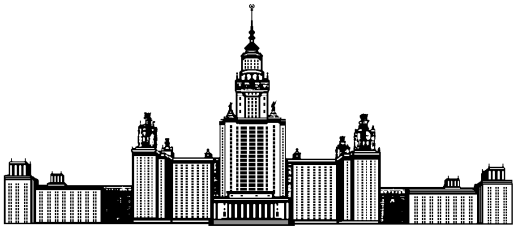
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики
Кафедра Математических Методов Прогнозирования

КУРСОВАЯ РАБОТА СТУДЕНТА 317 ГРУППЫ

«Со-встречаемости токенов в больших текстовых коллекциях»

«Co-occurrence of tokens in large text collections»

Выполнил:
студент 3 курса 317 группы
Солоткий Михаил

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

Москва, 2018

Содержание

Аннотация

В данной работе приводится параллельный алгоритм сбора статистики совстречаемостей токенов в больших текстовых коллекциях, таких как английская Википедия, а также эксперименты по измерению когерентности тематических моделей.

1 Введение

Тематическое моделирование — это одно из современных направлений статистического анализа текстов. Вероятностная тематическая модель (probabilistic topic model) выявляет тематику коллекции документов, представляя каждую тему дискретным распределением вероятностей токенов, а каждый документ — дискретным распределением вероятностей тем.

Одним из программных инструментов решения задачи тематического моделирования является библиотека с открытым кодом BigARTM [8]. В BigARTM обработка данных производится с помощью онлайнового параллельного алгоритма, что даёт высокую производительность и возможность обрабатывать большие текстовые коллекции.

Тема называется когерентной (согласованной), если наиболее частые токены данной темы часто встречаются рядом в документах коллекции. Оказалось [4], что чем когерентность коррелирует с человеческими оценками интерпретируемости темы.

В данной работе предложен параллельный алгоритм сбора статистики совстречаемости токенов в текстовых коллекциях неограниченной длины, который дополняет функциональность библиотеки BigARTM. Собранная статистика используется для вычисления когерентности тем для выявления априори наиболее интерпретируемых тем.

2 Тематическая модель PLSA

Пусть D — конечное множество документов (коллекция), W — конечное множество токенов данной коллекции (словарь). Обычно токенами являются слова, но могут быть и словосочетания. Предполагается, что появление произвольного токена w в произвольном документе d связано с некоторой темой t, принадлежащей конечному множеству тем T. Следующими модельными предположением является гипотеза «мешка слов», которая утверждает, что для выявления тематики документа не важен порядок токенов в документе, а важны лишь частоты вхождения токенов. Также вводится гипотеза «мешка документов», которая говорит, что для выявления тематики коллекции не важен порядок документов. Эти две гипотезы дают возможность построить вероятностную модель порождения текстовой коллекции. Пространством элементарных исходов является $\Omega = W \times D \times T$, а вся коллекция рассматривается как простая выборка троек (w_i, d_i, t_i) из категориального распределения $\mathsf{P}(w, d, t)$, причём темы токенов t_i являются скрытыми, а наблюдаются пары (w_i, d_i). Предполагается, что вероятность появления токена в документе связана с его темой и не связана с документом, в котором он встретился. Формально это записывается так: $\mathsf{P}(w|d,t) = \mathsf{P}(w|t)$. Таким образом тема есть вероятностное распределение на W.

Модель PLSA (probabilistic latent semantic analysis) [2]:

$$\mathsf{P}(w|d) = \sum_{t \in T} \mathsf{P}(w|t) \cdot \mathsf{P}(t|d) = \sum_{t \in T} \Phi_{wt} \cdot \Theta_{td}$$

Задача состоит в восстановлении матриц Φ и Θ по выборке, то есть в нахождении описания токенов и текстовых документов с помощью смеси тем. Восстановление матриц происходит посредством максимизации правдоподобия с помощью EM-алгоритма. Одной из популярных в компьютерной лингвистике мер качества оценки языковых моделей является перплексия. Применительно к модели PLSA она принимает вид:

$$\mathcal{P}(D) = \exp\left(-\frac{1}{n} \sum_{d \in D} \sum_{w \in d} n_{wd} \ln\big(p(w|d)\big) \right)$$

$$n = \sum_{d \in D} \sum_{w \in d} n_{wd}$$

n_{wd} — количество раз, сколько токен w встретился в документе d. По сути формула представляет собой усреднённый по всем токенам коллекции логарифм правдоподобия, от которого затем взята обратная экспонента.

