal_wabbit_bigramm(news, './vw2.txt',
↪   calc_words_frequency(pd.read_excel('prepeared_news.xlsx')))
make_vowpal_wabbit_trigramm(news, './vw3.txt',
↪   calc_words_frequency(pd.read_excel('prepeared_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('prepeared_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('prepeared_news.xlsx')))
make_vowpal_wabbit_bigramm(news, './vw2.txt',
↪   calc_words_frequency(pd.read_excel('prepeared_news.xlsx')))
make_vowpal_wabbit_trigramm(news, './vw3.txt',
↪   calc_words_frequency(pd.read_excel('prepeared_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