Параллельные и распределённые реализации алгоритмов тематического моделирования

Мурат Апишев
great-mel@yandex.ru

МФТИ (ГУ)

22 октября 2015

Тематическое моделирование

Тематическое моделирование — одно из приложений машинного обучения к анализу текстов.

Дано:

- коллекция документов D;
- словарь уникальных слов W;
- число тем $|T|$.

Найти:

- тематическую структуру каждого документа d из текстовой коллекции D на множестве тем T;
- тематический профиль всех слов W этой коллекции.

Тематическое моделирование

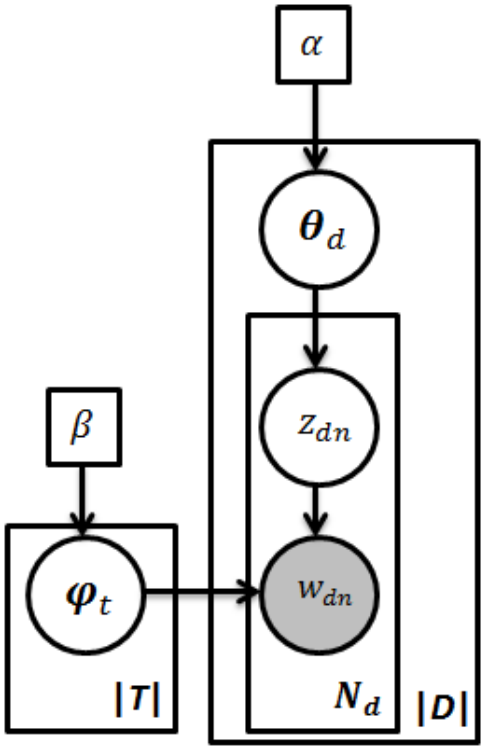
В тематическом моделировании активно используется *гипотеза мешка слов*: не важно, как слова расположены в документе, важно — сколько раз они встретились.

Входная коллекция представима в виде матрицы $N_{wd}|W|\times|D|$, строка соответствует слову из словаря, столбец — документу. На пересечении строки w и столбца d находится частота встречаемости этого слова в этом документе.

Нормируем эту матрицу по столбцам. Задача тематического моделирования — в поиске разложения этой матрицы в произведение двух, $\Phi_{|W|\times|T|}$ и $\Theta_{|T|\times|D|}$. Первая описывает темы словами, вторая — документы темами.

Обозначения и модель LDA

w — номер слова в словаре
t — номер темы
d — номер документа
N_d — число слов в документе
$\boldsymbol{w}_d$ — вектор слов в документе(для всех документов — $\mathcal{W}$)
$\boldsymbol{z}_d$ — вектор тем слов в документе (для всех документов — $\mathcal{Z}$)
Θ — вероятности тем в документах
Φ — вероятности слов в темах
N_{wt} — счётчик «слово-тема»
N_{td} — счётчик «тема-документ»



Модель LDA

Вероятностная модель *латентного размещения Дирихле* (LDA) [**?**] имеет вид

$$p(\mathcal{W}, \mathcal{Z}, \Theta, \Phi | \alpha, \beta) = \prod_{t=1}^{|T|} p(\boldsymbol{\phi}_t | \beta) \prod_{d=1}^{|D|} p(\boldsymbol{\theta}_d | \alpha) \prod_{n=1}^{N_d} p(w_{d,n} | z_{d,n}, \Phi) p(z_{d,n} | \boldsymbol{\theta}_d),$$

$$p(\boldsymbol{\phi}_t | \beta) = \mathrm{Dir}(\boldsymbol{\phi}_t | \beta), \quad p(\boldsymbol{\theta}_d | \alpha) = \mathrm{Dir}(\boldsymbol{\theta}_d | \alpha), \tag{1}$$

$$p(w_{d,n} | z_{d,n}, \Phi) = \Phi_{z_{d,n}, w_{d,n}}, \quad p(z_{d,n} | \boldsymbol{\theta}_d) = \Theta_{d, z_{d,n}}. \tag{2}$$

Фактически, это способ регуляризации задачи матричного разложения путём сглаживания столбцов Φ и Θ путём введения априорных распределений Дирихле.

