

Онлайновые, параллельные и распределённые реализации алгоритмов тематического моделирования

Мурат Апишев
great-mel@yandex.ru

ШАД Яндекс

19 апреля, 2016

1 Введение

- Задача тематического моделирования
- PLSA
- Методика изложения

2 LDA-based реализации

- AD-LDA
- Y!LDA
- Mr.LDA
- Gensim и Vowpal Wabbit LDA

3 BigARTM

- ARTM (продолжение теории) & BigARTM
- Синхронные алгоритмы
- Асинхронные алгоритмы
- Сравнение и результаты

Тематическое моделирование

Тематическое моделирование — приложение машинного обучения к статистическому анализу текстов.

Тема — терминология предметной области, набор терминов (униграм или n -грам) часто встречающихся вместе в документах.

Тематическая модель исследует скрытую тематическую структуру коллекции текстов:

- *тема t* — это вероятностное распределение $p(w|t)$ над терминами w
- *документ d* — это вероятностное распределение $p(t|d)$ над темами t

Приложения

Приложения:

- информационный поиск по длинным текстовым запросам;
- анализ данных социальных медиа (соцсети, блоги);
- мягкая кластеризация, визуализация данных;
- классификация текстов;
- суммаризация текстов.

Проблемы при работе с большими данными:

- слишком медленная обработка данных в однопоточном режиме;
- невозможность потоковой обработки;
- необходимость хранения в памяти больших массивов информации.

Необходимо использовать параллельные, распределённые и онлайн-варианты алгоритмов.

Задача тематического моделирования

Дано: W — словарь терминов (униграм или n -биграмм),
 D — коллекция текстовых документов $d \subset W$,
 n_{dw} — счётчик частоты появления слова w в документе d .

Найти: модель $p(w|d) = \sum_{t \in T} \phi_{wt} \theta_{td}$ с параметрами Φ и Θ :
 $\phi_{wt} = p(w|t)$ — вероятности терминов w в каждой теме t ,
 $\theta_{td} = p(t|d)$ — вероятности тем t в каждом документе d .

Критерий максимизация логарифма правдоподобия:

$$\sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt} \theta_{td} \rightarrow \max_{\phi, \theta};$$

$$\phi_{wt} \geq 0; \quad \sum_w \phi_{wt} = 1; \quad \theta_{td} \geq 0; \quad \sum_t \theta_{td} = 1.$$

PLSA и EM-алгоритм

Максимизация логарифма правдоподобия:

$$\sum_{d \in D} \sum_{w \in W} n_{dw} \ln \sum_t \phi_{wt} \theta_{td} \rightarrow \max_{\Phi, \Theta}$$

EM-алгоритм: метод простых итерация для решения системы уравнений

$$\begin{array}{l} \text{E-шаг:} \\ \text{M-шаг:} \end{array} \left\{ \begin{array}{l} p_{tdw} = \mathop{\text{norm}}_{t \in T}(\phi_{wt} \theta_{td}) \\ \phi_{wt} = \mathop{\text{norm}}_{w \in W}(n_{wt}), \quad n_{wt} = \sum_{d \in D} n_{dw} p_{tdw} \\ \theta_{td} = \mathop{\text{norm}}_{t \in T}(n_{td}), \quad n_{td} = \sum_{w \in W} n_{dw} p_{tdw} \end{array} \right.$$

где $\mathop{\text{norm}}_{i \in I} x_i = \frac{\max\{x_i, 0\}}{\sum_{j \in I} \max\{x_j, 0\}}$

Методика изложения

- Существуют различные постановки задач тематического моделирования (PLSA, LDA, ARTM).
- В рамках каждой постановки существуют различные методы обучения (EM-алгоритм, сэмплирование и т.д.).
- Дальше в лекции будут рассматриваться реализации различных моделей и подходов.

Для единообразия будет использоваться нотация PLSA (и его обобщения — ARTM), технически верно описывающая реализованный алгоритм ¹.

¹ даже если авторы алгоритма работали с байесовской моделью LDA

Самое главное

Базовая операция — обработка одного документа: `ProcessDocument`.

- Принимает на вход текущие счётчики n_{wt} (может и n_{td} , если они хранятся, иначе можно использовать случайные θ_d) и документ d .
- Возвращает инкременты \tilde{n}_{wt} (может и итоговые векторы θ_d , если они вычисляются и нужны)

Внутри могут быть разные реализации: сэмплирование Гиббса, итерации EM-алгоритма.

Любой процесс распараллеливания концептуально состоит из двух этапов:

- 1 Параллельный многократный запуск операций `ProcessDocument`.
- 2 Агрегирование всех инкрементов \tilde{n}_{wt} и их прибавление, возможно, с некоторым весом, к исходным n_{wt} .

Процесс повторяется итеративно до сходимости.

Метрика качества

Вопрос: сходится до сходимости **чего?**

Методов оценивания качества тематического моделирования много, они разнообразны и специфичны для разных задач.

Одна из универсальных величин, характеризующих степень сходимости модели с заданным словарём W — перплексия:

$$\mathcal{P}(D) = \exp\left(-\frac{1}{n} \sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt} \theta_{td}\right), \quad n = \sum_d n_d.$$

Она построена на основе логарифма правдоподобия и характеризует степень качества описания коллекции моделью. Чем ниже — тем лучше.

