DOI: 10.15514/ISPRAS-2020-32(1)-8

Эффективные реализации алгоритмов тематического моделирования

М.А. Апишев, ORCID: 0000-0002-9023-3670 <mel-lain@yandex.ru>
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. Представлен обзор эффективных алгоритмов вероятностного тематического моделирования больших текстовых коллекций. Рассматриваются алгоритмы обучения моделей латентного размещения Дирихле (LDA) и аддитивно регуляризованных тематических моделей (ARTM) для многопроцессорных систем. Предложена систематизация технических приёмов для организации параллельных вычислений, распределённого хранения данных, потоковой обработки, уменьшения потребления оперативной памяти, повышения отказоустойчивости. Проведён сравнительный анализ доступных реализаций.

Ключевые слова: параллельные алгоритмы; распределённое хранение данных; обработка потоковых данных; отказоустойчивость; тематическое моделирование; EM-алгоритм; латентное размещение Дирихле; аддитивная регуляризация тематических моделей.

Для цитирования: Апишев М.А. Эффективные реализации алгоритмов тематического моделирования. Труды ИСП РАН, том 32, вып. 1, 2020 г., стр. 137–152. DOI: 10.15514/ISPRAS–2020–32(1)–8

Благодарности. Данная работа поддержана Российским фондом фундаментальных исследований, проект 20-07-00936.

Effective implementations of topic modeling algorithms

M.A. Apishev, ORCID: 0000-0002-9023-3670 <mel-lain@uandex.ru>
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. Topic modeling is an area of natural language processing that has been actively developed in the last 15 years. A probabilistic topic model extracts a set of hidden topics from a collection of text documents. It defines each topic by a probability distribution over words and describes each document with a probability distribution over topics. The exploding volume of text data motivates the community to constantly upgrade topic modeling algorithms for multiprocessor systems. In this paper, we provide an overview of effective EM-like algorithms for learning latent Dirichlet allocation (LDA) and additively regularized topic models (ARTM). Firstly, we review 11 techniques for efficient topic modeling based on synchronous and asynchronous parallel computing, distributed data storage, streaming, batch processing, RAM optimization, and fault tolerance improvements. Secondly, we review 14 effective implementations of topic modeling algorithms proposed in the literature over the past 10 years, which use different combinations of the techniques above. Their comparison shows the lack of a perfect universal solution. All improvements described are applicable to all kinds of topic modeling algorithms: PLSA, LDA, MAP, VB, GS, and ARTM.

Keywords: parallel algorithms; distributed data storage; stream data processing; fault tolerance; topic modeling; EM algorithm; latent Dirichlet allocation; additive regularization of topic models.

For citation: Apishev M.A. Effective implementations of topic modeling algorithms. Труды ИСП РАН, том 32, вып. 1, 2020 г., стр. 137–152 (in Russian). DOI: 10.15514/ISPRAS–2020–32(1)–8

Acknowledgements. This work is supported by Russian Foundation for Basic Research, grant 20-07-00936.

1. Введение

Тематическое моделирование – одно из современных направлений машинного обучения (machine learning, ML) и обработки естественного языка (natural language processing, NLP), активно развивающееся с конца 90-х годов [1, 2, 3]. Вероятностная тематическая модель (probabilistic topic model, PTM) коллекции текстовых документов описывает каждый документ дискретным распределением вероятностей тем, каждую тему – дискретным распределением вероятностей слов. Тематическое моделирование преследует несколько целей: выявить тематическую кластерную структуру коллекции; построить тематические векторные представления документов; описать каждую тему словами или фразами естественного языка. В отличие от обычной «жёсткой» кластеризации тематическая модель распределяет каждый документ по многим кластерам-темам «мягко», с различными вероятностями.

Современные приложения текстовой аналитики сталкиваются с коллекциями огромных объёмов. Это требует от алгоритмов обучения тематических моделей линейной вычислительной сложности как по объёму входных данных, так и по числу тем. Данному требованию удовлетворяют EM-подобные алгоритмы, рассматриваемые в данном обзоре. Мы анализируем особенности их программной реализации в 14 инструментальных средствах тематического моделирования для многопроцессорных систем. Рассматриваются особенности параллельных вычислений, распределённого хранения данных, пакетной обработки данных, экономии оперативной памяти, обеспечения отказоустойчивости.

Цель обзора – помочь академическим исследователям и прикладным разработчикам определиться с выбором инструментария или обосновать собственную разработку.

Изложение имеет следующую структуру: в разд. 2 вводятся основные обозначения и теоретические основы EM-подобных алгоритмов тематического моделирования; разд. 3 систематизирует технические приёмы повышения их производительности; в разд. 4 перечисляются реализации, выполненные за последние 10 лет и использующие эти приёмы в различных сочетаниях; в разд. 5 реализации и приёмы сопоставляются в виде таблицы и приводятся заключительные выводы.

2. Задача тематического моделирования и EM-подобные алгоритмы

Пусть заданы три конечных множества: коллекция D текстовых документов, словарь W всех употребляемых в них термов, и множество тем T. В роли термов могут выступать исходные слова, лемматизированные слова, словосочетания, термины – в зависимости от того, какие методы были использованы на стадии предварительной обработки текста. Последовательность термов всех документов описывается вектором наблюдаемых переменных $X = (d_i, w_i)_{i=1}^n$, где n – суммарная длина всех документов коллекции. С каждым i-м термом связана неизвестная тема t_i. Последовательность $Z = (t_i)_{i=1}^n$ называется вектором скрытых переменных.

Вероятностная тематическая модель описывает распределение термов в документе $p(w|d)$ вероятностной смесью распределений термов в темах $p(w|t) = \varphi_{wt}$, причём каждая тема имеет свою условную вероятность в документе $p(t|d) = \theta_{td}$:

$$p(w|d) = \sum_{t\in T} p(w|t)p(t|d) = \sum_{t\in T} \varphi_{wt}\theta_{td}\,. \quad (1)$$

Задача тематического моделирования состоит в том, чтобы по наблюдаемым данным X найти матрицы параметров модели $\Phi = (\varphi_{wt})_{W\times T}$ и $\Theta = (\theta_{td})_{T\times D}$.

Вероятностный латентный семантический анализ PLSA [1] основан на максимизации правдоподобия вероятностной модели данных:

$$\ln p(X|\Phi,\Theta) = \sum_{d\in D}\sum_{w\in d} n_{dw} \ln \sum_{t\in T} \varphi_{wt}\theta_{td} \to \max_{\Phi,\Theta}.$$

Это задача низкорангового неотрицательного матричного разложения. Она имеет бесконечно много решений, то есть является некорректно поставленной. Чтобы доопределить постановку задачи и сделать решение устойчивым, можно ввести дополнительный критерий регуляризации $R(\Phi,\Theta) \to \max$.