Коллапсированая схема Гиббса:

1. Сэмплируем $z_{d,n}$

$$p(z_{d,n} = t | \mathcal{Z}^{\setminus(d,n)}, \mathcal{W}, \alpha, \beta) \propto \frac{N_{wt}^{\setminus(d,n)} + \beta}{\sum_w N_{wt}^{\setminus(d,n)} + |W|\beta} (N_{td}^{\setminus(d,n)} + \alpha). \quad (3)$$

2. Обновляем $N_{wt}^{new} = N_{wt}^{old} + \sum_d \sum_n [z_{d,n} = t, w_{d,n} = w]$.
3. Оцениваем Φ и Θ на основе матриц N_{wt} и N_{td}.

Методы обучения LDA

Вариационный вывод (EM-алгоритм):

1. В вариационном выводе рассчитывается приближённое распределение

$$q(\mathcal{Z}, \Theta, \Phi) = \prod_{t=1}^{|T|} \mathrm{Dir}(\boldsymbol{\phi}_t | \boldsymbol{\lambda}_t) \prod_{d=1}^{|D|} \mathrm{Dir}(\boldsymbol{\theta}_d | \boldsymbol{\gamma}_d) \prod_{n=1}^{N_d} \mathrm{Mult}(z_{d,n} | \boldsymbol{\mu}_{d,n}). \tag{4}$$

2. Организуем итерационный процесс:
   - **E-шаг:** Оценивание внутренних параметров факторизованных распределений $\boldsymbol{\lambda}, \boldsymbol{\gamma}, \boldsymbol{\mu}$.
   - **M-шаг:** Пересчет гиперпараметров распределений Дирихле α и β.
3. Оцениваем Φ и Θ на основе внутренних параметров.

Где возникают большие данные?

Везде. Тематическое моделирование используется (или может быть использовано) при:

- анализе социальных сетей;
- построении рекомендательных систем;
- тематизации электронных библиотек и документаций;
- тергетировании рекламы.

Во всех этих случаях приходится иметь дело с большими данными.

При этом данные могут быть статичными, а могут идти непрерывным потоком.

Моделирование больших данных

При использовании стандартных алгоритмов тематического моделирования на больших данных возникают следующие проблемы:

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

Всё это приводит к необходимости использования параллельных, распределённых и онлайновых вариантов алгоритмов.

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.

Сэмплирование в коллапсированной схеме Гиббса — сугубо последовательный процесс, его распаралливание нарушает теоретические свойства марковкой цепи и приводит к генерации выборки не из того распределения.

Описание

Но если слов в документах много, а процессоров-сэмплеров мало, то требование последовательности можно ослабить.
В AD-LDA

- Коллекция D распределяется по P. Документы распределяются случайным образом, без предварительной кластеризации.
- Слововхождения $\mathcal{W}$ и соответствующие им $\mathcal{Z}$ разделяются на P частей и распределяются по процессорам в соответствии со своим d.
- Счетчики N_{td} также распределены (обозначим N_{tdp}). Аналогично каждый процессор имеет свою локальную копию N_{wtp} глобальных счётчиков N_{wt}.