3 Модели со-встречаемостей токенов

Недостатком рассмотренных тематических моделей является то, что они никак не учитывают взаимное расположение токенов в коллекции. Согласно гипотезе дистрибутивности [1]: токены близки семантически, если совместно часто встречаются близко в тексте. Информация о том, как часто некоторые токены встречаются рядом в коллекции могла бы помочь построить более точную модель с более интерпретируемыми темами. Интерпретируемость — субъективное понятие. Процесс оценивания темы выглядит следующим образом: эксперт в некоторой предметной области получает топ k токенов некоторой темы (обычно $k = 10$) и должен без труда понять, что

связывает полученные токены и дать теме адекватное название. Затем по некоторое шкале выставляет оценку: насколько тема интерпретируема.

3.1 Когерентность тем

В [4] проводились попытки подобрать автоматически вычисляемую меру качества темы, которая бы коррелировала с оценками, выставленными экспертами (в смысле корреляции Спирмена). Лучшей мерой качества оказалась когерентность — средняя PMI (Pointwise Mutual Information) [3] по топ k токенам темы.

$$Coher_t = \underset{i,j=1,\ldots,k}{mean} PMI(w_i, w_j)$$

$$PMI(u, v) = \log \frac{\mathsf{P}(u, v)}{\mathsf{P}(u)\,\mathsf{P}(v)}$$

Также в [5] было показано, что использование Positive PMI вместо PMI улучшает качество в задачах семантической близости слов.

$$PPMI(u, v) = \max(0, PMI(u, v))$$

3.2 PPMI на частотах пар токенов

Рассмотрим в документе d множество позиций токенов. Скажем, что пара позиций (i, j) находится в некотором окне ширины k, если $0 < |i - j| \leqslant k$. Через w_{di} обозначим токен, находящийся на позиции i в документе d. Через n_{uv} обозначим количество позиций (i, j) суммарно во всех документах коллекции, принадлежащих некоторому окну и таких, что $w_{di} = u$, $w_{dj} = v$. Формально можно записать следующим образом:

$$n_{uv} = \sum_{d=1}^{|D|} \sum_{i=1}^{n_d} \sum_{j=1}^{n_d} [0 < |i - j| \leqslant k][w_{di} = u][w_{dj} = v]$$

Обозначим $n = \sum\limits_{u,v} n_{uv}$. Вводится совместное вероятностное распределение на множестве пар токенов следующим образом:

$$\mathsf{P}(u, v) = \frac{n_{uv}}{n}$$

Тогда вероятность токена u в коллекции выражается:

$$\mathsf{P}(u) = \sum_{v} \mathsf{P}(u, v)$$

Итоговая формула для PPMI:

$$PPMI(u, v) = \frac{n_{uv} \cdot n}{n_u \cdot n_v}$$

Данная величина есть доля пар, в которых участвовал токен u. Такой подход к учёту частот был предложен в [6].

