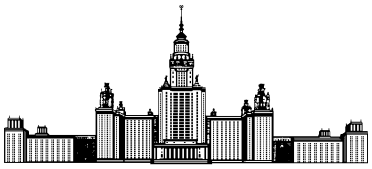

Московский государственный университет имени М. В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Математических Методов Прогнозирования

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Мультимодальные регуляризованные вероятностные тематические модели

Выполнил:
студент 4 курса 417 группы
Апишев Мурат Азаматович

Научный руководитель:
д.ф-м.н., доцент
Воронцов Константин Вячеславович

Москва, 2015

Содержание

Аннотация

Вероятностное тематическое моделирование текстовых коллекций — мощный инструмент для статистического анализа текстов. В данной работе реализована возможность построения мультимодальных регуляризованных тематических моделей на основе параллельного онлайнового EM-алгоритма. Обоснован выбор параллельной архитектуры. Реализован набор регуляризаторов. Реализация выполнена для библиотеки тематического моделирования с открытым кодом BigARTM. Выполнены эксперименты на текстовых коллекциях Wikipedia и EUR-lex, показавшие, что регуляризованные мультимодальные тематические модели в BigARTM позволяют достичь качества, аналогичного или лучшего в сравнении с известными реализациями.

1 Введение

Тематическое моделирование — активно развивающаяся в последние годы область машинного обучения. Оно позволяет решать задачи тематического поиска, категоризации и кластеризации корпусов текстовых документов. В последнее время появляется много приложений тематического моделирования для анализа коллекций изображений и видеозаписей. Тематическая модель определяет, к каким темам относится каждый документ, а также то, какие термины из словаря образуют ту или иную тему.

В работах [1, 2] предлагается полувероятностный подход к тематическому моделированию — *аддитивная регуляризация тематических моделей* (ARTM). Он позволяет записать любое количество дополнительных требований к тематической модели в виде взвешенной суммы критериев, добавляемых к основному функционалу логарифмированного правдоподобия. В [1] показано, что многие известные тематические модели допускают такое представление, то есть фактически являются частными случаями регуляризации. При этом, в отличие от стандартных задач машинного обучения, таких как классификация и регрессия, в тематическом моделировании возникает огромное разнообразие регуляризаторов, направленных на учёт различной дополнительной информации о текстовой коллекции. Задача тематического моделирования является, по сути, многокритериальной, и ARTM позволяет выразить это непосредственным образом. Комбинируя регуляризаторы, ARTM, фактически, позволяет комбинировать различные тематические модели, предлагавшиеся ранее, главным образом, в рамках байесовского подхода к обучению. Реализация такой возможности с помощью байесовской регуляризации представляется сложной математической проблемой и в литературе до сих пор не рассматривалась.

В [4] подход ARTM обощается на *мультимодальные тематические модели*. Многие коллекции, помимо основного текста, несут в себе различную метаинформацию (авторы, метки времени, категории, классы, ссылки и т.п.). Мультимодальные модели позволяют тематически учитывать всю эту информацию.

Благодаря аддитивности регуляризаторов и модальностей появляется новая возможность — строить тематические модели для решения широкого класса задач, выбирая подходящее подмножество из библиотеки регуляризаторов. В данной работе

в качестве основы для такого алгоритмического решения используется библиотека тематического моделирования BigARTM[1]. В ней обработка данных производится с помощью онлайнового параллельного алгоритма, что даёт высокую производительность и возможность обрабатывать большие текстовые коллекции.

В рамках данной работы выполнено обоснования выбора параллельной архитектуры для BigARTM, реализованы механизмы аддитивной регуляризации, мультимодальности и создан начальный набор регуляризаторов. Проведены эксперименты, демонстрирующие новые возможности библиотеки.

Работа имеет следующую структуру.

В разделе 2 описываются теоретические основы ARTM.

В разделе 3 описываются мультимодальне тематические модели.

В разделе 4 описывается онлайновый алгоритм, используемый для обучения тематических моделей в BigARTM.

Раздел 5 посвящён описанию библиотеки BigARTM. В нём даётся краткий обзор существующих параллельных алгоритмов для тематического моделирования, обосновывается выбор архитектуры параллельной обработки для BigARTM, описываются детали реализации механизма аддитивной регуляризации и мультимодальных тематических моделей в BigARTM.

В разделе 6 демонстрируются возможности BigARTM в экспериментах на больших текстовых коллекция.

В разделе 7 резюмируются основные результаты и направления дальнейших исследований.

2 Аддитивная регуляризация тематических моделей

2.1 Постановка задачи и регуляризованный EM-алгоритм

Пусть D — конечный набор текстов (коллекция), и W — конечный набор терминов, из которых состоят эти тексты (словарь). Термином может быть как слово, так и словосочетание. Каждый документ $d \in D$ является последовательностью терминов из W. Предположим, что появление каждого термина в каждом документе связано

[1] http://bigartm.org

с некоторой скрытой переменной из конечного набора тем T. Тогда D представляет собой выборку троек (w_i, d_i, t_i), $i = 1, \ldots, n$, взятых независимо из дискретного распределения $p(w, d, t)$ в пространстве $W \times D \times T$. Термины w_i и документы d_i — наблюдаемые переменные, а темы t_i — скрытые.

Тематическая модель *вероятностного латентного семантического анализа*, PLSA [5] описывает вероятности терминов $p(w|d)$ в каждом документе $d \in D$ смесью вероятностей терминов в темах и тем в документах:

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

Это представление напрямую следует из формулы полной вероятности и предположения об условной независимости $p(w|t) = p(w|d, t)$, означающего, что любая тема генерирует термины вне зависимости от документа.

Параметры $\theta_{td} = p(t|d)$ и $\phi_{wt} = p(w|t)$ составляют матрицы $\Theta = (\theta_{td})_{T \times D}$ и $\Phi = (\phi_{wt})_{W \times T}$. Эти матрицы являются *стохастическими*, т.е. их столбцы являются дискретными вероятностными распределениями. Число тем $|T|$ обычно существенно меньше, чем число слов $|W|$ и число документов $|D|$.

Настройка параметров Φ и Θ по коллекции производится с помощью максимизации логарифма правдоподобия:

$$\mathcal{L}(\Phi, \Theta) = \sum_{d \in D} \sum_{w \in W} n_{dw} \ln p(w|d) \to \max_{\Phi, \Theta},$$

где n_{dw} представляет собой число вхождений термина $w \in W$ в документ d.