AD-LDA

- Алгоритм Approximate Distributed LDA (AD-LDA) был предложен в «D. Newman, A. Asuncion, P. Smyth, and M. Welling — Distributed algorithms for topic models».
- Основан на коллапсированной схеме Гиббса.
- Распараллеливается по ядрам, т.е. даже в рамках машины используется не многопоточная, а многопроцессорная архитектура.

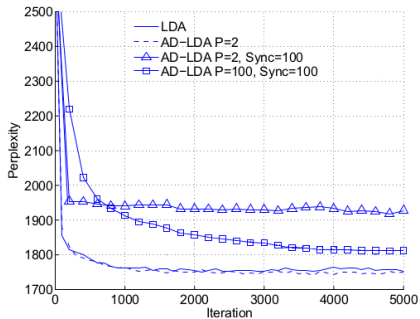
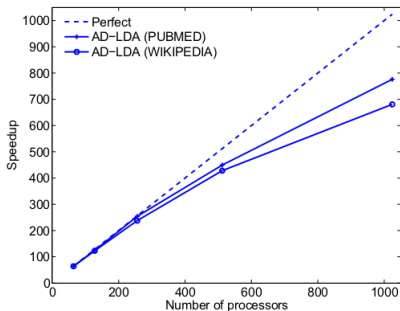
Описание:

- Коллекция D распределяется по P . Документы распределяются случайным образом, без предварительной кластеризации.
- Информация о словах каждого документа и счётчики n_{td} также распределены (обозначим последние n_{tdp}).
- Каждый процессор имеет свою локальную *полную* копию n_{wtp} глобальных счётчиков n_{wt} .

Алгоритм

- 1 На потоки (сэмплеры) загружается по частям коллекция и копируются счётчики n_{wt} .
- 2 Каждый сэмплер производит обработку своих данных, вызывая `ProcessDocument` для каждого документа. Аккумулируются инкременты \tilde{n}_{wtp} .
- 3 Обновляются локальные счётчики n_{tdp} .
- 4 Общий шаг синхронизации для обновления глобальных n_{wt} .
- 5 Новая n_{wt} копируется на процессоры и начинается следующая итерация обработки.

Эксперименты



Выводы

AD-LDA имеет ряд серьёзных недостатков:

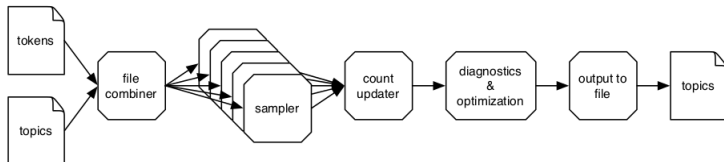
- Необходимость частых синхронизаций.
- Из-за синхронизации скорость определяется самым медленным процессором.
- Во время итераций сеть простаивает, а во время синхронизаций — перегружена.
- Большое потребление памяти — копия глобальных счётчиков n_{wt} хранится на каждом ядре.

Y!LDA

- Y!LDA был предложен в «A. Smola and S. Narayanamurthy — An architecture for parallel topic models,».
- Он так же основан на коллапсированной схеме Гиббса.
- Производит двухуровневую обработку:
 - 1 многопоточную в рамках одного нода;
 - 2 многопроцессорную в рамках кластера, в соответствии с т.н. *архитектурой классной доски*.

Схема многопоточного параллелизма

- На каждом узле создаётся несколько потоков-сэмплеров.
- И один поток, задача которого — сливать полученные от сэмплеров обновления \tilde{n}_{wt} в глобальную n_{wt} .
- Глобальное (в рамках узла) состояние n_{wtp} является общим для всех ядер узла.



Описание вычислений на кластере

Проблема: синхронизация глобальной (в рамках кластера) матрицы n_{wt} между всеми нодами-обработчиками.

Архитектура классной доски:

- глобальная матрица счётчиков n_{wt} хранится в единственном экземпляре;
- она является общей для всех нодов;
- её обновление производится сэмплерами асинхронно по одному слову w за один раз.

Схема обновлений n_{wt}

-
-
- 1 Инициализировать $n_{wt} = n_{wtp} = n_{wtp}^{old}$, для всех узлов p ;
 - 2 **repeat**
 - 3 Заблокировать глобально n_{wt} для некоторого слова;
 - 4 Заблокировать локально n_{wtp} для данного слова;
 - 5 Обновить глобальное состояние: $n_{wt} := n_{wt} + (n_{wtp} - n_{wtp}^{old})$;
 - 6 Обновить локальное состояние: $n_{wtp}^{old} = n_{wtp} = n_{wt}$;
 - 7 Разблокировать n_{wtp} ;
 - 8 Разблокировать n_{wt} ;
 - 9 **until** производится сэмплирование;
-

- n_{wt} — глобальное состояние.
- n_{wtp} — текущее локальное состояние.
- n_{wtp}^{old} — копия локального состояния на момент последней синхронизации с n_{wt} .

Технические детали реализации

- Все данные внутри программы реализуются и передаются с помощью технологии Google protocol buffers. Она позволяет описывать структуры данных на псевдо-языке, компилировать его в код на C++/Python/Java, сериализовывать/десериализовывать эти структуры.
- Для хранения глобального n_{wt} используется memcached — сервис, реализующий в оперативной памяти хранилище на основе хеш-таблицы.
- Алгоритм реализован на кластере рабочих станций с сервером и на Hadoop-кластере с машинами аналогичной мощности.