3.3 PPMI на частотах документов

Ещё одиним популярным способом подсчёта вероятностей для PPMI является учёт количества документов, в которых фигурировала хотя бы раз данная пара токенов в окне заданной ширины. Данный подход рассмотрен в [7].

$$n_{uv} = \sum_{d=1}^{|D|} [\exists\, i, j : w_{di} = u, w_{dj} = v, 0 < |i - j| \leqslant k]$$

Вводится совместное вероятностное распределение на множестве документов следующим образом:

$$\mathsf{P}(u, v) = \frac{n_{uv}}{|D|}$$

$$\mathsf{P}(u) = \frac{n_u}{|D|}$$

Тогда PPMI можно переписать следующим образом:

$$PPMI(u, v) = \frac{n_{uv} \cdot |D|}{n_u \cdot n_v}$$

4 Алгоритм подсчёта со-встречаемостей

В данной работе предложен алгоритм обработки текстовых коллекций и подсчёта статистики со-встречаемостей пар токенов. Особенности данного алгоритма:

- асинхронная обработка входной коллекции и промежуточных данных;
- возможность обработки потенциально неограниченных по числу документов коллекций;
- улучшение скорости работы при увеличении числа потоков на одном узле;

Условно алгоритм можно разбить на 3 этапа:

- обработка входной коллекции;
- агрегация статистики, собранной по коллекции;
- вычисление метрик на основе со-встречаемости.

4.1 Обработка входной коллекции

Обработка входной коллекции происходит по батчам, т.е. в оперативную память считывается *batch_size* документов коллекции и считаются со-встречаемости на этом подмножестве документов, как будто они составляют всю коллекцию. Параметр *batch_size* указывается пользователем. Также пользователь может указать некоторый словарь релевантных для него токенов — *vocab*, и со-встречаемости будут считаться только между токенами из данного множества. Дело в том, что множество всех токенов некоторого естественного языка может быть довольно большим, а всевозможные пары токенов со статистикой по каждой паре невозможно хранить в оперативной памяти. Таким образом *vocab* может представлять собой словарь коллекции

W, очищенный от стоп-слов и низкочастотных токенов, или он может состоять из токенов некоторого узкого набора тем. Статистика со-встречаемостей по каждому батчу сохраняется в файле в отсортированном формате для того, чтобы потом можно было легко объединить эти файлы в один. Также для вычисления $PPMI$ нужно значение n — общее количество рассмотренных пар в коллекции. Это значение можно вычислить на первом этапе алгоритма. Обработка батчей происходит параллельно асинхронно в n_{jobs} потоков.

Этап 1. Обработка входной коллекции.

Вход: коллекция D, $vocab$, $batch\_size$, ширина окна $width$, n_{jobs};
Выход: набор отсортированных файлов со-встречаемостей F, общее количество пар n;

$n \leftarrow 0$;
пока не конец коллекции
  для всех $job = 1, \ldots, n_{jobs}$
    инициализировать пустой сортирующий контейнер C;
    считать $batch\_size$ документов в $batch$;
    для всех $d \in batch$
      для всех $i = 1, \ldots, n_d - 1$
        для всех $j = 1, \ldots, width : \ i + j \leqslant n_d$
          $r = i + j$;
          $C[w_{di}, w_{dr}] \leftarrow C[w_{di}, w_{dr}] + 1$;
          $C[w_{dr}, w_{di}] \leftarrow C[w_{dr}, w_{di}] + 1$;
          $n \leftarrow n + 2$
    сохранить C во внешнюю память;

В алгоритме используется сортирующий контейнер C. Для эффективной обработки коллекции необходимо, чтобы была возможность быстрого доступа к его элементам и их изменение. Для реализации было выбрано двухуровневое красно-чёрное дерево. Это значит, что вначале по ключу (токену u) находятся все токены, с которыми оно встречалось в коллекции. А затем снова по ключу (токену v) во внутреннем красно-чёрном дереве ищется значение со-встречаемости пары (u, v). Файлы построены по тому же принципу, то есть сначала указан первый токен, а затем все, с которыми он встречался, и значение со-встречаемости.