Задача стохастического матричного разложения является некорректно поставленной, поэтому в ARTM предлагается ввести r дополнительных критериев $R_i(\Phi, \Theta)$, $i = 1, \ldots, r$, называемых *регуляризаторами*. Производится максимизация их линейной комбинации с весами ρ_i:

$$R(\Phi, \Theta) = \sum_{i=1}^{r} \rho_i R_i(\Phi, \Theta) \to \max_{\Phi, \Theta}.$$

Затем штрафное слагаемое $R(\Phi, \Theta)$ добавляется к логарифму правдоподобия, после чего решается задача условной оптимизации:

$$\mathcal{L}(\Phi, \Theta) + R(\Phi, \Theta) \to \max_{\Phi, \Theta}; \tag{1}$$

$$\sum_{w \in W} \phi_{wt} = 1,\ \phi_{wt} \geq 0; \qquad \sum_{t \in T} \theta_{td} = 1,\ \theta_{td} \geq 0. \tag{2}$$

Локальный максимум Φ, Θ задачи (1), (2) удовлетворяет следующей системе уравнений со вспомогательными переменными $p_{tdw} = p(t|d, w)$, полученной в соответствии с уловиями Каруша-Куна-Такера [2]:

$$p_{tdw} = \underset{t \in T}{\text{norm}}(\phi_{wt}\theta_{td}); \tag{3}$$

$$n_{wt} = \sum_{d \in D} n_{dw} p_{tdw};$$

$$n_{td} = \sum_{w \in d} n_{dw} p_{tdw};$$

$$\phi_{wt} = \underset{w \in W}{\text{norm}} \left(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}} \right); \tag{4}$$

$$\theta_{td} = \underset{t \in T}{\text{norm}} \left(n_{td} + \theta_{td} \frac{\partial R}{\partial \theta_{td}} \right); \tag{5}$$

где оператор $\underset{t \in T}{\text{norm}}(x_t) = \dfrac{\max\{x_t, 0\}}{\sum_{s \in T} \max\{x_s, 0\}}$.

На практике для решения системы уравнений (3)–(5) используется метод простых итераций, эквивалентный EM-алгоритму. Многие байесовские модели (например, LDA и его модификации) являются частными случаями ARTM с различными регуляризаторами [2].

2.2 Примеры регуляризаторов

Регуляризаторы сглаживания/разреживания и декорреляции тем предназначены для повышения разреженности и интерпретируемости тем [2].

Сглаживание/разреживание. Данный регуляризатор минимизирует либо максимизирует KL-дивергенцию[2] между вектор-столбцом θ_d (ϕ_t) и заданным распределением α (β). Применение этого регуляризатора даёт следующие формулы M-шага в EM-алгоритме:

[2]KL-дивергенция (дивергенция Кульбака-Лейблера) между дискретными вероятностными распределениями q и p, имеющими общий носитель, является несимметричной мерой близости этих распределений и определяется как $\text{KL}(q|p) = \sum_i q(i) \ln \dfrac{q(i)}{p(i)}$.

$$\theta_{td} = \underset{t \in T}{\text{norm}}(n_{td} + \alpha_0 \alpha_t), \qquad \phi_{wt} = \underset{w \in W}{\text{norm}}(n_{wt} + \beta_0 \beta_w),$$

где коэффициенты регуляризации α_0 и β_0 определяют степень воздействия регуляризатора на модель. При этом отрицательные значения коэффициентов соответствуют разреживанию, положительные — сглаживанию.

Обобщением этого регуляризатора является регуляризатор для *частичного обучения*, который позволяет задавать подмножества слов с ненулевыми β_{wt} для каждой темы t и подмножества тем t с ненулевыми α_{td} для каждого документа.

Декорреляция тем. Уменьшение корреляций между вектор-столбцами, соответсвующими разным темам, приводит к повышению различности и, как следствие, интерпретируемости тем. Формула М-шага для такого регуляризатора матрицы Ф имеет вид

$$\phi_{wt} = \underset{w \in W}{\text{norm}} \left(n_{wt} - \rho \phi_{wt} \sum_{s \in T \setminus t} \phi_{ws} \right).$$

Данный регуляризатор является разреживающим, величина эффекта разреживания определяется значением коэффициента регуляризации ρ.

3 Мультимодальные тематические модели

3.1 Мультимодальный регуляризованный ЕМ-алгоритм

Теперь предположим, что документ содержит не только слова, но ещё и термины других модальностей. Каждая модальность определяется конченым набором терминов (словарём) W^m, $m = 1, \ldots, M$.

Примерами модальностей могут служить авторы, метки классов или категорий, отметки времени, ссылки на другие документы, и т.п.

Как и в предыдущем разделе, коллекция представляет собой выборку независимых и одинаково распределённых троек $(w_i, d_i, t_i) \sim p(w, d, t)$, взятых из конечного вероятностного пространства $W \times D \times T$, но теперь $W = W^1 \sqcup \cdots \sqcup W^M$ является объединением словарей всех модальностей.

Для каждой модальности W^m, $m = 1, \ldots, M$ строится тематическая модель:

$$p(w|d) = \sum_{t \in T} p(w|t)p(t|d) = \sum_{t \in T} \phi_{wt}\theta_{td},\ w \in W^m.$$

Стохастические матрицы $\Phi^m = (\phi_{wt})_{W^m \times T}$ вероятностей терминов в темах, составленные вертикально, образуют матрицу Φ размера $|W| \times |T|$.

Для настройки параметров Φ^m и Θ мультимодальной коллекции максимизируется логарифм правдоподобия для каждой m-й модальности:

$$\mathcal{L}_m(\Phi^m, \Theta) = \sum_{d \in D} \sum_{w \in W^m} n_{dw} \ln p(w|d) \to \max_{\Phi^m, \Theta},$$

где n_{dw} — число вхождений термина $w \in W^m$ в документ d. Распределения тем в документах Θ являются общими для всех модальностей.

Следуя ARTM, добавим к логарифму правдоподобия регуляризатор $R(\Phi, \Theta)$. Получим задачу условной оптимизации с ограничениями:

$$\sum_{m=1}^{M} \tau_m \mathcal{L}_m(\Phi^m, \Theta) + R(\Phi, \Theta) \to \max_{\Phi, \Theta}; \tag{6}$$

$$\sum_{w \in W^m} \phi_{wt} = 1,\ \phi_{wt} \geq 0; \qquad \sum_{t \in T} \theta_{td} = 1,\ \theta_{td} \geq 0; \tag{7}$$

где *коэффициенты регуляризации* τ_m используются для регулирования степени важности различных модальностей.

Локальный максимум (Φ, Θ) задачи (6), (7) удовлетворяет системе уравнений с временными переменными $p_{tdw} = p(t|d, w)$:

$$p_{tdw} = \underset{t \in T}{\text{norm}}(\phi_{wt}\theta_{td}); \tag{8}$$

$$n_{wt} = \sum_{d \in D} \tau_{m(w)} n_{dw} p_{tdw};$$

$$n_{td} = \sum_{w \in d} \tau_{m(w)} n_{dw} p_{tdw};$$

$$\phi_{wt} = \underset{w \in W^m}{\text{norm}} \left(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}} \right); \tag{9}$$

$$\theta_{td} = \underset{t \in T}{\text{norm}} \left(n_{td} + \theta_{td} \frac{\partial R}{\partial \theta_{td}} \right); \tag{10}$$

где оператор $m(w)$ — модальность термина w, $w \in W^{m(w)}$. Доказательство данного факта можно найти в [4]. Данная задача при $M = 1$ переходит в задачу (1), (2).