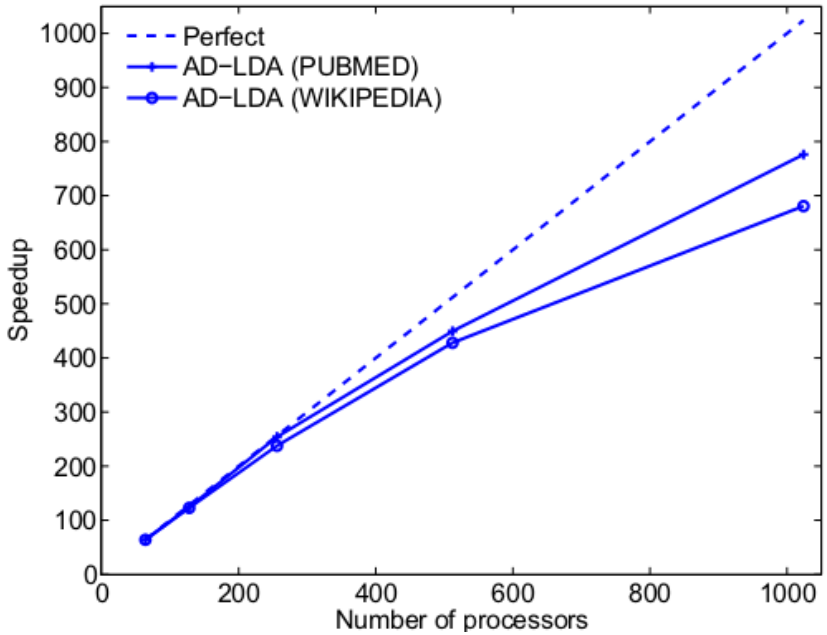
Псевдокод

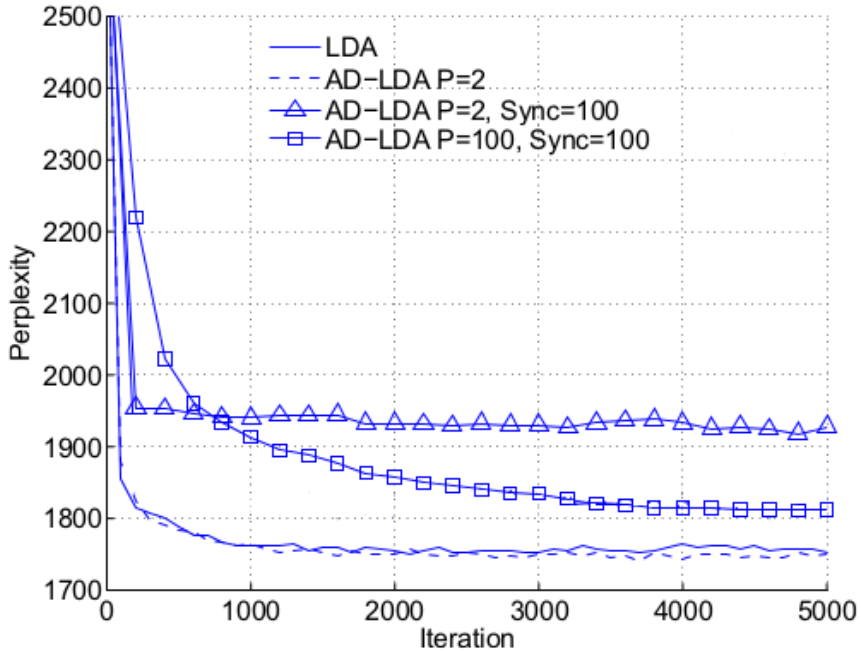
AD-LDA

1 **повторять**
2 | **для каждого** *процессора p одновременно* **выполнять**
3 | | Скопировать глобальные счётчики $N_{wtp} := N_{wt}$;
4 | | Сэмплировать $\boldsymbol{z}_p$: LDA-Gibbs-Sampling-Iter($\boldsymbol{w}_d$, $\boldsymbol{z}_d$, N_{tdp}, N_{wtp}, α, β);
5 | Синхронизация;
6 | Обновление глобальных счётчиков $N_{wt} := N_{wt} + \sum_{pin\{1,\dots,P\}}(N_{wtp} - N_{wt})$;
7 **до тех пор, пока** *не выполнен критерий останова*;

1. Каждый сэмплер производит обработку своих данных и обновление локальных z_p.
2. Обновляются локальные счётчики N_{tdp}.
3. Общий шаг синхронизации для обновления глобальных N_{wt}.
4. Новая N_{wt} копируется на процессоры и начинается следующая итерация обработки.