4.2 Агрегация собранной статистики

Основой второго этапа является сортировка во внешней памяти с помощью алгоритма k-way merge (ссылочку). Алгоритм в случае $k = 2$ совпадает со слиянием 2 отсортированных массивов, а в случае $k > 2$ строит пирамиду на минимальных элементах массивов. В данном случае массивы — это файлы, а элементу массива соответствует запись в файле, которая содержит некоторый токен u и все токены, с которыми он встречался в некотором батче, а также значение со-встречаемости. Данный алгоритм не требует хранения содержимого файлов целиком в оперативной памяти, поэтому идеально подходит для сортировки во внешней памяти. В оперативной памяти нужно хранить лишь 1 запись для каждого файла. Так как число

файлов потенциально неограниченно, алгоритм в явном виде нельзя применять, и используется его модифицированная версия: если количество файлов слишком большое, вначале происходит серия последовательных слияний малых порций файлов с помощью этого же алгоритма до тех пор, пока количество массивов не станет меньше заданной константы. Ситуация, когда количество файлов слишком велико и происходит предварительное слияние некоторых порций файлов, реализуется в случаях больших коллекций или малых значений $batch\_size$. Так как для слияния необходимо держать файлы открытыми, количество одновременно объединяемых файлов не может превосходить максимально возможного для данной операционной системы количества открытых файлов, иначе придётся производить открытие-закрытие файлов, что довольно дорогая операция.

Все слияния можно производить многопоточно: каждый поток будет иметь свой локальный набор файлов. Многопоточное слияние нужно прекратить по достижении какого-то малого количества файлов. Также на этом этапе алгоритма удобно посчитать частоты токенов n_u, которые будут использованы для вычисления $PPMI$. Во время последнего слияния файлов можно не записывать в итоговый файл те пары, значения со-встречаемости которых ниже некоторого заданного порога, так как низкочастотные пары не несут статистически значимой информации. Также это поможет сэкономить время исполнения на 3 этапе алгоритма.

Этап 2. Слияние отсортированных файлов.

Вход: набор отсортированных файлов со-встречаемостей F,
минимальное число файлов для окончательного слияния min_merge,
максимальное число открытых файлов max_open,
число потоков n_jobs, минимальное значение со-встречаемости min_cooc;
Выход: файл со-встречаемостей f, частоты токенов n_u;

1 **пока** $|F| > min\_merge$
2 $F' \leftarrow \{\}$;
3 **если** $|F| > max\_open$ **то**
4 **пока** $|F| > 0$
5 **для всех** $i = 1, \ldots, n\_jobs$
6 $batch\_size \leftarrow \min(\lfloor \frac{max\_open}{n_{jobs}} \rfloor, |F|)$;
7 $batch \leftarrow F_1, \ldots, F_{batch\_size}$;
8 $F \leftarrow F \setminus batch$;
9 $f \leftarrow k$-way merge$(batch)$;
10 Добавить f в F';
11 **иначе**
12 Сделать шаги $5 - 10$;
13 $F \leftarrow F'$
14 $f \leftarrow k$-way merge(F, min_cooc);

4.3 Вычисление метрик

Последний этап алгоритма заключается в вычислении метрик по имеющимся значениям со-встречаемости. В текущей реализации считается только $PPMI$, однако добавление других метрик не составляет труда.

Этап 3. Вычисление $PPMI$.

Вход: файл со-встречаемостей f';
Выход: файл $PPMI$ f';

```
для всех (u, v, n_uv) ∈ f
    ratio ← n_uv n / (n_u n_v);
    если ratio > 1 то
        PPMI ← log(ratio);
        f' ← PPMI;
```

4.4 Требования к коллекции

Для корректной работы алгоритма коллекция должна быть представлена в формате Wowpal Wabbit. Также на коллекцию и словарь *vocab* накладываются следующие ограничения:

(1) документ целиком можно записать в оперативную память;

(2) оперативной памяти хватает для записи словаря *vocab* вместе с 500 копиями;

(3) оперативной памяти хватает для хранения всех пар токенов любого документа и счётчиков со-встречаемостей в двухуровневом красно-чёрном дереве;

(4) оперативной памяти хватает для хранения всех пар токенов словаря и счётчиков со-встречаемостей в двухуровневом красно-чёрном дереве.