Алгоритм 4.1: Онлайновый EM-algorithm для мультимодального ARTM

Входные данные: коллекция D, коэффициент забывания $\rho \in (0, 1]$;

Выходные данные: матрица Φ;

1 инициализировать ϕ_{wt} для всех $w \in W$ и $t \in T$;

2 $n_{wt} := 0,\ \tilde{n}_{wt} := 0$ для всех $w \in W$ и $t \in T$;

3 **для каждого** *пакета* D_b, $b = 1, \ldots, B$ **выполнять**

4 $\quad (\tilde{n}_{wt}) := (\tilde{n}_{wt}) + \mathsf{ОбработаткаПакета}(D_b, \phi_{wt})$;

5 $\quad$ **если** *инициирована синхронизация* **тогда**

6 $\quad\quad n_{wt} := \rho n_{wt} + \tilde{n}_{wt}$ для всех $w \in W$ и $t \in T$;

7 $\quad\quad \phi_{wt} := \underset{w \in W^m}{\operatorname{norm}}\left(n_{wt} + \phi_{wt}\frac{\partial R}{\partial \phi_{wt}}\right)$ для всех $w \in W^m,\ m = 1, \ldots, M$ и $t \in T$;

8 $\quad\quad \tilde{n}_{wt} := 0$ для всех $w \in W$ и $t \in T$;

3.2 Регуляризатор для балансировки классов

Регуляризатор для балансирования классов (Label regularization [13]) улучшает качество тематической модели классификации в задачах с большим числом несбалансированных и пересекающихся классов. Пусть $\hat{p}_c$ — частота объектов класса c в обучающей коллекции, W^2 — множество меток классов, а Φ^2 — матрица «слова – темы», соответствующая модальности меток классов. Тогда формула M-шага для данного регуляризатора примет вид

$$\phi^2_{ct} = \underset{c \in W^2}{\operatorname{norm}}\left(n_{ct} + \tau_2 \hat{p}_c \frac{\phi^2_{ct} n_t}{\sum_{s \in T} \phi^2_{cs} n_s}\right).$$

4 Онлайновый EM-алгоритм

Для обработки больших коллекций итерации EM-алгоритма должны быть организованы так, чтобы избежать хранения трёхмерной матрицы p_{tdw}, а также большой матрицы Θ, размер которой пропорционален размеру коллекции. Кроме того, за один проход по коллекции параметры ϕ_{wt} успевают хорошо настроиться, при этом значения θ_{td} остаются далеки от оптимальных. Поэтому в таком случае лучше использовать *онлайновый пакетный EM-алгоритм*. Как и в алгоритме Online LDA [6],

Алгоритм 4.2: ОбработкаПакета(D_b, ϕ_{wt})

Входные данные: пакет D_b, матрица ϕ_{wt};

Выходные данные: матрица $(\tilde{n}_{wt})$;

1 $\tilde{n}_{wt} := 0$ для всех $w \in W$ и $t \in T$;

2 **для каждого** $d \in D_b$ **выполнять**

3 инициализировать $\theta_{td} := \frac{1}{|T|}$ для всех $t \in T$;

4 **повторять**

5 $p_{tdw} := \underset{t \in T}{\text{norm}}\big(\phi_{wt}\theta_{td}\big)$ для всех $w \in d$ и $t \in T$;

6 $n_{td} := \sum_{w \in d} \tau_{m(w)} n_{dw} p_{tdw}$ для всех $t \in T$;

7 $\theta_{td} := \underset{t \in T}{\text{norm}}\big(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}\big)$ для всех $t \in T$;

8 **до тех пор, пока** θ_d *не сойдётся*;

9 увеличить $\tilde{n}_{wt}$ на $\tau_{m(w)} n_{dw} p_{tdw}$ для всех $w \in d$ и $t \in T$;

коллекция D разделяется на пакеты D_b, $b = 1, \ldots, B$, и итерации EM-алгоритма организуются так, чтобы каждый вектор θ_d итерировался до сходимости при неизменном текущем приближении матрицы Φ. Переменные p_{tdw} вычисляются в момент, когда они необходимы и не хранятся. Вместо всей матрицы Θ храниться только вектор θ текущего обрабатываемого документа. Матрица Φ обновляется достаточно редко, например, после обработки каждого пакета или нескольких пакетов. Листинг 4.1, демонстрирующий описанный алгоритм, не фиксирует частоту обновления Φ на шагах 5–8. Эта гибкость особенно важна для параллельной реализации алгоритма, когда множество пакетов обрабатываются одновременно. В этом случае обновление можно производить после того, как со времени последней синхронизации было обработано какое-то количество документов.

Онлайновый алгоритм является реорганизацией шагов вычислений в EM-алгоритме и никак не связан с байесовским выводом, использовавшимся в [6] при описании алгоритма Online LDA. Различные тематические модели, от PLSA до мультимодальных и регуляризованных, могут быть обучены помощью предлагаемого онлайнового EM-алгоритма.

5 Библиотека BigARTM

Библиотека тематического моделирования BigARTM реализует описанную выше концепцию аддивной регуляризации мультимодальных тематических моделей. В её основе лежит онлайновый пакетный EM-алгоритм, в рамках которого осуществляется параллельная обработка пакетов документов.

5.1 Параллельная архитектура

BigARTM задумывалась как инструмент для тематического моделирования больших коллекций документов, поэтому вопрос эффективности обработки был ключевым при разработке и реализации её архитектуры. Прежде всего был изучен ряд ранних работ, связанных с параллельными и распределёнными реализациями алгоритмов тематического моделирования. Ниже приводится обзор этих работ и даётся обоснование выбора параллельной архитектуры BigARTM.

5.1.1 Обзор существующих реализаций

AD-LDA. Approximate Distributed LDA (AD-LDA) был предложен в [7]. Данный алгоритм основан на коллапсированной схеме Гиббса. Коллекция D распределяется по имеющимся процессорам примерно в равных пропроциях, после чего каждый из процессоров производит сэмплирование тем слов в своих документах. Копия глобальных счётчиков n_{wt} при этом имеется на каждом ядре. По окончании обработки данных всеми процессорами производится синхронизация, в ходе которой полученные на каждом процессоре локальные счётчики добавляются в глобальную таблицу n_{wt}.

Описанная схема имеет ряд существенных недостатков. Качество модели существенно зависит от частоты синхронизаций. Помимо этого, время, затрачиваемое на одну итерацию работы алгоритма, определяется самым медленным из процессоров. Сеть во время итераций простаивает, а во время синхронизаций — перегружена. Наконец, хранение копии глобальной n_{wt} на каждом ядре является очень расточительным с точки зрения расходуемой памяти (особенно если учитывать, что в экспериментах использовалось до 1024 ядер).

PLDA+. Ранняя версия этого алгоритма, PLDA ([8]), представляла собой модификацию AD-LDA, сохранившую все его ключевые недостатки. В [9] был представлен обновлённый вариант реализации. В PLDA+ был ликвидирован основной источник проблем — выделенный шаг синхронизации. Для этого все процессоры, задействованные в работе алгоритма, разделяются на две группы. Задача первых состоит в непосредственной обработке документов, вторая же группа занимается своевременной доставкой данных. Как следствие, производительность системы выросла, хотя появился новый недостаток — существенная часть вычислительных ресурсов процессоров тратится на перенос данных, а не на их обработку.