Достоинства Y!LDA

Таким образом, Y!LDA решает описанные выше проблемы AD-LDA:

- Отсутствие выделенного шага синхронизации позволяет более быстрым сэмплерам не ждать медленных.
- Сеть равномерно загружена всё время работы.
- Количество памяти, используемой для хранения копий глобальных счётчиков n_{wt} определяется не числом ядер в кластере, а числом узлов.

Но одна проблема есть:

выделенный поток (потoki) слияния могут стать узким местом алгоритма, и в рамках нода, и в рамках кластера.

Mr. LDA

- Реализация Mr. LDA описана в «K. Zhai, J. Boyd-Graber, N. Asadi, M. Alkhouja — Mr. LDA: A Flexible Large Scale Topic Modeling Package using Variational Inference in MapReduce».
- Алгоритм основан на вариационном выводе (вариационный EM-алгоритм).
- Обработка производится в рамках парадигмы MapReduce и реализована на Hadoop.

Описание схемы MapReduce

Описание этого алгоритма сделано в качестве примера обучения тематической модели с использованием MapReduce, оно не соответствует в деталях содержанию оригинальной статьи.

- На каждый документ коллекции создаётся mapper. Он производит вызов `ProcessDocument`.
- На каждую тему создается reducer. Он занимается слиянием полученных инкрементов n_{wt} .
- Третий компонент — `driver` — присутствует в системе в единственном экземпляре, управляет всем процессом обучения и вычисляет перплексию.
- Глобальные параметры n_{wt} хранятся в специальной, доступной только для чтения и общей для всех mapper-ов памяти, называемой *распределённым кэшем*.

Оффлайновые и онлайнные EM-алгоритмы

Все дальнейшие реализации основаны на EM-алгоритме.

Оффлайновый EM-алгоритм:

- 1 Многократное итерирование по коллекции.
- 2 Однократная обработка документа.
- 3 $\Phi (n_{wt})$ обновляется в конце каждого прохода по коллекции.
- 4 Применяется при обработке небольших коллекций.

Онлайнный EM-алгоритм:

- 1 Однократный проход по коллекции.
- 2 Многократная обработка одного документа.
- 3 $\Phi (n_{wt})$ обновляется через определённое число обработанных документов.
- 4 Применяется при обработке больших коллекций в потоковом режиме.

Online LDA: VW.LDA

Алгоритм Online LDA был предложен в «M. D. Hoffman, D. M. Blei, F. Bach — Online Learning for Latent Dirichlet Allocation», в его основе лежит вариационный EM-алгоритм.

Vowpal Wabbit LDA:

- Онлайновый.
- Не параллельный.
- Реализован на C++ без STL и сторонних библиотек.
- Относительно эффективный инструмент для моделирования больших коллекций в потоковом режиме.
- Может использоваться на кластере.

Online LDA: Gensim

- Онлайновый.
- Параллельный (однопоточная реализация LdaModel, многопоточная — LdaMulticore). Архитектурно похож на Y!LDA. Многопоточный параллелизм реализован не очень эффективно, плохая масштабируемость.
- Распределённый (неэффективная реализация).
- Реализован на Python.
- Удобный инструмент для моделирования небольших коллекций.

ARTM: ещё раз о задаче тематического моделирования

Дано: W — словарь терминов (униграм или n -биграмм),
 D — коллекция текстовых документов $d \subset W$,
 n_{dw} — счётчик частоты появления слова w в документе d .

Найти: модель $p(w|d) = \sum_{t \in T} \phi_{wt} \theta_{td}$ с параметрами Φ и Θ :
 Φ $W \times T$ и Θ $T \times D$

$\phi_{wt} = p(w|t)$ — вероятности терминов w в каждой теме t ,
 $\theta_{td} = p(t|d)$ — вероятности тем t в каждом документе d .

Критерий максимизация логарифма правдоподобия:

$$\sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt} \theta_{td} \rightarrow \max_{\phi, \theta};$$

$$\phi_{wt} \geq 0; \quad \sum_w \phi_{wt} = 1; \quad \theta_{td} \geq 0; \quad \sum_t \theta_{td} = 1.$$

Проблема: задача стохастического матричного разложения
 некорректно поставленная: $\Phi \Theta = (\Phi S)(S^{-1} \Theta) = \Phi' \Theta'$.

ARTM и регуляризованный EM-алгоритм

Максимизация логарифма правдоподобия с **дополнительными аддитивными регуляризаторами R** :

$$\sum_{d \in D} \sum_{w \in W} n_{dw} \ln \sum_t \phi_{wt} \theta_{td} + R(\Phi, \Theta) \rightarrow \max_{\Phi, \Theta}$$

EM-алгоритм: метод простых итераций для системы уравнений

$$\begin{cases} \text{E-шаг:} & p_{tdw} = \mathop{\text{norm}}_{t \in T}(\phi_{wt} \theta_{td}) \\ \text{M-шаг:} & \begin{cases} \phi_{wt} = \mathop{\text{norm}}_{w \in W} \left(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}} \right), & n_{wt} = \sum_{d \in D} n_{dw} p_{tdw} \\ \theta_{td} = \mathop{\text{norm}}_{t \in T} \left(n_{td} + \theta_{td} \frac{\partial R}{\partial \theta_{td}} \right), & n_{td} = \sum_{w \in W} n_{dw} p_{tdw} \end{cases} \end{cases}$$