Достаточно выполнения ограничений (1), (2) и любого из (3), (4).

4.5 Анализ сложности

4.5.1 Анализ первого этапа

Время работы первого этапа алгоритма линейно зависит от общей длины коллекции и от ширины окна, так как для почти всех токенов в коллекции (за исключением первых и последних $width$ токенов в каждом документе) необходимо просмотреть $width$ токенов, стоящих справа. Также для каждой найденной пары токенов необходимо изменить счётчики со-встречаемости в красно-чёрном дереве, то есть время работы первого этапа есть $\mathcal{O}\big(N \cdot width \cdot \log(n_{pairs})\big)$, где N — общее количество токенов в коллекции, а n_{pairs} — верхняя оценка на количество пар в красно-чёрном дереве. Так как единовременно в память пишется $batch\_size$ документов, можно утверждать $n_{pairs} = width \cdot \sum_{i=1}^{batch\_size} n_d = \{\text{ограничение (3)}\} = const$. С другой стороны, можно получить другую верхнюю оценку $\hat{n}_{pairs} = |vocab|^2 = \{\text{ограничение (4)}\} = const$.

Память на первом этапе тратится на содержание батча документов и красно-чёрного дерева. Отсюда необходимое условие (1). При выполнении условий (3) или

(4) можно гарантировать, что затраты по памяти будут $\mathcal{O}\big(n_{jobs} \cdot batch\_size\big)$. Выполнения условия (3) можно добиться, если предполагать, что отдельный документ мал по размеру, что как раз реализуется на практике: обычно текстовые документы представляют собой статьи из Википедии, новости или посты в социальных сетях.

4.5.2 Анализ второго этапа

В стандартном k-way merge время доступа к минимальному элементу есть $\mathcal{O}(1)$, а время перестроения после изменения положения указателя в одном файле есть $\mathcal{O}\big(\log k\big)$, где k — количество файлов. Общее время работы есть $\mathcal{O}\big(N \cdot \log k \cdot |vocab|\big)$, где N — верхняя оценка на количество записей в файлах, так как для слияния может понадобиться $\mathcal{O}\big(|vocab|\big)$ операций. Так как в данном случае всё множество файлов разбивается на части, на первом проходе по файлам время работы будет составлять $\mathcal{O}\big(N \cdot \frac{|F|}{max\_open} \cdot \log(max\_open)\big) = \mathcal{O}\big(|vocab|^2 \cdot |D|\big)$, где $|F| = \lceil \frac{|D|}{batch\_size} \rceil$. Всего проходов по файлам до достижения минимального количества происходит $\lceil \log_{max\_open} \frac{|F|}{min\_files} \rceil$, поэтому общее время работы можно оценить как $\mathcal{O}\big(|D| \cdot \log |D| \cdot |vocab|^2)\big)$.

На втором этапе требуется $\mathcal{O}\big(|vocab|\big)$, так как надо хранить записи из max_open файлов, где max_open — количество элементов в пирамиде и максимальное количество открытых файлов, а также 1 запись, в которой будет производиться слияние записей. Так как алгоритм предназначается для работы с текстами на естественном языке, можно предполагать, что словарь $vocab$ можно записать в оперативную память $500 + 1$ раз. Значение 500 было взято, как нижняя оценка на число открытых файлов, типичное для современных операционных систем. Отсюда ограничение (2).

4.5.3 Анализ третьего этапа

Третий этап алгоритма совсем тривиальный, требует $\mathcal{O}(1)$ памяти и $\mathcal{O}\big(|vocab|^2\big)$ времени.

4.6 Реализация алгоритма

Описанный выше алгоритм был реализован в библиотеке тематического моделирования с открытым кодом BigARTM [8] [1] [2] [3] на языке C++. Язык был выбран из соображений скорости и лёгкости использования стандартных структур данных. Например, в качестве красно-чёрного дерева использовался контейнер std::map.