Y!LDA. Этот алгоритм, представленный в [10] и также основанный на сэмлировании Гиббса, показал, что можно относительно недорого избавиться от выделенного шага синхронизации. Коллекция D распределяется по узлам, а на узле — по процессорам. В рамках каждого узла создаётся несколько потоков, занимающихся обработкой документов, и один поток, производящий обновление глобальных счётчиков. К этому потоку обработчики обращаются асинхронно по мере готовности новых обновлений. Параллелизм, основанный на многопоточности, позволил хранить копию глобальных счётчиков не на каждом ядре, а только на каждом узле. Синхронизация состояний всех узлов организована на основе т.н. *архитектуры классной доски*. Суть её в том, что глобальное состояние хранится в некоторой общей для всех узлов памяти, и поток синхронизации каждого узла асинхронно обращается к ней и производит обновление. Полученная система может производить параллельную обработку документов, эфективно используя имеющиеся вычислительные и сетевые ресурсы.

Mr. LDA. Предложенный в [11] алгоритм основан на вариационном выводе и реализован в рамках MapReduce на Hadoop. Использование вариационного вывода мотивировалось его совместимостью с MapReduce, ликвидацией проблемы частых синхронизаций, дерминированностью результатов и лучшей сходимостью в сравнении со сэмплированием Гиббса. Схема работы следующая: на шаге Map производится вывод гиперпараметров, связанных с документами, а на шаге Reduce — с темами. Алгоритм расширяем (в работе был приведён пример создания мультиязычной те-

матической модели), в нём организована оптимизация гиперпараметров, MapReduce обеспечивает хорошую отказоустойчивость.

Gensim. Данная библиотека предоставляет две реализации LDA на языке Python: LdaModel и LdaMulticore [12]. Первая из них почти в точности повторяет Online LDA из [6] и не является ни параллельной, ни распределённой. Тем не менее, LdaModel работает достаточно эффективно за счёт использования матричных вычислений в библиотеке NumPy. LdaMulticore же является параллельным, архитектурно он имеет много общего с Y!LDA. Однако масштабируемость этого алгоритма оставляет желать лучшего — эксперименты показали, что прирост производительности прекращается начиная с 4-х ядер [4].

5.1.2 Обоснование выбора параллельной архитектуры в BigARTM

В результате изучения описанных выше работ к архитектуре библиотеки были предъявлены следующие требования:

1. асинхронная обработка данных;
2. минимизация объёма используемой оперативной памяти;
3. хорошая масштабируемость при увеличении количества ядер на узле;
4. кроссплатформенность;
5. возможность быстрой установки и использования на одной машине.

В соответствии с описанными требованиями было принято решение не использовать MapReduce. Во-первых, обработка данных в рамках этого подхода является синхронной, во-вторых, такое решение плохо подходит для использования на одном компьютере. MapReduce прежде всего ориентирован на применение простых операций к большим объёмам данных, для реализации итеративных алгоритмов машинного обучения он слишком негибкий.

Таким образом, возникла необходимость «ручной» организации параллельной обработки данных на одном узле. Наилучшим решением оказалось использование многопоточного параллелизма в пределах одного процесса. Это позволяет как получить

хорошую скорость обработки, так и хранить информацию о модели в рамках узла, а не каждого ядра. Кроме того, такое решение обеспечивает возможность асинхронной работы с данными.

Для написания кода ядра библиотеки был выбран язык C++. Сделано это было как из соображений скорости, так и для кроссплатформенности.

Вопрос об способе организации распределённой обработки с помощью BigARTM остаётся открытым. На данный момент она осуществляется на основе механизма rpc. Это даёт большую гибкость в проектировании сетевых взаимодействий. Тем не менее, в будущем предполагается использование специализированных средств, таких как Hadoop 2.0, Spark и т.п.

5.1.3 Архитектура BigARTM

BigARTM — библиотека для эффективной онлайновой параллельной обработки прежде всего в пределах одной машины. Как уже было сказано, одной из основных целей, преследуемых при её проектировании, было достижение независимости объёма используемой оперативной памяти от размера обрабатываемой коллекции. Достигается это следующим образом — вся коллекция D, как и описывалось ранее, делится на пакеты D_b, сохраняется на диск в отдельных файлах, и в каждый момент времени в памяти находится только часть из них. Вся матрица Θ никогда не хранится. В каждый момент времени в памяти содержатся только те её куски, которые соответствуют обрабатываемым документам, и после окончания обработки они удаляются.

Организация параллельной обработки в BigARTM, как в Y!LDA и Gensim, основана на многопоточности. Общая схема показана на рис. 1. Существует множество *обработчиков* (Processor) и один *поток слияния* (Merger). Задача первых — производить параллельную обработку пакетов документов, выводя элементы матрицы Θ и рассчитывая обновления для матрицы Φ, второго — получать от первых асинхронно эти обновления и обновлять матрицу Φ. Для обработчиков существует т.н. *очередь заданий*, в которых хранятся подгруженные библиотекой в память пакеты. Оттуда обработчики берут задания. Также существует *очередь слияния*, в которую обработчики отправляют по мере необходимости обновления для матрицы Φ. В свою

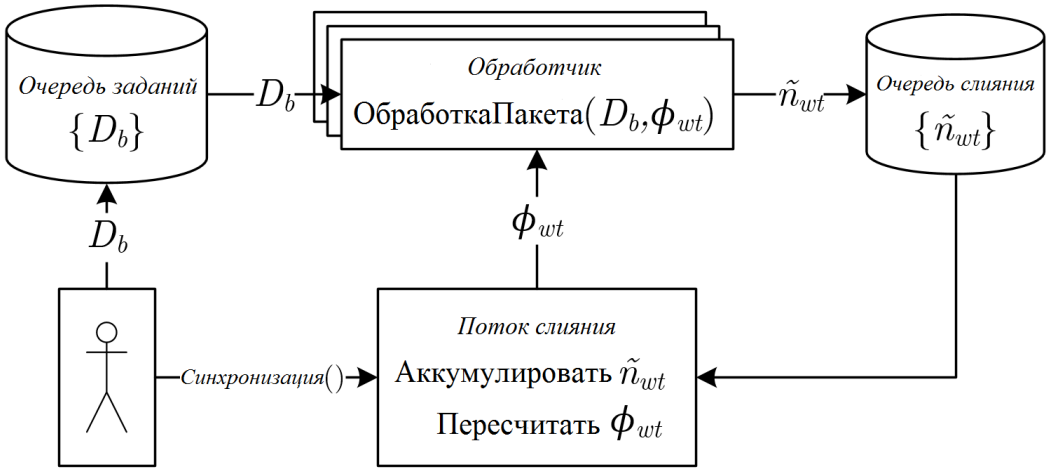

Рис. 1: Схема обработки данных в BigARTM.