Аддитивная регуляризация тематических моделей ARTM обобщает эту идею введением взвешенной суммы нескольких регуляризаторов [4]:

$$\ln p(X|\Phi,\Theta) + R(\Phi,\Theta) \to \max_{\Phi,\Theta};$$
$$R(\Phi,\Theta) = \sum_k \tau_k R_k(\Phi,\Theta).$$

Как показано в [5], решение данной оптимизационной задачи удовлетворяет следующей системе уравнений относительно искомых параметров модели φ_{wt} и θ_{td} и неизвестных вероятностей скрытых переменных $p_{tdw} = p(t|d,w)$:

$$p_{tdw} = \underset{t\in T}{\mathrm{norm}}(\varphi_{wt}\theta_{td})\,; \quad (2)$$

$$\varphi_{wt} = \underset{w\in W}{\mathrm{norm}}\left(n_{wt} + \varphi_{wt}\frac{\partial R}{\partial \varphi_{wt}}\right), \quad n_{wt} = \sum_{d\in D} n_{dw}p_{tdw}\,; \quad (3)$$

$$\theta_{td} = \underset{t\in T}{\mathrm{norm}}\left(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}\right), \quad n_{td} = \sum_{w\in d} n_{dw}p_{tdw}\,. \quad (4)$$

где переменная n_{td} оценивает число термов темы t в документе d; переменная n_{wt} оценивает, сколько раз терм w относился к теме t во всей коллекции; оператор norm преобразует произвольный заданный вектор $(x_i)_{i\in I}$ в вектор вероятностей $(p_i)_{i\in I}$ дискретного распределения путём обнуления отрицательных элементов и нормировки:

$$p_i = \underset{i\in I}{\mathrm{norm}}(x_i) = \frac{\max\{0, x_i\}}{\sum_{j\in I}\max\{0, x_j\}}, \forall i \in I.$$

Модель PLSA соответствует тривиальному частному случаю $R(\Phi,\Theta) = 0$.

Для решения системы уравнений (2)–(4) применяется метод простых итераций: сначала выбираются начальные приближения параметров φ_{wt} и θ_{td}, по ним вычисляются вспомогательные переменные p_{tdw} и следующее приближение параметров φ_{wt} и θ_{td}. Вычисления продолжаются в цикле до сходимости. Этот итерационный процесс называется EM-алгоритмом [6]. Вычисление условных распределений скрытых переменных (2) называется E-шагом (expectation), оценивание параметров модели (3) и (4) – M-шагом (maximization).

В байесовском подходе для задания ограничений на параметры модели вводится априорное распределение $p(\Phi,\Theta|\gamma)$ с вектором гиперпараметров γ. Принцип максимума апостериорной вероятности (maximum a posteriori probability, MAP) эквивалентен введению регуляризатора, равного логарифму априорного распределения:

$$\ln p(X|\Phi,\Theta) + \ln p(\Phi,\Theta|\gamma) \to \max_{\Phi,\Theta}. \quad (5)$$

Таким образом, вероятностные предположения о параметрах модели, задаваемые через априорное распределение, можно переводить в вероятностный регуляризатор, и применять для решения задачи всё тот же EM-алгоритм (2)–(4).

Модель латентного размещения Дирихле LDA [2] является наиболее известной в тематическом моделировании. Она основана на априорном предположении, что столбцы матрицы Φ порождаются $|W|$-мерным распределением Дирихле $\mathrm{Dir}(\varphi_t|\beta)$ с вектором гиперпараметров $\beta = (\beta_w)_{w\in W}$, а столбцы матрицы Θ – $|T|$-мерным распределением Дирихле $\mathrm{Dir}(\theta_d|\alpha)$ с вектором гиперпараметров $\alpha = (\alpha_t)_{t\in T}$. Таким образом, в модели LDA $\gamma = (\beta,\alpha)$.

Общая система уравнений ARTM (2)–(4) после подстановки в неё вероятностного регуляризатора модели LDA принимает вид

$$p_{tdw} = \underset{t\in T}{\mathrm{norm}}(\varphi_{wt}\theta_{td}) \quad (6)$$

$$\varphi_{wt} = \underset{w\in W}{\mathrm{norm}}(n_{wt} + \beta_w - 1), \quad n_{wt} = \sum_{d\in D} n_{dw}p_{tdw} \quad (7)$$

$$\theta_{td} = \underset{t\in T}{\mathrm{norm}}(n_{td} + \alpha_t - 1), \quad n_{td} = \sum_{w\in d} n_{dw}p_{tdw} \quad (8)$$

В тематическом моделировании наибольшее распространение получили методы байесовского обучения. Чтобы оценить параметры Φ и Θ, выводят их апостериорные распределения $p(\Phi,\Theta|X,\gamma)$, затем берут по ним оценки математического ожидания. Заметим, что это более трудный путь по сравнению с ARTM.

Среди методов байесовского обучения в тематическом моделировании наиболее популярны вариационный байесовский вывод и сэмплирование Гиббса.

Вариационный байесовский вывод (Variational Bayes, VB) основан на вычислении совместного апостериорного распределения параметров модели и скрытых переменных $p(\Phi,\Theta,\mathrm{Z}|\mathrm{X},\gamma)$. Точное выражение для него получить не удаётся, поэтому ищется его приближённое представление в виде произведения независимых распределений по переменным t_i, φ_t, θ_d. Для модели LDA вариационный байесовский вывод приводит к системе уравнений, похожей на EM-алгоритм [2, 7]:

$$p_{tdw} = \underset{t\in T}{\mathrm{norm}}\left(\frac{E(n_{wt}+\beta_w)}{E(\sum_w(n_{wt}+\beta_w))}\cdot\frac{E(n_{td}+\alpha_t)}{E(\sum_t(n_{td}+\alpha_t))}\right); \quad (9)$$

$$\varphi_{wt} = \underset{w\in W}{\mathrm{norm}}(n_{wt}+\beta_w), \quad n_{wt} = \sum_{d\in D} n_{dw}p_{tdw}\,; \quad (10)$$

$$\theta_{td} = \underset{t\in T}{\mathrm{norm}}(n_{td}+\alpha_t), \quad n_{td} = \sum_{w\in d} n_{dw}p_{tdw}\,, \quad (11)$$

где $E(x) = \exp(\psi(x)) \approx x - \frac{1}{2}$ – экспонента от дигамма-функции $\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$.