Примеры регуляризаторов

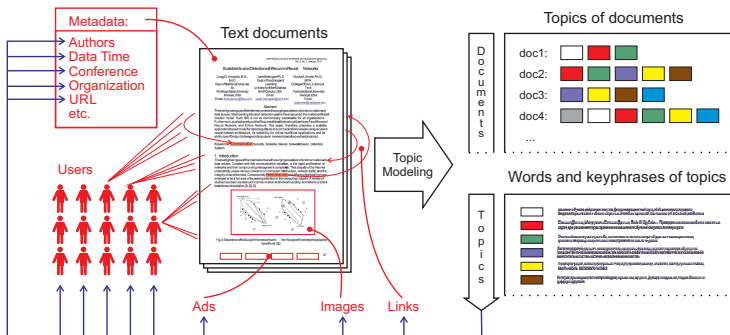
Многие байесовские модели могут быть интерпретированы в терминах ARTM.

Примеры регуляризаторов:

- 1 Сглаживание Φ / Θ (приводит к известной модели LDA)
- 2 Разреживание Φ / Θ
- 3 Декорреляция тем в Φ
- 4 Частичное обучение
- 5 Максимизация когерентности тем
- 6 Отбор тем
- 7 ...

Мультимодальная тематическая модель

Мультимодальная тематическая модель распределения тем на терминах $p(w|t)$, авторах $p(a|t)$, метках времени $p(y|t)$, изображениях $p(o|t)$, связанных документвх $p(d'|t)$, рекламных баннерах $p(b|t)$, пользователей $p(u|t)$, **и объединяет все эти модальности в одно тематическую модель.**



M-ARTM и мультимодальный регуляризованный EM-алгоритм

W^m — словарь терминов m -й модальности, $m \in M$,
 $W = W^1 \sqcup W^m$ как объединение словарей всех модальностей.

Максимизация логарифма **мультимодального** правдоподобия с аддитивными регуляризаторами R :

$$\sum_{m \in M} \lambda_m \sum_{d \in D} \sum_{w \in W^m} n_{dw} \ln \sum_t \phi_{wt} \theta_{td} + R(\Phi, \Theta) \rightarrow \max_{\Phi, \Theta}$$

EM-алгоритм: метод простых итерация для системы уравнений

$$\begin{cases} \text{E-шаг:} & \left\{ \begin{array}{l} p_{tdw} = \mathop{\text{norm}}_{t \in T} (\phi_{wt} \theta_{td}) \\ \phi_{wt} = \mathop{\text{norm}}_{w \in W^m} \left(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}} \right), \quad n_{wt} = \sum_{d \in D} \lambda_{m(w)} n_{dw} p_{tdw} \\ \theta_{td} = \mathop{\text{norm}}_{t \in T} \left(n_{td} + \theta_{td} \frac{\partial R}{\partial \theta_{td}} \right), \quad n_{td} = \sum_{w \in d} \lambda_{m(w)} n_{dw} p_{tdw} \end{array} \right. \end{cases}$$

Проект BigARTM

Особенности BigARTM:

- Быстрая² параллельная и онлайн-обработка данных;
- Поддержка мультимодальных регуляризованных тематических моделей;
- Встроенная расширяемая библиотека регуляризаторов и метрик качества;

Сообщество BigARTM:

- Открытый репозиторий <https://github.com/bigartm>
- Описание и документация <http://bigartm.org>

Лицензия BigARTM и программные особенности:

- Бесплатное коммерческое использование (BSD 3-Clause license)
- Кроссплатформенная — Windows, Linux, Mac OS X (32 bit, 64 bit)
- Программные API: command line, C++, Python

²Vorontsov K., Frei O., Apishev M., Romov P., Dudarenko M. BigARTM: Open Source Library for Regularized Multimodal Topic Modeling of Large Collections Analysis of Images, Social Networks and Texts. 2015

Операция ProcessDocument

Input: документ $d \in D$, матрица $\Phi = (\phi_{wt})$;

Output: матрица (\tilde{n}_{wt}) , вектор θ_{td} ;

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

2 **repeat**

3 $p_{tdw} := \mathop{\text{norm}}_{t \in T}(\phi_{wt}\theta_{td})$ для всех $w \in d$ и $t \in T$;

4 $\theta_{td} := \mathop{\text{norm}}_{t \in T}(\sum_{w \in d} n_{dw}p_{tdw} + \theta_{td} \frac{\partial R}{\partial \theta_{td}})$ для всех $t \in T$;

5 **until** до сходимости θ_d ;

6 $\tilde{n}_{wt} := n_{dw}p_{tdw}$ для всех $w \in d$ и $t \in T$;

Оффлайнный алгоритм: описание

Коллекция документов D разбивается на пакеты, называемые *батчами*.

Оффлайнный алгоритм производит сканирование коллекции, вызывая `ProcessDocument` для каждого документа $d \in D$ в коллекции.

Затем он агрегирует результирующие матрицы (\tilde{n}_{wt}) в финальную матрицу (n_{wt}) размера $|W| \times |T|$.