очередь, поток Merger извлекает из этой очереди данные, которые нужны ему для работы. Все данные хранятся в памяти статично и не копируются, «перемещения» производятся с помощью указателей. В каждый момент времени Merger хранит две копии матрицы Φ — *базовую* и *активную*. Первая из них является доступной для чтения обработчикам, вторая — всегда доступна для записи потоку слияния. Каждый обработчик перед началом работы берёт себе указатель на текущую активную матрицу Φ и использует её для вывода векторов в $\boldsymbol{\theta}_d$. Merger, получая обновления из очереди слияния, обновляет базовую матрицу. Как только все текущие обновления завершаются, она становится активной, и Merger создаёт новую базовую матрицу[3]. Обновление матрицы Φ (синхронизацию) можно инициировать в любой момент из пользовательского кода.

5.2 Реализация механизма регуляризации

В BigARTM все регуляризаторы подразделяются на два типа — для матрицы Θ и для матрицы Φ[4]. На уровне реализации они представляют собой отдельные модули с унифицированным интерфейсом. Эти модули имеют методы регуляризации, которые

[3]На самом деле, в некоторые моменты времени существует *три* матрицы Φ — старая активная, которая существует до тех пор, пока есть обработчики, использующие её, новая активная, которая передаётся новым обработчикам и базовая.

[4]Под Φ, как будет видно далее, подразумеваются все Φ_i.

получают на вход элементы соответствующей матрицы и дополнительные данные, после чего производят вычисление требуемой аддитивной добавки.

Регуляризаторы Θ. Регуляризация элементов матрицы Θ производится на каждой итерации цикла 4–7 алгоритма 4.2, т.е. при каждом проходе по обрабатываемому документу. Перед началом обработки очередного пакета документов инициализируется набор регуляризаторов, которые будут к нему применены. В конце каждого прохода по каждому документу производится вызов соответствующих методов всех регуляризаторов, которые вычисляют поправки. В конце этого процесса все эти поправки применяются к вектор-столбцу, соответствующему документу, производятся операции нормировки и обнуления отрицательных значений.

Регуляризаторы Φ. Процедура регуляризации Φ производится при каждой её синхронизации. Как только все текущие обновления, поступившие из очереди слияния, были прибавлены к матрице, вызываются методы регуляризаторов Φ, рассчитывающие необходимые добавки для каждого её элемента. После этого все полученные добавки прибавляются к Φ, производится нормировка и обнуления отрицательных значений.

На данный момент в библиотеке реализованы регуляризаторы сглаживания/разреживания, декорреляции тем и балансирования классов, описанные выше. Для каждого регуляризатора можно определить подмножество тем, на котором он будет действовать.

Помимо использования существующих регуляризаторов, всегда имеется возможность добавить в библиотеку новые. Процесс добавления регуляризаторов в билиотеку BigARTM описан в Приложении А.

5.3 Реализация поддержки мультимодальных моделей

Обозначим r_{wt} матрицу, каждый элемент которой представляет собой сумму добавок, полученных в результате применения всех регуляризаторов, работающих с этим элементом.

Рассмотрим способ хранения тематической модели, представленной матрицей счётчиков n_{wt}, матрицей добавок r_{wt} и вектором нормировочных констант для каж-

дой из тем n_t. Матрица Φ получается из этой структуры путём сложения всех соответствующих n_{wt} и r_{wt}, обнуления всех отрицательных значений и нормировки полученного результата по столбцам на n_t. Идентификация терминов до введения мультимодальности в BigARTM производилась по их текстовым представлениям.

Для реализации мультимодальных моделей в описанную выше схему было внесено два изменения. Во-первых, термин стал идентифицироваться с помощью пары строк — его текстового представления и имени модальности, к которой этот термин относится. Таким образом, n_{wt} всех терминов w всех модальностей физически хранятся в одной структуре и логически поделены на Φ_i с помощью идентификаторов модальностей в именах терминов. Вторым изменением стала замена вектора n_t на набор таких векторов, по одному на каждую из модальностей. Затем был дорабон поток Merger, чтобы он обновлял все матрицы Φ_i, и модифицирован код обработчиков, чтобы они рассчитывали θ_d, используя все модальности.

Видно, что переход к мультимодальным моделям не усложнил механизма регуляризации. Для регуляризаторов матриц Θ не изменилось вообще ничего. Регуляризаторы Φ претерпели единственное изменение — в процессе работы они должны проверять, что термин, счётчик n_{wt} которого они хотят модифицировать, принадлежит нужной модальности.

6 Вычислительные эксперименты

Проведены два эксперимента с регуляризованными тематическими моделями. В первом строятся обычные (унимодальные) тематические модели большой текстовой коллекции, обучаемые с помощью LDA и ARTM, после чего производится их сравнение по нескольким критериям. Второй эксперимент проводится на относительно небольшой коллекции документов с метками классов и направлен на получение качественной мультимодальной тематической модели классификации.

6.1 Эксперименты с регуляризаторами

Цель данного эксперимента — показать возможности комбинирования регуляризаторов и однопроходной обработки большой коллекции в BigARTM, а также

продемонстрировать, что онлайн-реализация ARTM даёт результаты, аналогичные офлайн-реализации [4]. Оба метода обучения моделировались на BigARTM: LDA — запуском обучения со сглаживанием Φ и Θ, ARTM — с комбинацией регуляризаторов.

6.1.1 Используемые данные и метрики качества

Используется коллекция Wikipedia статей английской Википедии, $|D| \approx 3.7 \times 10^6$, полученная с помощью gensim.make wikicorpus script (как и в [4]), который удалил все страницы, не являющиеся статьями, а также статьи, содержащие менее 50 слов. Кроме того, были удалены все слова из словаря, встречающиеся менее чем в 20 документах или более чем в 10% документов. Итоговый размер словаря составил $|W| \approx 10^5$ терминов. Суммарный размер Wikipedia в словах — примерно 577×10^6. Модель коллекции обучается онлайновым алгоритмом за один проход по коллекции. При этом словарь W считается известным заранее, а не собирается по ходу работы алгоритма. Критериями качества моделирования являются перплексия[5] на отложенной выборке, разреженности матриц Φ и Θ, а также *характеристики ядер тем* (*размер*, *контрастность* и *чистота*), описанные в [2]. Эти величины характеризуют каждую тему, поэтому для оценивания качества модели в целом берётся их среднее значение по всем темам. Средние контрастность и чистота ядер тем модели позволяют оценивать её интерпретируемость.

6.1.2 Параметры эксперимента и результаты

Параметры эксперимента:

- количество тем $|T| = 100$;
- 20 проходов по каждому документу;
- размер пакета $|D_b| = 10000$ документов;
- коэффициент забывания $\rho = (\tau_0 + t)^{-\kappa},\ \tau_0 = 64,\ \kappa = 0.5,\ t$ — отношение числа обработанных документов к размеру пакета.

[5]Перплексия определяется как $\mathcal{P}(D; p) = \exp\big(-\frac{1}{n} L(\Phi, \Theta)\big)$, где L — логарифм правдоподобия коллекции, n — суммарное число слов в коллекции.