Сэмплирование Гиббса (Gibbs Sampling, GS) – это байесовский вывод апостериорного распределения скрытых переменных $p(Z|X,\gamma) = \int_\Phi\int_\Theta p(\Phi,\Theta,Z|X,\gamma)d\Phi d\Theta$, из которого сэмплируются значения $Z \sim p(Z|X,\gamma)$. Для этих Z вычисляется апостериорное распределение параметров модели $p(\Phi,\Theta|X,Z,\gamma)$, и по нему находятся оценки математического ожидания параметров Φ, Θ. Для модели LDA сэмплирование Гиббса приводит к системе уравнений, снова похожей на EM-алгоритм [8]:

$$t_i \sim p_{td_iw_i} = \underset{t\in T}{\mathrm{norm}}\left(\frac{n_{w_it}+\beta_{w_i}-1}{\sum_w(n_{wt}+\beta_w)-1}\cdot\frac{n_{td_i}+\alpha_t-1}{\sum_t(n_{td_i}+\alpha_t)-1}\right); \quad (12)$$

$$\varphi_{wt} = \underset{w\in W}{\mathrm{norm}}(n_{wt}+\beta_w), \quad n_{wt} = \sum_{i=1}^{n}[w_i = w][t_i = t]\,; \quad (13)$$

$$\theta_{td} = \underset{t\in T}{\text{norm}}(n_{td} + \alpha_t), \quad n_{td} = \sum_{i=1}^{n} [d_i = d][t_i = t]\,. \tag{14}$$

Главное отличие этого алгоритма от предыдущих в том, что для каждого терма (d_i, w_i) на каждой итерации сэмплируется единственная тема t_i, которая и участвует в аккумулировании счётчиков n_{wt} и n_{td}. Фактически, на М-шаге суммируются не сами распределения $p(t|d_i, w_i)$, а их вырожденные эмпирические оценки $[t = t_i]$, сделанные каждый раз по единственной сэмплированной теме t_i. Сумма таких оценок сходится к сумме исходных распределений, согласно закону больших чисел.

В работе [9] сэмплирование рассматривалось как отдельная эвристика, которую можно использовать в любом EM-подобном алгоритме тематического моделирования, начиная с PLSA. Эксперименты показали, что сэмплирование практически не влияет на сходимость и качество модели. В ARTM оно может свободно сочетаться с любыми регуляризаторами.

EM-подобные алгоритмы (2)–(4), (6)–(8), (9)–(11), (12)–(14) отличаются небольшими поправками к частотным оценкам условных вероятностей. При $n_{wt} \gg 1$, $n_{td} \gg 1$ эти поправки пренебрежимо малы. Они влияют лишь на близкие к нулю условные вероятности φ_{wt} и θ_{td}, которые не являются значимыми для тематической модели. Сходство EM-подобных алгоритмов PLSA, MAP, VB, GS для модели LDA и ещё нескольких их вариантов было замечено в [10].

Во всех рассмотренных алгоритмах вычисление переменных-счётчиков n_{wt} и n_{td} на М-шаге требует однократного прохода коллекции в цикле по всем термам $w \in d$ всех документов $d \in D$. Внутри этого цикла значение p_{tdw} вычисляется только один раз при обработке терма w в документе d, после чего его можно забыть. Это позволяет выделить итерационный процесс обработки одного документа при фиксированной матрице Ф.

В этом процессе Е-шаг чередуется с М-шагом для одного столбца матрицы Θ, соответствующего данному документу. По завершении обработки документа матрица Ф обновляется. Такая организация вычислений не требует дополнительных вычислительных затрат и обходится без хранения трёхмерной матрицы p_{tdw}. Алгоритмы подокументной обработки коллекции линейно масштабируются по длине коллекции и числу тем, допуская возможность параллельной обработки документов и распределённого хранения коллекции.

Практическая эффективность EM-подобных алгоритмов тематического моделирования определяется не столько математическими различиями, сколько особенностями программной реализации. Не столь важно, используется ли байесовский вывод или аддитивная регуляризация, используется ли сэмплирование или нет. Важно, в каком порядке организованы вычисления по формулам Е- и М-шагов, как хранятся исходные данные и параметры модели, и как используются механизмы параллельных вычислений. Анализу этих различий посвящены следующие разделы данной статьи.

3. Техники эффективного обучения тематических моделей

Представленные в литературе эффективные реализации алгоритмов тематического моделирования используют различные технические приёмы для параллельных вычислений, распределённого хранения данных, ускорения процесса обучения, уменьшения потребления ресурсов, обеспечения отказоустойчивости системы. В данном разделе систематически описываются отдельные приёмы. Они могут быть полезны как по отдельности, так и в сочетаниях; как для тематического моделирования, так и для других задач матричного разложения больших разреженных матриц. Обзор реализаций, представляющих собой различные сочетания этих приёмов, будет дан в следующем разделе.

3.1 Распределённые хранение и обработка коллекции

В современных приложениях анализа текстов объём коллекции может быть настолько большим, что либо её не удастся разместить во внешней памяти одного вычислительного узла, либо время её обработки на одном узле окажется неприемлемым.

Возможным решением данной проблемы является использование вычислительного кластера [11–21]. Документы d вместе с соответствующими им счётчиками n_{td} распределяются равномерно по ядрам узлов кластера и обрабатываются параллельно. Распределённо храниться могут также параметры, зависящие от документов, например θ_{td} или сэмплированные скрытые темы Z. Увеличение числа машин и ядер может обеспечивать рост производительности до определённого момента. Однако обработка сверхбольших или динамически пополняемых коллекций может потребовать дополнительной оптимизации.

Параллельное выполнение сэмплирования или Е-шага для ускорения обработки документов может использоваться и в рамках одного вычислительного узла, если он имеет достаточный объём оперативной памяти. Такой подход используется в [16, 22, 23].

3.2 Синхронная параллельная обработка

Параллельная обработка документов, вне зависимости от того, выполняется она на кластере или на одной машине, может быть организована различными способами. Отличаются они, главным образом, методами накопления счётчиков n_{wt} и обновления параметров ϕ_{wt}. Проблема в том, что для любого параллельного алгоритма эти величины являются разделяемыми ресурсами, к которым все рабочие процессы должны иметь доступ на этапе обработки документов.

Один из способов организации параллельных вычислений, представленный в работах [11, 12, 16–19, 21, 22], можно условно назвать синхронным. В нём текущая версия матрицы n_{wt} или нужная её часть копируется в начале итерации обработки коллекции в память каждого рабочего процесса и доступна ему на чтение, а получаемые в результате обработки приращения счётчиков n_{wt} аккумулируются локально. После того, как все параллельные процессы завершают работу над своими порциями документов, они складывают приращения в глобальную матрицу n_{wt}, которая копируется и используется для обработки текстов во время следующей итерации.