Эксперименты





Выводы

AD-LDA имеет ряд серьёзных недостатков:

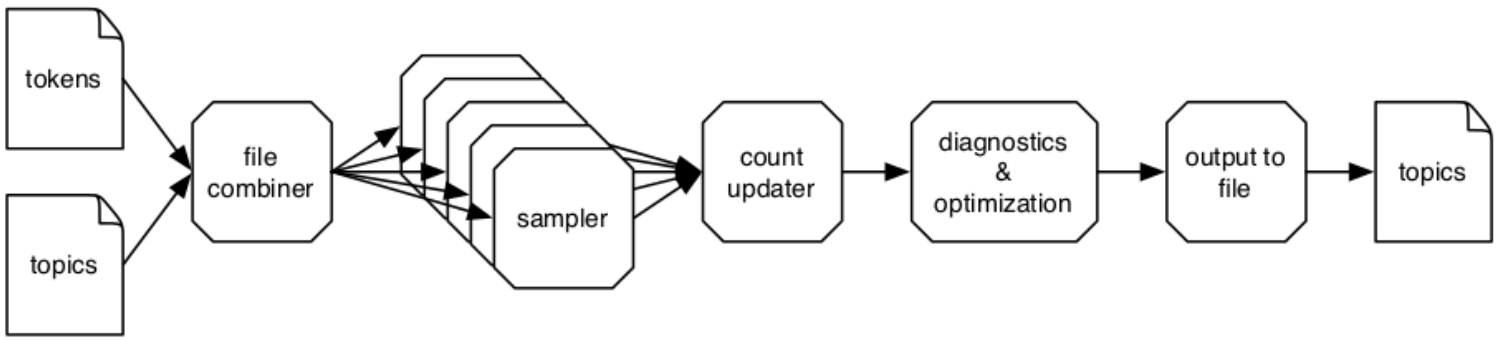
- Необходимость частых синхронизаций.
- Из-за синхронизации скорость определяется самым медленным процессором.
- Во время итераций сеть простаивает, а во время синхронизаций — перегружена.
- Большое потребление памяти — копия глобальных счётчиков N_{wt} хранится на каждом ядре.

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 ноде создаётся несколько потоков-сэмплеров.
- И один поток, задача которого — сливать полученные от сэмплеров обновления в глобальную N_{wt}.
- Глобальное состояние является общим для всех ядер нода.



Описание вычислений на кластере

Основной проблемой при реализации LDA на кластере является синхронизация глобальной (в рамках кластера) таблицы N_{wt} между всеми нодами-обработчиками.

Архитектура классной доски: глобальная таблица счётчиков N_{wt} хранится в единственном экземпляре и является общей для всех нодов, её обновление производится сэмплерами асинхронно по одному слову w за один раз.

Схема обновлений N_{wt}

Алгоритм синхронизации состояний в Y!LDA

1 Инициализировать $N_{wt} = N_{wt}^{i} = N_{wt}^{i\ old}$, для всех нодов i;
2 **до тех пор, пока** *производится сэмплирование* **выполнять**
3 Заблокировать глобально N_{wt} для некоторого слова;
4 Заблокировать локально N_{wt}^{i} для данного слова;
5 Обновить глобальное состояние: $N_{wt} := N_{wt} + (N_{wt}^{i} - N_{wt}^{i\ old})$;
6 Обновить локальное состояние: $N_{wt}^{i\ old} = N_{wt}^{i} = N_{wt}$;
7 Разблокировать N_{wt}^{i};
8 Разблокировать N_{wt};

- Глобальные и локальные счётчики находятся в одном и том же состоянии.
- N_{wt} — глобальное состояние.
- N_{wt}^{i} — текущее локальное состояние.
- $N_{wt}^{i\ old}$ — 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.
- Реализован на кластере рабочих станций с сервером и на Hadoop-кластере с машинами аналогичной мощности.