После каждого прохода по коллекции алгоритм пересчитывает матрицу Φ по формуле

$$\phi_{wt} = \operatorname{norm}_{w \in W} \left(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}} \right), \quad n_{wt} = \sum_{d \in D} n_{dw} p_{tdw}$$

Оффлайновый алгоритм: листинг

Input: коллекция D ;

Output: матрица $\Phi = (\phi_{wt})$;

1 инициализировать (ϕ_{wt}) ;

2 создать батчи $D := D_1 \sqcup D_2 \sqcup \dots \sqcup D_B$;

3 **repeat**

4 $(n_{wt}) := \sum_{b=1, \dots, B} \sum_{d \in D_b} \text{ProcessDocument}(d, \Phi)$;

5 $(\phi_{wt}) := \text{norm}_{w \in W} (n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}})$;

6 **until** до сходимости (ϕ_{wt}) ;

Оффлайновый алгоритм: обсуждение

Внешний цикл распараллеливается по потокам.

Внутри каждого пакета внутренний цикл по документам $d \in D_b$ производится в однопоточном режиме.

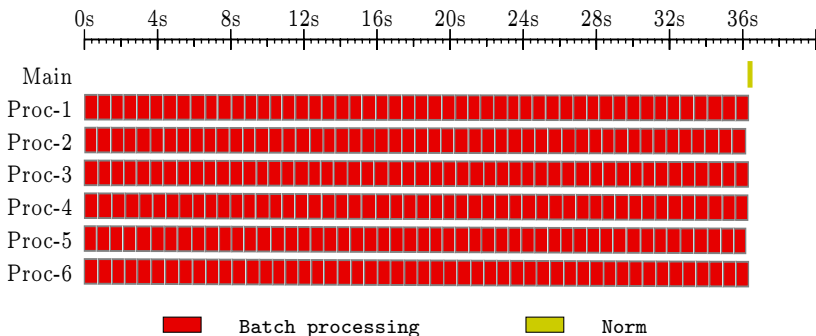
Значения θ_{td} появляются только в функции ProcessDocument
 \Rightarrow

Эффективное использование памяти: реализация никогда не хранит в памяти целиком всю матрицу Θ .

Вместо этого значения θ_{td} пересчитываются при каждом проходе по коллекции.

Операция ProcessDocument может быть полезна как отдельная функция для поиска распределений θ_{td} новых документов, не участвовавших в обучении.

Оффлайновый алгоритм: диаграмма Гантта



- Этот и последующие диаграммы Гантта создавались с помощью коллекции NYTimes: <https://archive.ics.uci.edu/ml/datasets/Bag+of+Words>
- Размер коллекции $\approx 300k$ документов, но алгоритмы запускались на подмножествах размером от 70% до 100% для достижения единого времени работы (≈ 36 сек.)

Онлайновый алгоритм: описание

Алгоритм является обобщением алгоритма Online variational Bayes для модели LDA (на нём основаны Gensim и VW.LDA).

Онлайновый ARTM улучшает скорость сходимости оффлайн-алгоритма путём пересчёта матрицы Φ после каждых η батчей.

Введём элементарную операцию для упрощения нотации:

$$\text{ProcessBatches}(\{D_b\}, \Phi) = \sum_{D_b} \sum_{d \in D_b} \text{ProcessDocument}(d, \Phi)$$

Онлайновый алгоритм: листинг

Input: коллекция D , параметры η, τ_0, κ ;

Output: матрица $\Phi = (\phi_{wt})$;

- 1 создать батчи $D := D_1 \sqcup D_2 \sqcup \dots \sqcup D_B$;
- 2 инициализация (ϕ_{wt}^0) ;
- 3 **for all** обновить $i = 1, \dots, \lfloor B/\eta \rfloor$
- 4 $(\tilde{n}_{wt}^i) := \text{ProcessBatches}(\{D_{\eta(i-1)+1}, \dots, D_{\eta i}\}, \Phi^{i-1})$;
- 5 $\rho_i := (\tau_0 + i)^{-\kappa}$;
- 6 $(n_{wt}^i) := (1 - \rho_i) \cdot (n_{wt}^{i-1}) + \rho_i \cdot (\tilde{n}_{wt}^i)$;
- 7 $(\phi_{wt}^i) := \text{norm}_{w \in W}(n_{wt}^i + \phi_{wt}^{i-1} \frac{\partial R}{\partial \phi_{wt}})$;

Онлайновый алгоритм: обсуждение

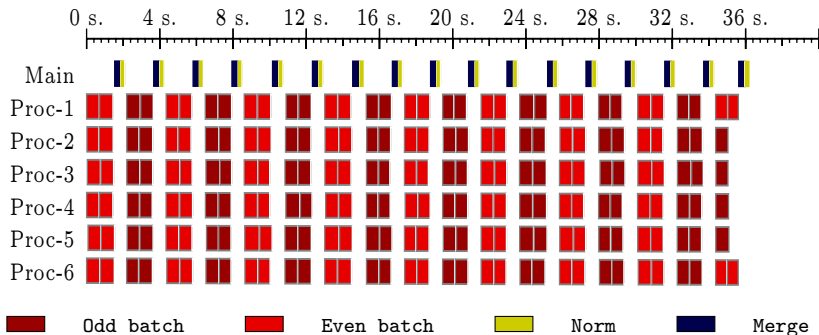
Разбиение коллекции на батчи имеет важное значение — оно сказывается на итоговой модели.