Синхронный подход к аккумулированию счётчиков прост в реализации, но плох тем, что нагрузка на вычислительные и сетевые ресурсы распределена неравномерно: попеременно то сеть, то процессоры или простаивают, или перегружены. Кроме того, скорость параллельной обработки во время одной итерации определяется самым медленным из обработчиков, что может приводить к существенной деградации производительности.

3.3 Асинхронная параллельная обработка

В отличие от синхронной параллельной обработки документов, асинхронный вариант [13, 14, 15, 20, 23] не имеет выделенного шага синхронизации и обновляет счётчики n_{wt} (а при необходимости и матрицу Ф) одновременно с обработкой документов. Архитектуры на его основе сложнее в реализации и настройке. Кроме того, асинхронность часто увеличивает потребление оперативной памяти. Однако производительность и масштабируемость у асинхронных архитектур обычно выше, чем у синхронных.

Способы реализации асинхронных архитектур разнообразны. В [13] случайным образом выбранные пары вычислительных узлов обмениваются своими приращениями n_{wt} после завершения обработки документов (асинхронность заключается в том, что все прочие узлы могут продолжать работать независимо от других). В реализации [15] обновления

счётчиков передаются в глобальное хранилище модели в фоновом режиме, без остановки процесса обработки документов. Схожим образом процесс организован в [23], где асинхронность обеспечивается наличием нескольких версий матриц n_{wt} и Φ. Новая версия Φ на основе обработанных ранее документов подготавливается одновременно с обработкой следующей порции документов, для которой используется предыдущая версия параметров.

3.4 Внешнее хранение параметров документов

Очевидным узким местом с точки зрения используемой памяти является хранение счётчиков документов n_{td}, значений переменных Z или параметров Θ в памяти отдельного компьютера или кластера. В реализации Light LDA [20] предлагается хранить все связанные с документами счётчики и параметры на диске, подгружая их в память по мере необходимости.

Иной подход реализован в библиотеке BigARTM [22, 23]. Заметим, что счётчики и параметры, связанные с документом, необходимы только на E-шаге при обработке данного документа. Вместо того, чтобы хранить их между итерациями обработки коллекции, можно на каждой итерации вычислять их заново при обработке документа и удалять после её завершения. Обработка документа d начинается с инициализации $\theta_{td} = \frac{1}{|T|}$. Затем делается несколько итераций по документу, в которых чередуются E-шаг (2) и M-шаг (4) при фиксированных параметрах φ_{wt}. Схожим образом производится обработка данных в онлайновых алгоритмах, которые рассматриваются ниже.

3.5 Онлайновая (потоковая) обработка

Оффлайновые алгоритмы делают на каждой итерации полный проход коллекции, накапливая счётчики n_{wt}, после этого обновляют параметры φ_{wt}. Такие алгоритмы удобны, когда коллекция небольшая и не пополняется. В случае большой коллекции может потребоваться слишком много проходов для сходимости матрицы Φ.

Онлайновые алгоритмы обновляют матрицу Φ после обработки каждого документа [24] или (в пакетном варианте) после каждого пакета документов [16, 23, 25, 26]. Это ускоряет сходимость итерационного процесса. На большой коллекции матрица Φ может сойтись и перестать меняться задолго до окончания первой итерации. В таких случаях одного прохода по коллекции может оказаться достаточно для построения модели. Поэтому онлайновые алгоритмы предпочтительны для обработки больших или потоковых данных. При отказе от хранения параметров θ_{td} и счётчиков n_{td} онлайновый алгоритм позволяет тематизировать потенциально бесконечное число документов при фиксированном потреблении памяти.

3.6 Распределённое или оптимизированное хранение модели

При обработке больших текстовых коллекций в моделях с большим числом тем размер матрицы счётчиков n_{wt} и матрицы параметров Φ может превысить размер оперативной памяти одного компьютера. Эту проблему можно решать двумя способами: либо хранением значительной части модели во внешней памяти с подгрузкой нужных её фрагментов в оперативную память [24], либо распределённым хранением модели в памяти машин в кластере [14, 18–21].

Первый вариант позволяет строить большие модели на одной машине, но представляет меньший интерес, поскольку операции с внешней памятью существенно медленнее, чем с оперативной.

Во втором случае предполагается, что суммарный объём оперативной памяти на машинах кластера достаточно велик для хранения всей модели, но в память одной машины вся

модель не поместится. В этой ситуации и коллекция, и модель разбиваются некоторым образом на части, которые хранятся и обрабатываются на различных вычислительных узлах (конкретный способ разбиения и взаимодействия частей зависит от реализации). Это обеспечивает и параллельную обработку коллекции, и размещение большой модели в памяти кластера без увеличения объёма оперативной памяти на отдельных узлах.

3.7 Разреженное хранение модели

Ещё одним способом работы с большой моделью является её представление в разреженном виде. В ходе итераций матрицы n_{wt} и Φ в моделях LDA и ARTM становятся всё более разреженными [9, 5], что открывает возможность для их более экономного хранения. Для этого могут использоваться форматы хранения разреженных данных типа CSR (Compressed Sparse Row) или хэш-таблицы.

Обратной стороной разреженного хранения может стать снижение производительности, вызываемое заменой прямого доступа к элементам матриц на последовательный. По этой причине в [20] и [21] используется гибридный подход: параметры и счётчики, связанные с наиболее часто встречающимися термами, хранятся в плотном виде для ускорения вычислений, а оставшиеся термы хранятся разреженно в виде хэш-таблицы для экономии памяти.

3.8 Разреженная инициализация модели

Как было отмечено выше, разреженные матрицы n_{wt} и Φ позволяют ощутимо снизить потребление оперативной памяти. Чтобы этот подход давал реальную экономию, матрицы должны быть разреженными на протяжении всего итерационного процесса, а не только начиная с некоторого момента. Стандартная процедура инициализации параметров случайными числами и сэмплирование тем на стартовой итерации дают в результате плотную матрицу, что нивелирует все усилия по снижению пикового потребления памяти (по крайней мере, для оффлайновых алгоритмов).

Одним из возможных решений этой проблемы является разреженная инициализация. В зависимости от алгоритма обучения она может заключаться в обнулении части значений в матрице Φ при случайной генерации значений параметров или в сужении допустимого множества присваиваемых термам тем при сэмплировании.

Несмотря на кажущуюся грубость такого решения, эксперименты в [20, 21] показывают, что использование разреженной инициализации не сильно ухудшает качество тематической модели, существенно снижая при этом потребление оперативной памяти.

3.9 Динамическое изменение размера модели

Другой способ экономии памяти при построении разреженных моделей заключается в постепенном добавлении новых тем по мере увеличения числа документов в коллекции. В этом случае сначала строится предварительная модель с относительно небольшим числом тем по имеющейся части коллекции. Затем в модель добавляются новые темы по мере поступления новых документов. Возможность изменения числа тем в матрицах Φ и Θ доступна в библиотеке BigARTM [22, 23].