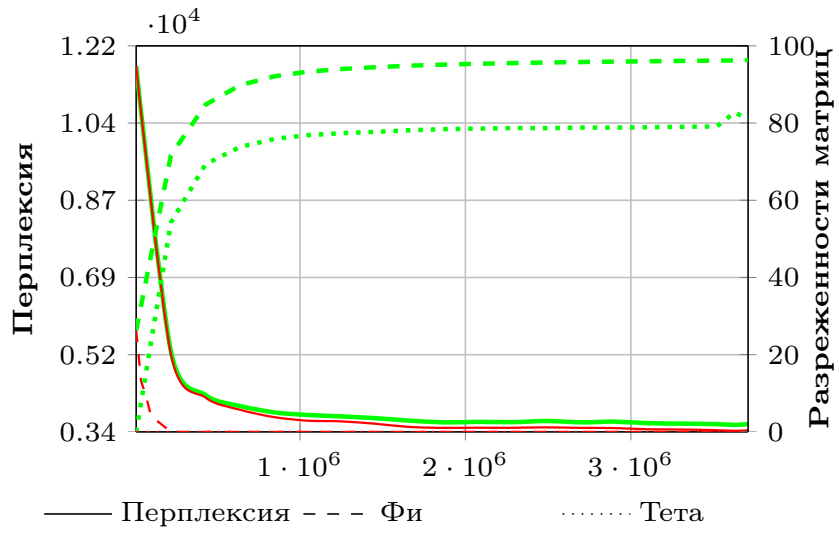

Рис. 2: График перплексии и разреженностей матриц Φ и Θ для моделей LDA (тонкие красные линии) и ARTM (жирные зелёные линии).

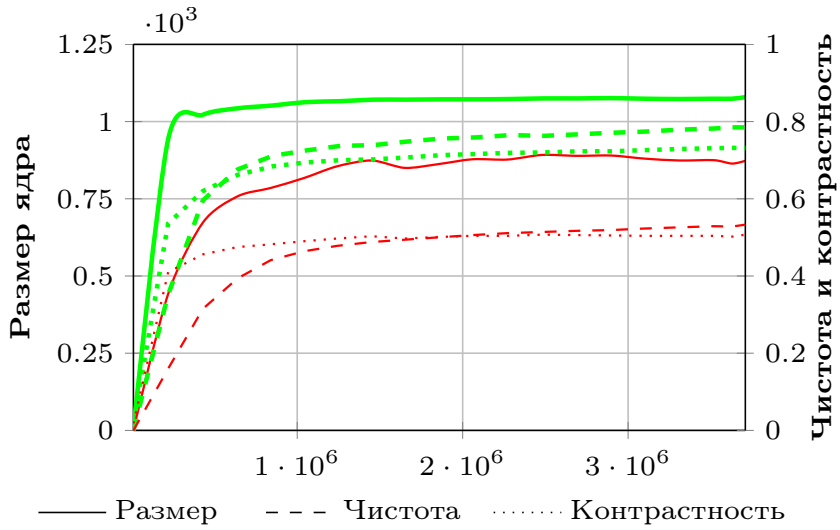

Рис. 3: График характеристик ядер тем для моделей LDA (тонкие красные линии) и ARTM (жирные зелёные линии).

- 1 поток–обработчик;
- гиперпараметры LDA $\alpha_{LDA} = \beta_{LDA} = \frac{1}{|T|} = 0.01$;
- для модели ARTM использовалась комбинация регуляризаторов разреживания Θ и Φ и декорреляции тем в Φ с коэффициентами -0.15, -0.011 и 5.8×10^5 соответственно (описанные в подразделе 2.2 дискретные распределения α и β взяты равномерными);

Таблица 1: Сравнение моделей ARTM и LDA. Метрики качества: $\mathcal{P}_{10k}$, $\mathcal{P}_{100k}$ — перплексия на отложенных выборках в 10 тыс. и 100 тыс. документов, $\mathcal{S}_\Phi$, $\mathcal{S}_\Theta$ — разреженность матриц Φ и Θ (в %), $\mathcal{K}_s$, $\mathcal{K}_p$, $\mathcal{K}_c$ — средние размер, чистота и контрастность ядер тем.

Model	$\mathcal{P}_{10k}$	$\mathcal{P}_{100k}$	$\mathcal{S}_\Phi$	$\mathcal{S}_\Theta$	$\mathcal{K}_s$	$\mathcal{K}_p$	$\mathcal{K}_c$
LDA	3436	3801	0.0	0.0	873	0.533	0.507
ARTM	3577	3947	96.3	80.9	1079	0.785	0.731

- перплексия рассчитывалась на одном контрольном пакете; кроме того, финальные её значения для обеих моделей были посчитаны на выборке из 10^5 документов.

Результаты показаны на рис. 2, рис. 3 и в таблице 1.

Видно, что при незначительном ухудшении перплексии ARTM дала модель, существенно лучшую по всем остальным параметрам. Модель LDA не обеспечивает разреженность, интерпретируемость, показателями которой служат контрастность и чистота ядер тем, также существенно ниже, чем у ARTM.

6.2 Эксперименты с тематической моделью классификации

Использование тематических моделей для классификации текстовых документов предпочтительно в случаях большого числа несбалансированных, пересекающихся, взаимнозависимых классов. В [13] показано, что тематические модели способны решать такие задачи лучше, чем SVM. Цель данного эксперимента — показать, что мультимодальные регуляризованные тематические модели в реализации BigARTM справляются с задачей не хуже, чем алгоритм Dependency LDA, предоженный в [13].

6.2.1 Используемые данные и метрики качества

Эксперимент производился на коллекции EUR-lex, содержащей около 20 тысяч документов. Словарь коллекции составляет $|W| \approx 1.9 \times 10^5$ слов. После предобработки, предложенной в [13], в ходе которой были удалены все слова, встретившиеся во всей коллекции менее чем 20 раз, W сократился до $\approx 2 \times 10^4$ слов. Число классов, по которым разбиты документы коллекции, примерно равно 3950. При этом каждый

документ может относится к нескольким классам одновременно. В ходе предобработки были удалены метки классов, которые встретились во всей коллекции один раз (таких оказалось около 700).

В экспериментах использовались следующие метрики качества, считаемые по отложенной выборке:

- Площадь под кривой precision-recall.
- Площадь по ROC-кривой.
- Доля документов, у которых самая вероятная метка неверная.
- Доля документов, не классифицированных полностью правильно.

Для устранения эффектов выбросов коллекция десятикратно разбивается на десять примерно равных частей, и каждый раз девять частей используются в качестве обучающей выборки, а одна — в качестве контрольной. Финальные результаты усредняются по всем десяти разбиениям. На каждом разбиении из выборки дополнительно удалялись метки классов, которые встречались в тестовой выборке и не встречались в обучающей (однако их число всегда было незначительным, 4–6 меток).

Модель коллекции EUR-lex, из-за её относительно небольшого размера, обучается с помощью многократного итерирования по всей коллекции, т.е. несмотря на онлайновость алгоритма с технической точки зрения, используется он в оффлайн-режиме, хотя и с многократным проходом по каждому документу.