Инициализация и восстановление после сбоев

- В момент инициализации никаких присваиваний тем $\mathcal{Z}$ нет.
- Можно инициализировать моделью, обученной на небольшом подмножестве документов из D.
- По аналогичной схеме производится процедура восстановления в случае сбоя.

Достоинства Y!LDA

Таким образом, Y!LDA сразу 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 нодов.

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.
- Обработка производится в рамках парадигмы MapReduce и реализована на Hadoop:

Вариационный вывод имеет примущества по сравнению с MCMC.

1. Детерминированность результата, более высокая скорость сходимости в многомерных пространствах.
2. VB позволяет считать документы независимыми и делать их вывод параллельно.
3. Вариационный EM-алгоритм хорошо вписывается в MapReduce.

Описание схемы MapReduce

Все обозначения параметров описаны в формуле ?? (приведена ниже). В алгоритме присутствуют три вида компонентов — mapper, reducer и driver.

$$q(\mathcal{Z}, \Theta, \Phi) = \prod_{t=1}^{|T|} \mathrm{Dir}(\boldsymbol{\phi}_t|\boldsymbol{\lambda}_t) \prod_{d=1}^{|D|} \mathrm{Dir}(\boldsymbol{\theta}_d|\boldsymbol{\gamma}_d) \prod_{n=1}^{N_d} \mathrm{Mult}(z_{d,n}|\boldsymbol{\mu}_{d,n}).$$

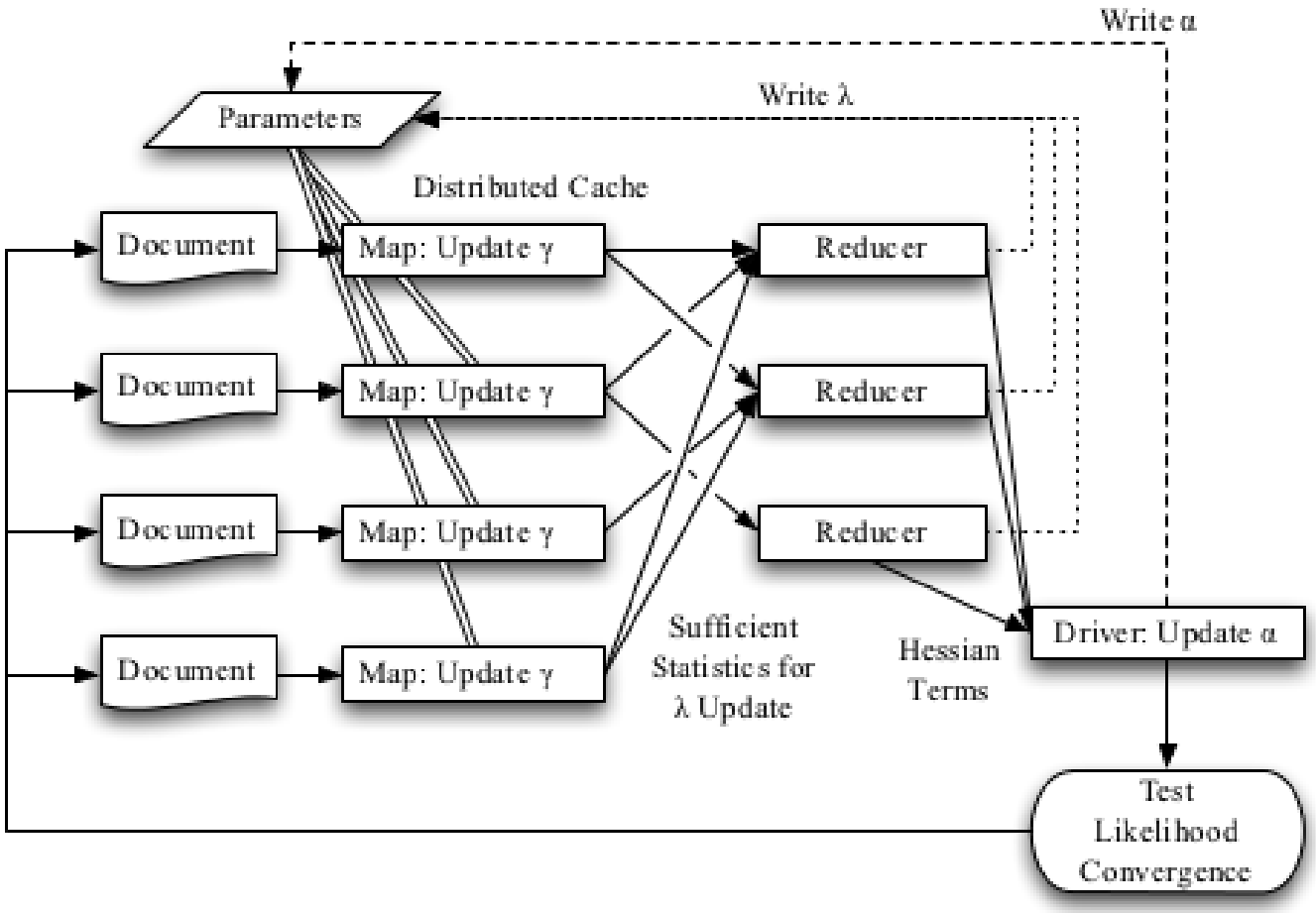
- На каждый документ коллекции создаётся mapper. Его задача — вычисление обновлений для связанных с документом гиперпараметров $\boldsymbol{\mu}_d$ и $\boldsymbol{\gamma}_d$.
- На каждую тему создается reducer. В его задачи входит пересчёт ассоциированного с темой t параметра $\boldsymbol{\lambda}_t$ с помощью обновлений, посчитанных mapper-ами.