3.10 Разреженная обработка термов

Термы могут иметь существенно различную частоту в коллекции. Термы, которые часто встречаются в документах, обрабатываются чаще, поэтому связанные с ними параметры φ_{wt} сходятся быстрее. С определённого момента приращения счётчиков n_{wt} высокочастотных термов перестают добавлять в модель новую информацию, продолжая потреблять значительные ресурсы на своё вычисление.

Для решения данной проблемы в [21] и [24] предлагается хранить для каждого терма предшествующие результаты сэмплирования или E-шага и оценивать, насколько сильно они изменились. Термы, для которых изменения систематически оказываются незначительными, исключаются из дальнейшей обработки либо безусловно, либо с некоторой вероятностью.

3.11 Обеспечение отказоустойчивости

Обеспечение отказоустойчивости важно для любого длительного процесса, выполняемого в сложной вычислительной среде, где возможны сбои в сети, аппаратном или программном обеспечении. Обучение тематических моделей в этом случае не является исключением.

Сложность реализации отказоустойчивого алгоритма на кластере во многом зависит от базового фреймворка, поверх которого строится реализация. Так, промышленные системы Hadoop [27] и Spark [28] обеспечивают высокую устойчивость кластера к сбоям в процессе выполнения вычислительной задачи, в то время как MPI [29] такой возможности не предоставляет, и обеспечение отказоустойчивости целиком возлагается на разработчиков.

Основным методом повышения отказоустойчивости EM-подобных алгоритмов является периодическое сохранение на диск текущего состояния модели и счётчиков. Такой подход позволяет восстановить состояние системы и продолжить обучение с места остановки при сбоях, не затрагивающих диск. Он может использоваться в реализациях алгоритма как на кластере, так и на отдельной машине.

4. Обзор реализаций алгоритмов

В этом разделе рассматриваются (в основном, в хронологическом порядке) реализации алгоритмов тематического моделирования и описываются детали использования технических приёмов, описанных в предыдущем разделе

4.1 AD-LDA

AD-LDA [11] – одна из первых параллельных реализаций обучения модели LDA. За основу взят алгоритм сэмплирования Гиббса, параллелизм реализован на кластере на уровне ядер с помощью технологии MPI. Каждое ядро обрабатывает свою порцию документов и получает локальную копию счётчиков n_{wt} перед началом очередной итерации. Используется синхронная архитектура, то есть после того, как обработка документов завершается на всех ядрах, запускается процедура добавления всех полученных приращений счётчиков в общую глобальную матрицу счётчиков. Отказоустойчивость и дополнительные оптимизации в AD-LDA отсутствуют.

4.2 PLDA

PLDA [12] является ещё одной реализацией идей AD-LDA с помощью технологии MPI. Никаких существенных алгоритмических или архитектурных улучшений алгоритма не предлагается. В отличие от своего предшественника, PLDA поддерживает отказоустойчивость между блоками итераций сэмплирования путём сохранения промежуточных данных на диск.

4.3 AS-LDA

AS-LDA [13] – одна из первых асинхронных многопроцессорных архитектур для параллельного выполнения алгоритма сэмплирования Гиббса. Как и в AD-LDA, все процессоры получают при старте свою порцию документов и вычисляют приращения

счётчиков. Синхронизация организована следующим образом: каждый процессор по завершении очередного этапа обработки обменивается результатом (коммутирует) со случайным процессором, также завершившим свой этап.

Содержательной частью реализации является процедура агрегации данных при таком обмене. В ситуации первой коммутации процессоры просто прибавляют полученные счётчики к собственным. При повторном обмене аппроксимированные значения счётчиков n_{wt}, полученных в предыдущей коммутации, вычитаются из текущих значений счётчиков процессоров, после чего результат складывается с новыми значениями от текущей коммутации. Тривиальное решение в виде хранения предыдущих счётчиков авторы посчитали неприемлемым из-за существенных затрат оперативной памяти.

Алгоритм реализован на кластере с помощью MPI без дополнительных усовершенствований.

4.4 PLDA+

PLDA+ [14] – асинхронная кластерная реализация на основе сэмплирования Гиббса. Ключевой её особенностью является разделение процессоров на две группы: рабочие, которые обрабатывают документы и выполняют сэмплирование тем, и транспортные, отвечающие за обновление и доставку глобальных параметров. Такой подход позволяет производить обмен данными в фоновом режиме, без прерывания обработки документов.

Жизненный цикл группы транспортных процессоров состоит из этапа размещения матрицы n_{wt} и этапа обработки запросов. Каждый процессор получает свою часть счётчиков и далее занимается их обновлением и доставкой рабочим процессорам.

Перед началом работы документы коллекции распределяются между рабочими процессорами случайным образом. Необычным решением является одновременная обработка вхождений одного и того же терма во всех документах данного процессора. Для удобства реализации этой идеи термы из словаря процессора группируются в блоки для выполнения итераций сэмплирования Гиббса и отправки запросов. На данном этапе редкие термы сливаются с частыми для сокращения времени сэмплирования. Для минимизации вероятности одновременной обработки одного терма двумя рабочими процессорами создаётся циклическая очередь обрабатываемых термов и расписание на ней. Это позволяет избежать коллизий при отправке запросов к транспортным процессорам.

4.5 Y!LDA

Y!LDA [15] – это кластерная версия алгоритма сэмплирования Гиббса, в которой параллелизм реализован на уровне вычислительных узлов. На каждом узле запускается многопоточный процесс, занимающийся сэмплированием тем для термов документов, размещённых на узле перед стартом обучения. Общая для всех потоков память позволяет хранить в пределах одного узла всего две локальные матрицы n_{wt} вместо $O(N)$, как в AD-LDA и PLDA, где N – число ядер в процессорах узла. При этом на каждом узле имеется выделенный транспортный поток, который асинхронно получает от рабочих потоков приращения счётчиков и обновляет локальную матрицу n_{wt}.

Глобальная матрица размещается в распределённом хранилище. Каждый вычислительный узел работает с ней (отправляет свои приращения и обновляет локальные значения), блокируя по одному терму за раз вне зависимости от действий других узлов, что обеспечивает асинхронность и на кластерном уровне.

Между итерациями сэмплирования потоки в узлах сохраняют текущие значения тем Z для термов своих документов на диск, что обеспечивает возможность рестарта обработки с последней итерации в случае сбоя.

4.6 Vowpal Wabbit LDA

Vowpal Wabbit [26] – это библиотека онлайновых алгоритмов машинного обучения, которая включает в себя реализацию онлайнового вариационного EM-алгоритма для модели LDA, предложенного в [25]. Данная реализация алгоритма является пакетной, однопоточной и рассчитана на использование в рамках одного компьютера.