Весовая формула для новых счётчиков $\rho_i := (\tau_0 + i)^{-\kappa}$ была взята из Online LDA, это одна из возможных эвристик, не самая универсальная.

Типичные значения τ_0 лежат между 64 и 1024, κ — между 0.5 и 0.7.

Проблема: все рабочие потоки проставивают во время операций агрегации и нормализации, поскольку новая версия матрицы Φ , которая нужна для вывода, ещё не готова.

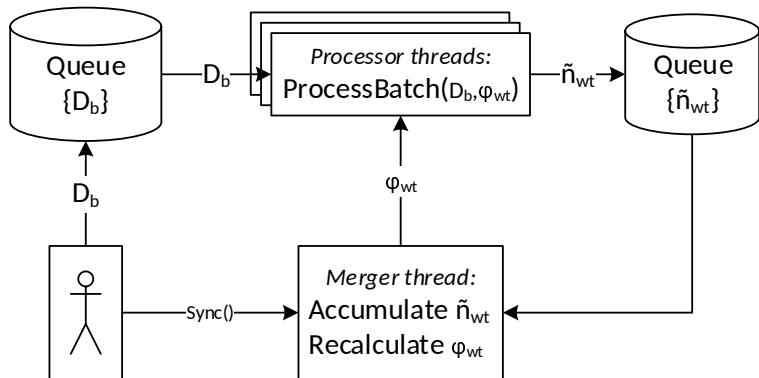
Онлайновый алгоритм: диаграмма Ганта



Асинхронный онлайнный алгоритм (Async): описание

- Есть поток `DataLoader`, который загружает батчи с диска в очередь обработки `Processor queue`.
- Каждый поток обработчик `Processor` извлекает из очереди по одному батчу и производит на нём итерации EM-алгоритма.
- После того, как `Processor` окончил обработку батча, он помещает инкременты $\tilde{\eta}_{wt}$ в очередь слияния `Merger queue` (если в ней есть место) и начинает обработку следующего батча.
- Когда число обновлений в очереди `Merger queue` становится равным параметру η , выделенный поток `Merger` производит обновление матрицы Φ .

Алгоритм Async: схема



Алгоритм Async: достоинства

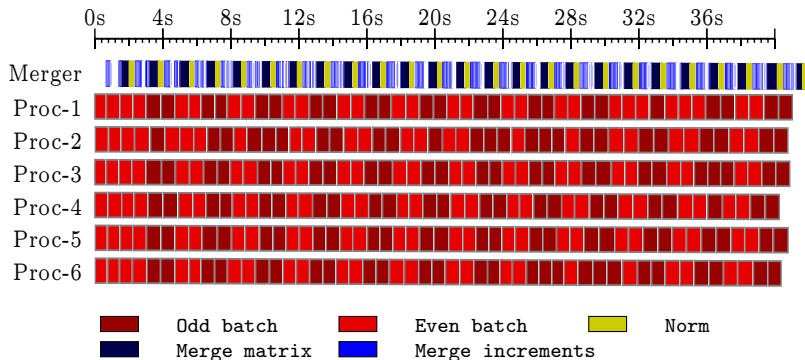
В нормальной ситуации Async совмещает все полезные качества предыдущих алгоритмов (т.е. хорошую сходимость и высокую степень загрузки потоков).

- 3.7M статей англ. Википедии, $|W|=100k$

| Фреймворк | procs | обучение | вывод | перплексия |
|----------------|-------|----------------|---------------|------------|
| BigARTM | 1 | 35 min | 72 sec | 4000 |
| LdaModel | 1 | 369 min | 395 sec | 4161 |
| VW.LDA | 1 | 73 min | 120 sec | 4108 |
| BigARTM | 4 | 9 min | 20 sec | 4061 |
| LdaMulticore | 4 | 60 min | 222 sec | 4111 |
| BigARTM | 8 | 4.5 min | 14 sec | 4304 |
| LdaMulticore | 8 | 57 min | 224 sec | 4455 |

- $procs$ = число параллельных потоков
- $вывод$ = время подсчёта распределений θ_d для отложенной выборки в 100k документов

Алгоритм Аsync: диаграмма Гантта в нормальной ситуации



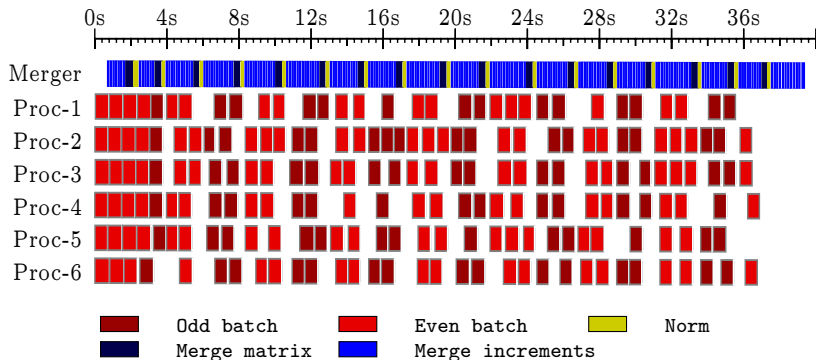
Алгоритм Async: недостатки

- Алгоритм Async не определяет порядок слияния $\tilde{n}_{wt} \Rightarrow$ результирующая матрица Φ различна от запуска к запуску.
- Помещение инкрементов \tilde{n}_{wt} в очередь может серьёзно увеличить потребление памяти \Rightarrow что приводит к перегрузке потока слияния *Merger*.