Описание схемы MapReduce (продолжение)

- Третий компонент — driver — присутствует в системе в единственном экземпляре и управляет всем процессом обучения. Кроме этого, он осуществляет оптимизацию гиперпараметра α, тоже используя результаты работы mapper-ов, и вычисляет вариационную нижнюю оценку обоснованности.

Глобальные параметры $\boldsymbol{\lambda}$ и α хранятся в специальной, доступной только для чтения и общей для всех mapper-ов памяти, называемой *распределённым кэшем*.

Схема MapReduce



Стуруктурные модификации

Производительность Mr. LDA ограничивается двумя факторами:

1. огромным количеством генерируемых reducer-ами промежуточных значений;
2. временем, которое mapper-ам требуется для чтения текущих вариационных параметров перед началом своей работы.

Возможные модификации:

1. Кэширование выхода reducer.
2. Слияние файлов.

В основе BigARTM лежит теория ARTM (APTM — *аддитивная регуляризация тематических моделей* «Vorontsov K. V., Potapenko A. A. — Additive Regularization of Topic Models»).

- Задача тематического моделирования в ARTM сводится к поиску разложения матрицы вероятностей слов в документах в произведение Φ и Θ без введения априорных распределений.
- Задача некорректно поставленная и должна быть регуляризована.
- Регуляризаторы — дополнительные ограничения на правдоподобие.
- LDA является частным случаем этой модели с регуляризаторами сглаживания.
- PLSA является частным случаем этой модели без регуляризаторов.

ARTM (продолжение)

- Обучение можно производить обычным EM-алгоритмом, APTM добавит в его формулы M-шага новые слагаемые, получающиеся в результате применения регуляризации.
- Для обработки больших коллекций лучше использовать *пакетный онлайновый* вариант алгоритма. В нём пересчёт $\boldsymbol{\theta}_d$ вынесен в E-шаг и обработка документов производится по-пакетно. Такому алгоритму достаточно одного прохода по всей коллекции и нескольких итераций по каждому документу.