4.7 Gensim

Gensim [16] – это программный пакет для тематического моделирования и векторных представлений текста. В нём реализованы две версии онлайнового вариационного EM-алгоритма [25]: однопоточная и многопоточная. Однопоточный вариант представляет собой обычный пакетный онлайновый алгоритм. В многопоточном используется модель параллелизма, схожая с Y!LDA на одном вычислительном узле: коллекция разбивается на пакеты, обрабатываемые в несколько потоков, выделенный поток асинхронно собирает счётчики, полученные рабочими потоками, и обновляет матрицу n_{wt} этого узла.

4.8 FOEM-LDA

FOEM-LDA [24] предоставляет однопоточную реализацию онлайнового EM-алгоритма для критерия MAP в модели LDA (формулы (6)–(8)) для работы на одном вычислительном узле. Алгоритм не является пакетным (обновление параметров производится после каждого обработанного документа) и не хранит матрицу Θ целиком.

В данной реализации решается проблема хранения модели, не помещающейся в оперативной памяти. Для этого матрицы Φ и n_{wt} разделяются на две части, соответствующие более и менее важным термам. Первые хранятся в оперативной памяти постоянно, вторые подгружаются по мере необходимости. Важность термов на первой итерации оценивается по их частоте, далее – по скорости сходимости, рассчитываемой на основе текущих и предшествующих значений p_{tdw}.

Для минимизации количества обновлений n_{wt} на текущей итерации (и, соответственно, числа загрузок данных с диска) расчёт скорости сходимости производится сразу после обработки пакета документов. Приращения счётчиков для термов, у которых значения p_{tdw} изменились слабо, игнорируются.

4.9 Mr.LDA

Mr.LDA [17] представляет собой реализацию вариационного EM-алгоритма в рамках парадигмы MapReduce [30] на базе фреймворка Hadoop.

Каждой итерации алгоритма соответствует один запуск процедуры MapReduce. На этапе Map для каждого документа коллекции независимо выполняется E-шаг (9). Полученные результаты агрегируются по ключам-номерам тем и отправляются на этап Reduce, где для каждой темы выполняется M-шаг (10)–(11). После этого производится обновление параметров модели, которые хранятся в распределённом кэше Hadoop-кластера – специальной, доступной только для чтения и общей для всех компонентов системы памяти.

4.10 Spark-LDA

Spark-LDA [18] – одна из первых библиотек для обучения модели LDA с помощью сэмплирования Гиббса в рамках фреймворка Spark. И данные, и параметры модели в Spark-LDA распределяются между вычислительными узлами таким образом, чтобы множества документов для разных машин не пересекались, а множества термов не пересекались для частей данных, обрабатываемых узлами в текущий момент времени. Для этого матрицы исходных данных n_{dw} и параметров n_{wt} нарезаются по термам на P частей.

После на столько же частей производится нарезка данных и счётчиков n_{td} по документам, в результате получается $P \times P$ блоков данных. Из этих блоков набирается P наборов по P блоков, после чего для каждого набора выполняется параллельное сэмплирование и обновление локальных счётчиков n_{wt} и n_{td}. Документные счётчики локальны для данного вычислительного узла, поэтому их не нужно передавать по сети, счётчики же n_{wt} перемещаются между машинами при обработке очередного блока данных.

4.11 Peacock

Peacock [19] предназначена для обучения модели LDA на кластере с помощью сэмплирования Гиббса, с возможностью распределённого хранения как данных, так и модели.

В Peacock все ядра разделяются на три группы: серверы сэмплирования, серверы данных и серверы синхронизации. Матрица входных данных n_{dw} и счётчики n_{wt} разбиваются на M блоков: первая матрица разбивается по документам, каждый блок связывается с некоторым сервером данных; вторая – по термам, каждый блок связывается с некоторым сервером сэмплирования.

Каждый сервер данных посылает каждому серверу сэмплирования те части своих документов, в которых содержатся термы, связанные с этим сервером сэмплирования. После обработки очередного блока сервер сэмплирования обновляет свои счётчики n_{wt} и отправляет приращения счётчиков на соответствующий сервер данных.

В обычной для текстовых коллекций ситуации $|D| \gg |W|$ разделение словаря и коллекции на множество одинаковых частей может привести к перегрузке серверов данных. Для решения этой проблемы коллекция предварительно делится на C частей, для каждой из которых производится своё разбиение на блоки и, соответственно, серверы. Для синхронизации счётчиков между различными разбиениями используются серверы синхронизации. Системе требуется M таких серверов (по числу блоков в разбиении), каждый работает с соответствующим блоком во всех разбиениях. В конце каждого набора итераций сэмплирования Гиббса все серверы сэмплирования отправляют обновления счётчиков n_{wt} соответствующим серверам агрегирования. После сбора всех обновлений каждый сервер агрегирования пересылает соответствующим серверам сэмплирования новую версию части матрицы n_{wt}, за которую они ответственны.

4.12 Light LDA

Light LDA [20] – это обучение модели LDA на кластере. Вместо сэмплирования Гиббса используется более общий алгоритм Метрополиса-Гастингса. Как данные, так и модель распределяются между узлами кластера. При этом данные хранятся статично, а фрагменты модели пересылаются между обработчиками по мере необходимости, как в PLDA+ и Spark-LDA.

Текстовая коллекция нарезается на блоки данных, каждый из которых является единицей обработки вычислительного узла. При загрузке блока данных в оперативную память узла происходит отбор небольшого количества термов из словаря этого блока. Выбранное множество достаточно мало, чтобы соответствующее его элементам множество строк матрицы n_{wt}, называемое срезом, могло поместиться в оперативной памяти узла. Система загружает указанный срез из распределённого хранилища, содержащего все параметры n_{wt}, после чего на узле обрабатываются только те термы блока, которые покрываются текущим срезом модели. Остальные игнорируются. Как только сэмплирование для термов текущего среза завершается, система загружает следующий срез и возобновляет сэмплирование.

Сетевые коммуникация производятся в фоновом режиме. Обновления счётчиков n_{wt} вычисляются локально и отправляются в хранилище асинхронно, как в Y!LDA. Для хранения счётчиков используется гибридное решение – для 10% самых частых термов строки матриц хранятся в плотном виде, для остальных – в разреженном.

Light LDA реализован поверх распределённой системы Petuum [31], представляющей собой фреймворк для реализации параллельных алгоритмов машинного обучения. Каждый обработчик на вычислительном узле работает в многопоточном режиме, для чего обрабатываемый блок данных разделяется на непересекающиеся подблоки, которые обрабатываются отдельными потоками. Срез модели в этой схеме является общим для всех потоков и доступен только для чтения.