6.2.2 Параметры эксперимента и результаты

В эксперименте производилось сравнение точности классификации моделей мультимодальной ARTM (M-ARTM), Dependency LDA [13] и наилучших результатов, полученных с помощью SVM (последние два — по данным из [13]). Параметры M-ARTM — число тем, набор регуляризаторов и их коэффициентов, число проходов по коллекции и документу — подбирались экспериментально. Ниже приведён список финальных оптимальных параметров, с которыми производилось обучение M-ARTM для классификации[6]:

[6]Напомним, что в тематической модели классификации матрица «слова – темы» для меток классов была обозначена Φ^2.

- количество тем $|T| = 15000$[7] ;
- 4 прохода по коллекции;
- 25 проходов по каждому документу на первых трёх проходах по коллекции, 22 — на последнем;
- 1 поток–обработчик;
- вес модальности «слова» при пересчёте векторов θ на каждом проходе по коллекции: 5, 10, 15 и 20;
- вес модальности «метки классов» при пересчёте векторов θ на каждом проходе по коллекции: 0.3, 0.4, 0.6 и 0.8;
- использовалась комбинация регуляризаторов сглаживания равномерным распределением Θ и Φ^2 с коэффициентами 0.001 и 0.0075 соответственно, а также Label Regularization для матрицы Φ^2 с постоянным коэффициентом 3×10^6;

Финальные и некоторые промежуточные результаты приведены в таблицах 3 и 2[8]. Видно, что наилучшее качество по всем метрикам в M-ARTM было достигнуто при использовании комбинации сглаживания по равномерному распределению и Label Regularization при числе тем $|T| = 10000$. Меньшее число тем и отсутствие сглаживающих регуляризаторов приводят к ухудшению результата.

Стоит отметить, что все использованные стратегии разреживания матриц только ухудшали результат по сравнению с отсутствием регуляризации. Это подтверждает гипотезу о том, что для задач с маленькими по объёму классами лучше использовать сглаживание, поскольку разреживание приводит к потери нужной информации.

[7]Такое сравнительно большое число тем оказалось оптимальным в ходе экспериментов. Для DLDA оно существенно меньше (200). Тем не менее, говорить о том, что M-ARTM даёт преимущества исключительно за счёт числа тем, нельзя, поскольку число тем в DLDA также подбиралось по сетке значений, и $|T| = 200$ признано его авторами оптимальным.

[8]На самом деле, в [13] исследовались две модели SVM: настроенная и с параметрами по-умолчанию. Каждая из них была лучше по какой-то из метрик, здесь для наглядности показывается лучший результат для обеих моделей.

Таблица 2: Сравнение моделей классификации M-ARTM, Dependency LDA и SVM. Метрики качества: AUC_{PR} — площадь под кривой precision-recall, AUC_{ROC} — площадь под ROC-кривой, One Error — доля документов, в которых наиболее вероятная метка оказалась неверной, Is Error — доля документов, не классифицированных идеально верно. Наилучшие результаты выделены жирным шрифтом. $|T|_{optimal}$ — оптимальное число тем для тематической модели.

	$\|T\|_{optimal}$	$\text{AUC}_{\text{PR}} \Uparrow$	$\text{AUC}_{\text{ROC}} \Uparrow$	One Error $\Downarrow$	Is Error $\Downarrow$
M-ARTM (без регуляризации)	15000	0.524	0.978	28.2	94.3
M-ARTM (с регуляризацией)	15000	**0.529**	0.980	**27.1**	**94.2**
DLDA	200	0.492	**0.982**	32.0	97.2
SVM_{best}	–	0.435	0.975	31.6	98.1

Таблица 3: Качество классификации модели M-ARTM с различным числом тем (без регуляризации).

$\|T\|$	$\text{AUC}_{\text{PR}} \Uparrow$	$\text{AUC}_{\text{ROC}} \Uparrow$	One Error $\Downarrow$	Is Error $\Downarrow$
100	0.227	0.956	61.7	99.9
500	0.315	0.964	49.5	99.7
1000	0.373	0.970	43.8	99.0
2000	0.409	0.973	38.6	98.8
5000	0.459	0.976	33.8	97.3
10000	0.508	0.978	30.2	95.3
12000	0.515	0.978	29.0	94.9
15000	0.524	0.978	28.2	94.3
18000	0.526	0.977	28.2	94.3

Кроме того, было проверено экспериментально, что дальнейшее увеличение тем оказалось нецелесообразным.

Также можно сделать вывод, что по трём характеристикам M-ARTM обходит как DLDA, так и лучший результат SVM. По характеристике площади под ROC-кривой M-ARTM немного уступает DLDA, но значительно превосходит SVM.

7 Заключение

7.1 Результаты, выносимые на защиту

- Обоснование стратегии параллельной обработки данных в библиотеке BigARTM на основе обзора существующих параллельных, распределённых и онлайновых реализаций.
- Реализация в библиотеке механизмов и интерфейсов регуляризации, а также базового набора регуляризаторов, эффективность которых при использовании онлайнового алгоритма была экспериментально подтверждена.
- Реализация мультимодального обобщения EM-алгоритма в BigARTM, благодаря которому появилась возможность создания сложных тематических моделей, учитывающих произвольное число модальностей слов. Создание качественной тематической модели классификации на основе экспериментального подбора траектории регуляризации.

7.2 Выводы и направления дальнейших исследований

Библиотека BigARTM достаточно легко расширяема, эффективна с вычислительной точки зрения. В совокупности с ARTM и поддержкой мультимодальных моделей это даёт очень гибкий, производительный и простой в использовании инструмент для тематического моделирования текстовых коллекций в самых различных задачах и приложениях. Библиотека имеет большой потенциал, в будущем запланирован ряд усовершенствования, как технических (вычисления на GPU и полноценная распределённость на кластере), так и алгоритмических.

Список литературы

[1] К. Воронцов Аддитивная Регуляризация Тематических Моделей Коллекций Текстовых Документов *Доклады РАН, Т.455, №3. С.268-271*, 2014.

[2] A. Potapenko, K. Vorontsov Tutorial on Probabilistic Topic Modeling: Additive Regularization for Stochastic Matrix Factorization. *AIST, Analysis of Images, Social networks and Texts. – Springer International Publishing Switzerland, 2014. Communications in Computer and Information Science (CCIS), Vol. 436. pp. 29-46.*, 2014.

[3] D. Blei., A. Ng, M. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research., 3 (4-5): pp. 993-1022*, 2003.

[4] K. Vorontsov, O. Frei, M. Apishev., P. Romov, M. Dudarenko. BigARTM: Open Source Library for Regularized Multimodal Topic Modeling of Large Collections. *The 4th International Conference on Analysis of Images, Social Networks, and Texts (to appear)*, 2015.

[5] T. Hofmann. Probabilistic Latent Semantic Indexing. *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval., Pp. 50-57*, 1999.

[6] M. Hoffman, D. Blei, F. Bach. Online Learning for Latent Dirichlet Allocation. *Neural Information Processing Systems — Curran Associates, Inc., page 856-864*, 2010.

[7] P. Smyth D. Newman, A. Asuncion, M. Welling. Distributed Algorithms for Topic Models. *Neural Information Processing Systems — The Journal of Machine Learning Research, Vol. 10, pp. 1801-1828*, 2009.