Описание BigARTM

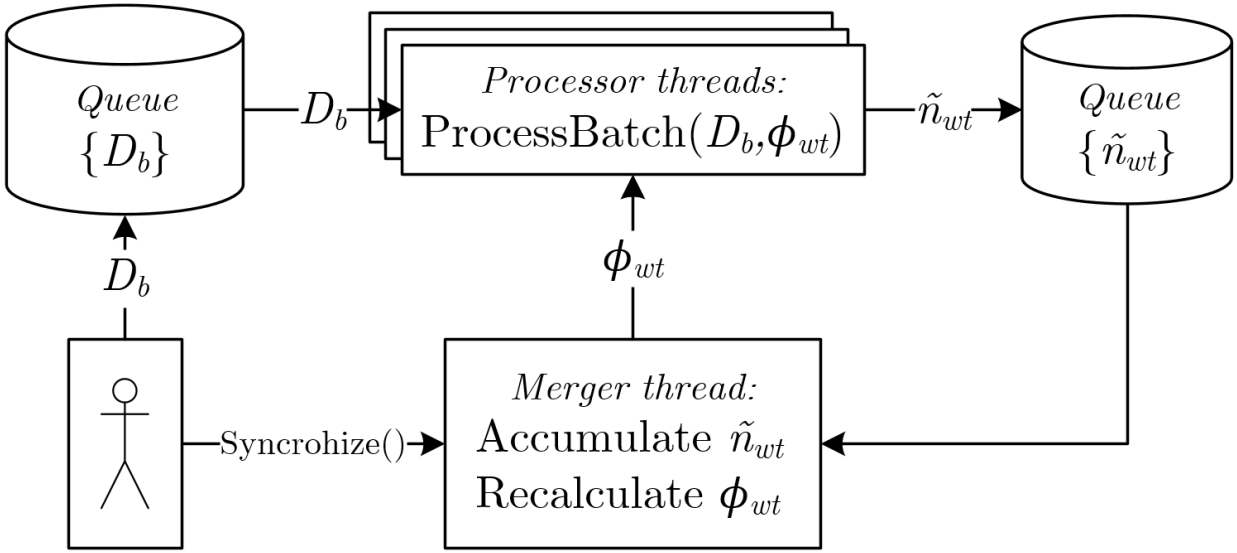
BigARTM — библиотека для эффективной онлайновой параллельной обработки прежде всего в пределах одной машины.

- Цель проектирования — независимость объёма используемой оперативной памяти от размера обрабатываемой коллекции.
- D делится на части (*батчи*) D_b, сохраняется на диск в отдельных файлах.
- В каждый момент времени в памяти находится только часть из них. В памяти хранится только часть Θ, соответствующая текущим обрабатываемым документам.

Старая архитектура

Параллелизм BigARTM основан на многопоточности.

- Множество *обработчиков* (Processor).
- Один *поток слияния* (Merger)
- Очередь заданий, откуда обработчики берут данные.
- Очередь слияния, в которую обработчики помещают результаты, и из которой их забирает поток слияния.



Старая архитектура (продолжение)

- 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*.
- Обновление матрицы Φ (синхронизацию) можно инициировать в любой момент из пользовательского кода.

Новая архитектура

Идея: зачем создавать специальный поток слияния, если он может стать узким местом всей архитектуры?

Решение: пусть потоки-обработчики пишут свои результаты напрямую в матрицу Φ.

Идея: матрицы Φ могут понадобиться самые разнообразные (как минимум p_{wt}, n_{wt}).

Решение: матриц типа Φ можно создавать любое количество вручную, они будут идентифицироваться именами и любая операция (например, нормализация или регуляризация) может быть применена к любой матрице.

Матрица может быть подана на чтение или на запись. В последнем случае матрица будет создана, если она не существовала.