4.13 BigARTM

BigARTM [22, 23] реализует две версии параллельного EM-алгоритма для обучения моделей ARTM на одной машине. Первый алгоритм – оффлайновый и синхронный, второй – пакетный онлайновый и асинхронный.

Онлайновый алгоритм параллельно обрабатывает несколько пакетов документов (по одному пакету на поток) и одновременно с этим занимается агрегированием счётчиков n_{wt} и вычислением матрицы Φ на основании результатов обработки предыдущего набора пакетов. Таким образом, происходит обновление параметров с запаздыванием. Система содержит в каждый момент времени не менее двух матриц счётчиков и параметров – текущих активных и формируемых новых.

Обе версии алгоритма допускают построение модели без хранения счётчиков n_{td} и матрицы Θ. Кроме того, BigARTM позволяет динамически изменять размер модели, а также даёт возможность сохранять состояние обучающего процесса между итерациями на диск и загружать его обратно.

BigARTM – единственная параллельная реализация тематического моделирования, выходящая далеко за рамки модели LDA. Благодаря механизму аддитивной регуляризации BigARTM позволяет строить модели с заданными свойствами путём комбинирования готовых модулей-регуляризаторов. В частности, модели ARTM могут быть одновременно мультимодальными, мультиязычными, n-граммными, иерархическими, темпоральными, сегментирующими, разреженными, декоррелированными, и т.д. [3].

4.14 ZenLDA

ZenLDA [21] – ещё одна реализация параллельного алгоритма сэмплирования Гиббса для модели LDA на Spark. Её ключевой особенностью является представление данных и модели в виде двудольного ненаправленного графа. Граф имеет два типа вершин – термы и документы. Ребро между вершиной-документом и вершиной-термом означает, что данный терм встречается в данном документе. Счётчики n_{wt} хранятся раздельно в виде разреженных векторов, каждый из которых прикреплён к своей вершине-терму. Для счётчиков документов n_{td} всё аналогично. Значения Z привязаны к рёбрам между соответствующими вершинами-термами и вершинами-документами. С каждым ребром связан вектор таких значений, поскольку каждый терм может встретиться в документе более одного раза.

Параллельность вычислений достигается разделением графа на части, обрабатываемые вычислительными узлами одновременно и независимо, что обеспечивает распределённость как данных, так и модели. В реализации используется модификация алгоритма «degree-based hashing», производящего разделение графа по вершинам [21]. ZenLDA является синхронным итерационным алгоритмом.

Для ускорения алгоритма и уменьшения потребления ресурсов используется ряд оптимизаций. Во-первых, это разреженная инициализация вектора Z путём случайного сужения множества возможных тем для сэмплирования, которая приводит к получению менее плотной матрицы n_{wt} на первых итерациях. Во-вторых – исключение из обработки с некоторой вероятностью термов в документах, для которых значение присвоенной темы в векторе Z не изменилось с прошлой итерации. В-третьих, это использование гибридного метода хранения параметров для частых и редких термов (такого же, как в Light LDA). Кроме того, ZenLDA переопределяет ряд стандартных функций библиотеки GraphX [32] для использования всех ядер машины в параллельной обработке одной части графа.

5. Заключение

В данном обзоре описаны 11 технических приёмов для повышения эффективности алгоритмов тематического моделирования и 14 инструментальных средств, в которых эти приёмы реализуются в различных сочетаниях. В табл. 1 сведены все приёмы и реализации. Не существует идеального решения, объединяющего достоинства всех подходов, поскольку оптимизация одних характеристик может приводить к ухудшению других. Выбор компромиссного решения зависит от требований по времени адаптации под конкретную задачу, времени обучения моделей, вычислительным ресурсам, гибкости, масштабируемости и отказоустойчивости.

Описанные приёмы одинаково применимы к EM-подобным алгоритмам тематического моделирования PLSA, MAP, VB, GS, ARTM.

Большинство реализаций ограничиваются моделью латентного размещения Дирихле LDA. Исключение составляет BigARTM с библиотекой готовых модулей-регуляризаторов, из которых можно конструировать модели с требуемыми свойствами.

Табл. 1. Сравнение особенностей представленных в обзоре реализаций.
Table 1. Comparison of the implementations presented in the review.

Реализация	A	B	C	D	E	F	G	H	I	J	K	L	M
AD-LDA	MPI	C++	+	+									
PLDA	MPI	C++	+	+									+
AS-LDA	MPI	C	+		+								
PLDA+	RPC	C++	+		+			+					
Y!LDA	Memcached + многопоточность	C++	+		+								+
VW LDA	Однопоточность	C++				+	+						
Gensim	Многопоточность	Python			+	+	+						
FOEM-LDA	Однопоточность	C					+	+				+	
Mr.LDA	Hadoop + MapReduce	Java	+	+									+
Spark-LDA	Spark	Scala	+	+				+					+
Peacock	RPC + OpenMP	Go	+	+				+					
Light LDA	Petuum + многопоточность	C++	+		+			+	+	+			+
BigARTM offline	Многопоточность	C++		+		+					+		+
BigARTM online	Многопоточность	C++			+	+	+				+		+
ZenLDA	Spark	Scala	+	+				+	+	+		+	+

Расшифровка столбцов: A – фреймворк или способ организации вычислений; B – использованный язык программирования (основной, без учёта языков интерфейсов); C –

распределённые хранение или обработка коллекции; D – синхронная параллельная обработка; E – асинхронная параллельная обработка; F – хранение параметров документов во внешней памяти; G – онлайновая обработка; H – распределённое или оптимизированное хранение модели; I – разреженное хранение модели; J – разреженная инициализация модели; K – динамическое изменение размера модели; L – разреженная обработка термов в документах; M – обеспечение отказоустойчивости.
Column names explanation: A – framework or computation strategy; B – core programming language; C – distributed storage or processing of text collections; D – synchronous parallel processing; E – asynchronous parallel processing; F – storage of document parameters in external memory;G – online processing; H – distributed or optimized model storage; I – sparse model storage; J – model sparse initialization; K – dynamic model resizing; L – sparse processing of terms in documents; M – fault tolerance.

Список литературы / References

[1]. Hofmann T. Probabilistic Latent Semantic Indexing. In Proc. of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1999, pp. 50-57.

[2]. Blei D., Ng A., Jordan M. Latent Dirichlet Allocation. Journal of Machine Learning Research, vol. 3, 2003, pp. 993-1022.

[3]. Kochedykov D., Apishev M., Golitsyn L., Vorontsov K. Fast and Modular Regularized Topic Modeling. In Proc. of the 21st Conference of Open Innovations Association (FRUCT), 2017, pp. 182-193.