[8] M. Stanton W. Chen Y. Wang, H. Bai, E. Chang. PLDA: Parallel Latent Dirichlet Allocation for Large-scale Applications. *AAIM '09 Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management, pp. 301-314*, 2009.

[9] Z. Liu, Y. Zhang, E. Chang, M. Sun. PLDA+: Parallel Latent Dirichlet Allocation with Data Placement and Pipeline Processing. *ACM Transactions on Intelligent Systems and Technology, special issue on Large Scale Machine Learning*, 2011.

[10] A. Smola, S. Narayanamurthy. An Architecture for Parallel Topic Models. *Proceedings of the VLDB Endowment, vol. 3 (1-2), pp. 703-710*, 2010.

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

[12] R. Rehurek, P. Sojka. Software Framework for Topic Modelling with Large Corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, pp. 45-50*, 2010.

[13] T. Rubin, A. Chambers, P. Smyth, M. Steyvers. Statistical Topic Models for Multi-label Document Classification. *Machine Learning Journal, vol. 88, no. 1-2, pp. 157-208*, 2012.

A Приложение

В настоящем приложении будет описан процесс создания и включения в BigARTM новых регуляризаторов[9]. При отсылках к файлам предполагается, что пользователь находится в корневой директории библиотеки.

При создании регуляризаторов в BigARTM активно используется технология Google Protocol Buffers[10].

Инструкции будут сопровождаться примерами добавления регуляризаторов. Поскольку в процессе создания регуляризаторов Φ и Θ есть существенные различия, примеров будет два: New Regularizer Phi и New Regularizer Theta.

A.1 Общие шаги при создании любого регуляризатора

- В файле 'src/artm/messages.proto', в сообщении RegularizerConfig найти перечислимое поле Type и добавить в него значение типа, соответствующее регуляризатору. Пример:

```
message RegularizerConfig {
  enum Type {
    SmoothSparseTheta = 0;
    SmoothSparsePhi = 1;
    DecorrelatorPhi = 2;
    MultiLanguagePhi = 3;
    LabelRegularizationPhi = 4;
    NewRegularizerPhi = 5;
    NewRegularizerTheta = 6;
  }
  optional string name = 1;
  optional Type type = 2;
```

[9]Инструкция является актуальной для последней релиз-версии библиотеки на момент защиты данной выпускной квалификационной работы.

[10]http://code.google.com/p/protobuf/

```
  optional bytes config = 3;
}
```

В этом же файле нужно описать proto-сообщение, соответствующее конфигурации нового регуляризатора. Содержимое его зависит от того, какие параметры требуются регуляризатору, как минимум у регуляризатора есть список имён тем, с которыми он должен работать. Иногда есть ещё вид модальности, имя словаря с дополнительными данными (эти параметры свойственны регуляризаторам Φ), а также важный для регуляризаторов Θ параметр alpha_iter, который является массивом дополнительных коэффициентов регуляризации, определяющих степень воздействия регуляризатора на элементы Θ в зависимости от номера итерации прохода по документу. Для New Regularizer Phi и New Regularizer Theta такие конфигурации могут иметь вид:

```
message NewRegularizerPhiConfig {
  repeated string topic_name = 1;
  repeated string class_id = 2;
  optional string dictionary_name = 3;
}

message NewRegularizerThetaConfig {
  repeated string topic_name = 1;
  repeated float alpha_iter = 2;
}
```

- Регуляризаторы являются частью ядра, поэтому реализуются только на C++. Для каждого регуляризатора следует создать .h и .cc файлы, и поместить их в 'src/artm/regularizer/'. Для того, чтобы файлы имели нужный формат, лучше всего скопировать их аналоги для существующих регуляризаторов нужной матрицы и изменить все названия, а после и содержимое. Создадим для регуляризаторов из примера файлы 'new_regularizer_phi.h', 'new_regularizer_phi.cc', 'new_regularizer_theta.h' и 'new_regularizer_theta.cc'.

- Добавить информацию об этих файлах в несколько других. В каждом случае требуется найти в нужном файле имена других регуляризаторов и действовать по аналогии:

  1. В файл 'src/artm/CMakeLists.txt'.
  2. В файл 'utils/cpplint_files.txt'
  3. В файл 'src/artm/core/instance.cc'. Здесь необходимо добавить .h файлы директивой `#include`, а также в методе `CreateOrReconfigureRegularizer()` найти `switch` по типам регуляризаторов и внести туда изменения.

- Сделать регуляризатор доступным из Python API. Для этого открыть 'src/python/artm/library.py' и добавить в находящийся в начале файла список констант ту, что соответствуют добавленному регуляризатору. В описываемом примере это будут

  ```
  RegularizerConfig_Type_NewRegularizerPhi = 5
  RegularizerConfig_Type_NewRegularizerTheta = 6
  ```

  После этого желательно (но не обязательно) создать в этом же файле методы `CreateNewRegularizerPhiRegularizer()` и `CreateNewRegularizerThetaRegularizer()` (в полной аналогии с такими методами для существующих регуляризаторов).

- Описать файл .cc регуляризатора, в котором будет производится его работа, этот процесс опишем ниже подробнее отдельно для регуляризаторов Φ и Θ. На этапе написания кода регуляризатора рекомендуется использовать в качестве примера код какого-либо из существующих регуляризаторов этой матрицы.

A.2 Реализация кода регуляризатора Φ

Регуляризаторы матрицы Φ устроены сравнительно несложно. Если при создании файлов .h и .cc воспользоваться копиями этих файлов для существующего регуляризатора и произвести в них замену всех имён (что предлагалось сделать выше), то единственное, что останется реализовать — это метод `bool NewRegularizerPhi::RegularizePhi(`

`::artm::core::Regularizable* topic_model, double tau)`. `topic_model` — это тематическая модель, объект, содержащий в себе матрицу Φ, а также счётчики n_{wt} и нормировочные константы n_t, из которых она была получена, и списки слов и тем. Всё это, вместе с информацией из словарей (если имена таковых были переданы в качестве параметров регуляризатора в его конфигурации) и другими параметрами (которые тоже должны были быть переданы в конфигурации) используется для реализации регуляризации. `tau` — это коэффициент регуляризации данного регуляризатора. Все изменения заносятся в `topic_model` методом

```
topic_model->IncreaseRegularizerWeight(token_id, topic_id, value);
```

Первые два параметра определяют индекс ячейки, для которой значение изменяется, последний является обновлением, на который значение изменится.

A.3 Реализация кода регуляризатора Θ

Регуляризаторы Θ устроены немного сложнее. Для каждого регуляризатора создаётся объект класса-агента, являющегося наследником класса `RegularizeThetaAgent`. Этот объект при создании получает от регуляризатора все параметры, проверяет их корректность и сохраняет внутри себя в удобном для работы представлении. Далее, для этого агента следует определить метод

```
void Apply(int item_index, int inner_iter, int topics_size, float* theta)
```

Задача этого метода — собственно производить регуляризацию вектора `theta` с учётом номера документа `item_index`, номера прохода по этому документу `inner_iter`, числа тем `topics_size` и всех параметров, которые были сохранены в полях агента при его создании.