Особенно поток слияния могут перегрузить маленькие батчи или малое число внутренних итераций в `ProcessDocument`.

- Это означает, что пользователь должен настраивать технические параметры, что недопустимо.

Алгоритм Аsync: диаграмма Гантта в плохой ситуации



Детерминированный асинхронный онлайнный алгоритм (DetAsync): описание

Чтобы избежать недетерминированного поведения, потребуем обновлений не после *первых* η батчей, а после *заданных* η батчей.

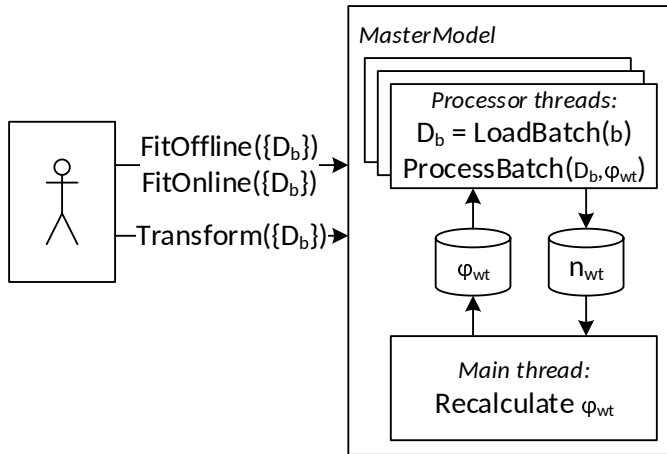
Введём две новые операции: `AsyncProcessBatches` и `Await`.

`AsyncProcessBatches` эквивалентна `ProcessBatches`, кроме того, что она берёт задачу на выполнение и сразу возвращает управление в вызвавший поток.

Она выдаёт объект типа `future` (примером является `std::future` из стандарта C++11), который может быть затем подан в операцию `Await` для получения результатов, т.е. инкрементов \tilde{n}_{wt} .

Между вызовами `AsyncProcessBatches` и `Await` алгоритм может выполнять полезную работу по обновлению Φ , пока в фоновом режиме потоки `Processor` вычисляют матрицу \tilde{n}_{wt} .

DetAsync: схема



DetAsync: листинг

Input: коллекция D , параметры η, τ_0, κ ;

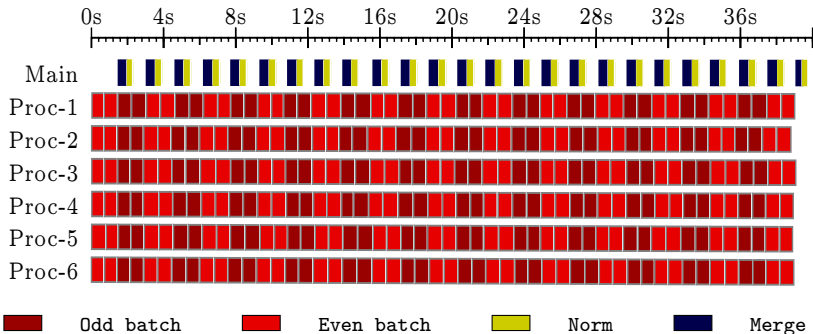
Output: матрица $\Phi = (\phi_{wt})$;

```

1 создать батчи  $D := D_1 \sqcup D_2 \sqcup \dots \sqcup D_B$ ;
2 инициализировать  $(\phi_{wt}^0)$ ;
3  $F^1 := \text{AsyncProcessBatches}(\{D_1, \dots, D_\eta\}, \Phi^0)$ ;
4 for all обновления  $i = 1, \dots, \lfloor B/\eta \rfloor$ 
5     if  $i \neq \lfloor B/\eta \rfloor$  then
6          $F^{i+1} := \text{AsyncProcessBatches}(\{D_{\eta i+1}, \dots, D_{\eta i+\eta}\}, \Phi^{i-1})$ ;
7          $(\hat{n}_{wt}^i) := \text{Await}(F^i)$ ;
8          $\rho_i := (\tau_0 + i)^{-\kappa}$ ;
9          $(n_{wt}^i) := (1 - \rho_i) \cdot (n_{wt}^{i-1}) + \rho_i \cdot (\hat{n}_{wt}^i)$ ;
10         $(\phi_{wt}^i) := \text{norm}_{w \in W}(n_{wt}^i + \phi_{wt}^{i-1} \frac{\partial R}{\partial \phi_{wt}})$ ;

```


DetAsync: диаграмма Гантта



DetAsync: детали реализации

- В старой архитектуре матрицы \tilde{n}_{wt} , построенные по разным батчам, хранились в очереди и агрегировались потоком слияния *Merger*.
- В новой архитектуре поток *Merger* был удалён, потоки-обработчики пишут обновления напрямую в n_{wt} (локальный аналог «архитектуры классной доски»).
- *Spin lock*-и предотвращают одновременное обновление одной строки несколькими потоками.
- Тройка операций «lock-update-release» производится в цикле по всем словам документа $w \in d$ в конце операции *ProcessDocument*.
- Выделенный поток загрузки данных *DataLoader* был так же ликвидирован, потоки-обработчики сами загружают себе батчи на обработку с диска.