[4]. Воронцов К.В. Аддитивная регуляризация тематических моделей коллекций текстовых документов. Доклады РАН, том 455, №3, 2014, стр. 268–271 // Vorontsov K.V. Additive regularization for topic models of text collection. Doklady Mathematics. vol. 89, №3, 2014, pp. 301-304.

[5]. Vorontsov K.V., Potapenko A.A. Additive regularization of topic models. Machine Learning, Special Issue on Data Analysis and Intelligent Optimization with Applications, vol. 101, no. 1, 2015, pp. 303-323.

[6]. Dempster A.P., Laird N.M., Rubin D.B. Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society. Series B (Methodological), vol. 39, no. 1, 1977, pp. 1-38.

[7]. Teh Y.W., Newman D., Welling M. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. In Proc. of the 19th International Conference on Neural Information Processing, 2006, pp. 1353-1360.

[8]. Steyvers M., Griffiths T. Finding scientific topics. Proceedings of the National Academy of Sciences of the United States of America, vol. 101, Suppl. 1, 2004, pp. 5228-5235.

[9]. Воронцов К.В., Потапенко А.А. Модификации EM-алгоритма для вероятностного тематического моделирования. Машинное обучение и анализ данных, том 1, № 6, 2013 г., стр. 657–686 / Vorontsov K.V., Potapenko A.A. EM-like Algorithms for Probabilistic Topic Modeling. Machine Learning and Data Analysis, vol. 1, no. 6, 2013, pp. 657-686 (in Russian).

[10]. Asuncion A., Wekking M., Smyth P., Teh Y. W. On Smoothing and Inference for Topic Models. In Proc. of the International Conference on Uncertainty in Artificial Intelligence, 2009, pp. 27-34.

[11]. Newman D., Asuncion A., Smyth P., Welling M. Distributed Algorithms for Topic Models. The Journal of Machine Learning Research, vol. 10, 2009, pp. 1801-1828.

[12]. Wang Y., Bai H., Stanton M., Chen W., Chang E. PLDA: Parallel Latent Dirichlet Allocation for Large-Scale Applications. Lecture Notes in Computer Science, vol. 5564, 2009, pp. 301-314.

[13]. Asuncion A., Smyth P., Welling M. Asynchronous Distributed Estimation of Topic Models for Document Analysis. Statistical Methodology, vol. 8, no. 1, 2010, pp. 3-17.

[14]. Liu Z., Zhang Y., Chang E., Sun M. PLDA+: Parallel Latent Dirichlet Allocation withData Placement and Pipeline Processing. ACM Transactions on Intelligent Systems and Technology, vol. 2, issue 3, article no. 26.

[15]. Smola A., Narayanamurthy S. An architecture for parallel topic models // Proceedings of the VLDB Endowment, 2010. Vol. 3, Issue 1-2. Pp. 703-710.

[16]. Řehůřek R., Sojka P. Software Framework for Topic Modelling with Large Corpora. In Proc. of the LREC 2010 Workshop on New Challenges for NLP Frameworks, 2010, pp. 45-50.

[17]. Zhai K., Boyd-Graber J., Asadi N., Alkhouja M. Mr. LDA: A Flexible Large Scale Topic Modeling Package using Variational Inference in MapReduce. In Proc, of the 21st International Conference on World Wide Web, 2012, pp. 879-888.

[18]. Qiu Z., Wu B., Wang B., Yu L. Collapsed Gibbs Sampling for Latent Dirichlet Allocation on Spark. Proceedings of Machine Learning Research, vol. 36, 2014, pp. 17-28.

[19]. Wang Y., Zhao X., Sun Z., Yan H., Wang L., Jin Z., Wang L., Gao Y., Law C., Zeng J. Peacock: Learning Long-Tail Topic Features for Industrial Applications. ACM Transactions on Intelligent Systems and Technology, 2015, vol. 6, no. 4, article no. 47.

[20]. Yuan J., Gao F., Ho Q., Dai W., Wei J., Zheng X., Xing E., Liu T., Ma W. LightLDA: Big Topic Models on Modest Computer Clusters. In Proc. of the 24th International Conference on World Wide Web, 2015, pp. 1351-1361.

[21]. Zhao B., Zhou H., Li G., Huang Y. ZenLDA: An Efficient and Scalable Topic Model Training System on Distributed Data-Parallel Platform. Big Data Mining and Analytics, vol. 1, issue 1, 2018, pp. 57-74.

[22]. Vorontsov K., Frei O., Apishev M., Romov P., Suvorova M., Yanina A. Non-Bayesian Additive Regularization for Multimodal Topic Modeling of Large Collections. In Proc. of the 2015 Workshop on Topic Models: Post-Processing and Applications, 2015, pp. 29-37.

[23]. Frei O., Apishev M. Parallel Non-blocking Deterministic Algorithm for Online Topic Modeling. Communications in Computer and Information Science, vol. 661, 2016, pp. 132-144.

[24]. Zeng J., Liu Z., Cao X. Fast Online EM for Big Topic Modeling. IEEE Transactions on Knowledge and Data Engineering, vol. 28, issue 3, 2016, pp. 675-688.

[25]. Hoffman M., Blei D., Bach F. Online Learning for Latent Dirichlet Allocation. In Proc. of the 23rd International Conference on Neural Information Processing Systems, vol. 1, 2010, pp. 856-864.

[26]. Vowpal Wabbit ML library. Available at: https://github.com/JohnLangford/vowpal_wabbit, accessed 15.01.2020.

[27]. White T. Hadoop: The definitive guide. O'Reilly Media, Inc., 2012, 688 p.

[28]. Zaharia M., Chowdhury M., Franklin M. Spark: Cluster Computing with Working Sets. In Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing, 2010, 10 p.

[29]. MPI: A Message-Passing Interface Standard, Version 3.0. Forum Message Passing Interface, 2012. Available at: https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf, accessed 15.01.2020.

[30]. Dean J., Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. In Proc. of the 6th Symposium on Operating Systems Design and Implementation, 2004, pp. 137-149.

[31]. Petuum ML platform. Available at: http://www.petuum.com, accessed 15.01.2020.

[32]. Spark GraphX library. Available at: https://spark.apache.org/graphx, accessed 15.01.2020.

Информация об авторах / Information about authors

Мурат Азаматович АПИШЕВ – аспирант факультета вычислительной математики и кибернетики, кафедра математических методов прогнозирования. Сфера научных интересов: машинное обучение, обработка естественного языка, семантические векторные представления текста, тематическое моделирование, параллельные алгоритмы тематического моделирования.

Murat Azamatovich APISHEV – postgraduate student at Faculty of Computational Mathematics and Cybernetics, Department of Mathematical Methods of Forecasting. Research interests: machine learning, natural language processing, text embeddings, topic modeling, parallel algorithms of topic modeling.