Новая архитектура (продолжение)

Теперь вся работа по созданию матриц и выполнению операций над ними находится в руках пользователя.

Основные операции-кирпичи:

`ProcessBatches` — вход: p_{wt} и батчи, выход: n_{wt}.

`Regularize` — вход: p_{wt}, n_{wt}, выход: r_{wt}.

`Normalize` — вход: n_{wt}, r_{wt}, выход: p_{wt}.

`Merge` — вход: набор матриц n_{wt}^i и коэффициентов t^i, выход: n_{wt}, равная линейной комбинации входных матриц с соответствующими коэффициентами, состав слов результата — объединение слов слагаемых матриц.

В первых трёх операциях все матрицы должны иметь одинаковые размеры и наборы слов и тем.

Новая архитектура (продолжение)

Проблема: все описанные вызовы — блокирующие, т.е. требуется синхронизация после каждого завершённого вызова `ProcessBatches`.

Решение: ввести новую неблокирующую операцию `AsyncProcessBatches`. В результате можно организовать асинхронный конвейер для обработки коллекции в потоковом режиме.

При этом будет существовать несколько версий матрицы Ф, разной степени новизны.

Такое «запаздывание» не страшно с точки зрения качества, зато позволяет ускорить работу системы.

Эффективность и ARTM

- Как показывают эксперименты, BigARTM позволяет очень эфективно обрабатывать данные, получая на выходе хорошие модели. Ниже рассмотрено сравнение производительности BigARTM и двух популярных фреймворков, основанных на Online LDA: VW.LDA и Gensim.
- Поддержка APTM организована на модульном принципе, т.е. пользователь всегда может самостоятельно написать тот регуляризатор, который ему нужен, и включить его в библиотеку. Аналогичная ситуация с функционалами качества тематической модели.

API и мультимодальные тематические модели

- У библиотеки есть пользовательские API на C++ и Python, которые позволяют писать программы, очень тонко описывающие и постоянно контролирующие процесс обучения.
- BigARTM поддерживает т.н. *мультимодальные тематические модели* (MTM), которые позволяют использовать мета-информацию, связанную с документами, строить мультиязычные модели. В рамках теории MTM расписываются многие сложные байесовские модели, основанные на LDA.

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. Принципиально этот алгоритм ничем не отличается от пакетного онлайнового PLSA.

Vowpal Wabbit LDA:

- Онлайновый.
- Не параллельный.
- Реализован на C++ без STL и сторонних библиотек.
- Эффективный инструмент для моделирования больших коллекций в потоковом режиме.

Online LDA: Gensim

Gensim:

- Онлайновый.
- Параллельный (однопоточная реализация LdaModel, многопоточная — LdaMulticore). Архитектура похожа на BigARTM.
- Распределённый (неэффективная реализация).
- Реализован на Python.
- Эффективный инструмент для моделирования больших коллекций в потоковом режиме.

BigARTM (старая архитектура) vs. VW.LDA vs. Gensim

Библиотека	Число процессоров	Время обучения модели [1]
BigARTM	1	35 минут
LdaModel	1	369 минут
VW.LDA	1	73 минуты
BigARTM	4	9 минут
LdaMulticore	4	60 минут
BigARTM	8	4.5 минуты
LdaMulticore	8	57 минут

[1] Использовалась коллекция документов английской Википедии, $|D| \approx 3.7 \times 10^6$.

Список литературы I


D. M. Blei, A. Ng, and M. Jordan. (2003). Latent Dirichlet allocation, // Journal of Machine Learning Research, vol. 3, pp. 993–1022.


D. Newman, A. Asuncion, P. Smyth, and M. Welling. (2009). Distributed algorithms for topic models, // NIPS.


Alexander Smola and Shravan 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.


Воронцов К. В.. Аддитивная регуляризация тематических моделей коллекций текстовых документов, // Доклады РАН. 2014. — Т. 455., No3. 268–271.

Список литературы II

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.