Все структуры данных, передаваемые внутри библиотеки, сгенерированы технологией Google protocol buffers.

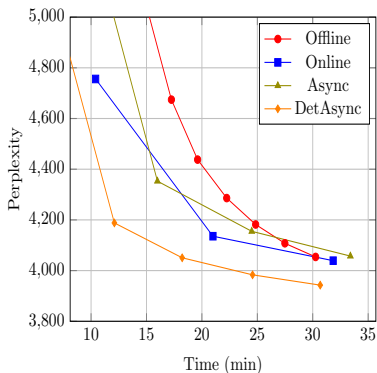
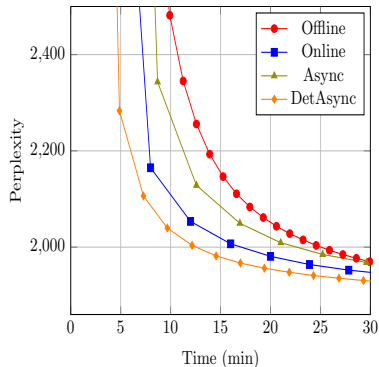
Эксперименты

- Коллекции: *Википедия* ($|D| = 3.7\text{M}$ статей, $|W| = 100\text{K}$ слов), *Pubmed* ($|D| = 8.2\text{M}$ аннотаций, $|W| = 141\text{K}$ слов).
- Машина: Intel Xeon CPU E5-2650 v2 system с 2 процессорами, 16 физическими ядрами в совокупности (32 с гипер-тредингом).
- Метрика: значение перплексии \mathcal{P} достигнутое за выделенное время.
- Время: каждому алгоритму было выделено на работу 30 минут.

Пиковое потребление оперативной памяти (Gb):

| | $ T $ | Оффлайн | Онлайн | DetAsync | Async (v0.6) |
|--------|-------|---------|--------|----------|--------------|
| Pubmed | 1000 | 5.17 | 4.68 | 8.18 | 13.4 |
| Pubmed | 100 | 1.86 | 1.62 | 2.17 | 3.71 |
| Вики | 1000 | 1.74 | 2.44 | 3.93 | 7.9 |
| Вики | 100 | 0.54 | 0.53 | 0.83 | 1.28 |

Достигнутое значение перплексии








Википедия (слева), Pubmed (справа).

Алгоритм DetAsync достиг наилучшего значения перплексии за выделенное время.


Выводы

- Как и в любой задаче машинного обучения, при распараллеливании методов обучения тематической модели важны объёмы потребляемой памяти, степень загрузки вычислительных ресурсов, скорость сходимости. Выигрыш в одном ведёт к потерям в другом, но найти компромисс можно и нужно.
- Алгоритмы AD-LDA, Y!LDA и Mr.LDA представляют, прежде всего, теоретический интерес, как различные подходы к параллельному и распределённому обучению моделей.
- Библиотеки BigARTM, Vowpal Wabbit LDA и Gensim — реально используемые инструменты для тематического моделирования на одной машине (в случае VW.LDA — ещё и на кластере).
- При использовании на одном узле наиболее эффективным является алгоритм DetAsync, реализованный в BigARTM. Кроме этого, библиотека полезна поддержкой мультимодальных моделей ARTM, готовым набором регуляризаторов и метрик качества.

Список литературы I

-  *D. Newman, A. Asuncion, P. Smyth, and M. Welling.* (2009). Distributed algorithms for topic models, // NIPS.
-  *Alexander Smola and Shравan Narayanamurthy.* (2010). An architecture for parallel topic models, // VLDB.
-  *Ke Zhai, Jordan Boyd-Graber, Nima Asadi, Mohamad Alkhouja.* (2012). Mr. LDA: A Flexible Large Scale Topic Modeling Package using Variational Inference in MapReduce, // ACM.
-  *T. Hofmann.* Probabilistic latent semantic indexing, // Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. — New York, NY, USA: ACM, 1999. — Pp. 50–57.
-  *D. M. Blei, A. Ng, and M. Jordan.* (2003). Latent Dirichlet allocation, // Journal of Machine Learning Research, vol. 3, pp. 993–1022.

Список литературы II

-  *Воронцов К. В.*. Аддитивная регуляризация тематических моделей коллекций текстовых документов, // Доклады РАН. 2014. — Т. 455., No3. 268–271.
-  *Vorontsov K. V., Potapenko A. A.* (2014). Additive Regularization of Topic Models, // Machine Learning Journal. Special Issue «Data Analysis and Intelligent Optimization with Applications».
-  *Matthew D. Hoffman, David M. Blei, Francis Bach.* (2010). Online Learning for Latent Dirichlet Allocation, // NIPS.
-  *K. Vorontsov, O. Frei, M. Apishev., P. Romov, M. Suvorova, A. Yanina.* (2015). BigARTM: Non-Bayesian Additive Regularization for Multimodal Topic Modeling of Large Collections // Topic Models: Post-Processing and Applications, CIKM 2015 Workshop, October 19, 2015, Melbourne, Australia. ACM, New York, NY, USA. pp. 29–37.
-  *Frei Olexander, Apishev Murat* (2016). Parallel Non-blocking Deterministic Algorithm for Online Topic Modeling, // AIST (to appear).