5 Влияние параметров

Среди перечисленных параметров, с которыми запускается алгоритм есть параметр $batch\_size$, варьируя который можно получать разное время работы, а также разный суммарный размер файлов, которые сохраняются во внешней памяти.

[1] `http://bigartm.org` — сайт проекта BigARTM.

[2] `http://docs.bigartm.org/en/stable/tutorials/python_userguide/coherence.html` — тюториал по измерению когерентности в BigARTM.

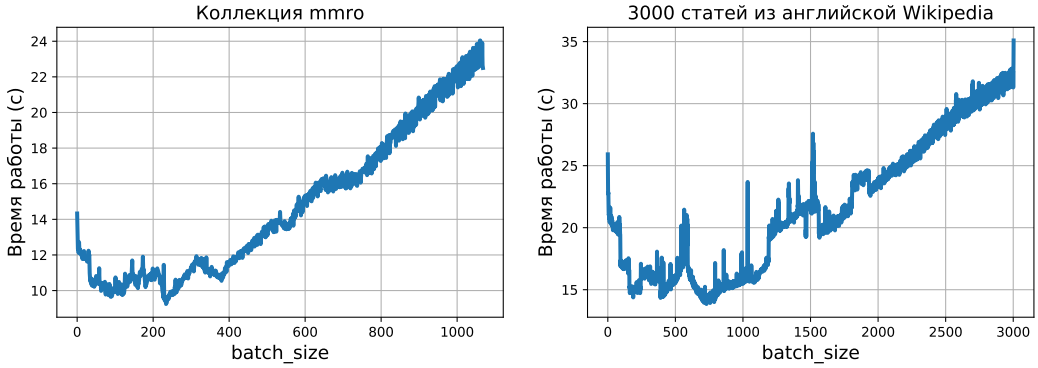
[3] `http://docs.bigartm.org/en/stable/tutorials/bigartm_cli.html` — документация к CLI BigARTM и инструкция по запуску алгоритма.

Некоторые пары токенов встречаются в большом количестве документов, поэтому если пара (i, j) встретилась в разных батчах, она будет записана в разные файлы несколько раз. Чтобы избежать подобного дублирования, стоит увеличивать параметр $batch\_size$, однако вместе с его увеличением растёт затрачиваемая оперативная память, и при больших значениях $batch\_size$ есть риск, что очередной $batch$ документов не поместиться в оперативную память, и программа завершится с ошибкой. С другой стороны, если коллекцию можно записать целиком в оперативную память, для минимизации времени работы лучше не брать значение $batch\_size = |D|$, так как никакого эффекта от параллелизма не будет.

Все эксперименты выполнялись на ноутбуке с 4х-ядерным процессором AMD A6-7310.

5.1 Реальное время работы

Проводилось несколько экспериментов по замеру реального времени работы алгоритма на больших и малых коллекциях. В качестве малых коллекций была взята коллекция записей выступлений на конференции ММРО (1069 документов), а также 3000 статей из английской Википедии. В коллекции ММРО 7805 уникальных токенов и 804423 токенов всего. В 3000 статьях Википедии 112407 уникальных токенов и 899343 токенов всего. Словарь $vocab$ совпадал с полным словарём коллекции W, а параметр ширины окна $width = 10$. Для определения зависимости времени работы от параметра $batch\_size$ алгоритм запускался для каждого значения от 1 до размера коллекции включительно. Число потоков равно 4. Полученные графики времени работы представлены ниже.



Видно, что на заключительном участке время растёт, то есть эффект от параллелизма исчезает, а время обработки коллекции становится временем обработки самого большого батча. На начальном участке время работы становится меньше с увеличением размера батча, а глобальный минимум достигается примерно на четверти коллекции. Соответственно, совет по выбору $batch\_size$: выбирать настолько большое значение, которое позволяет оперативная память, но не больше, чем $\frac{|D|}{n_{jobs}}$.

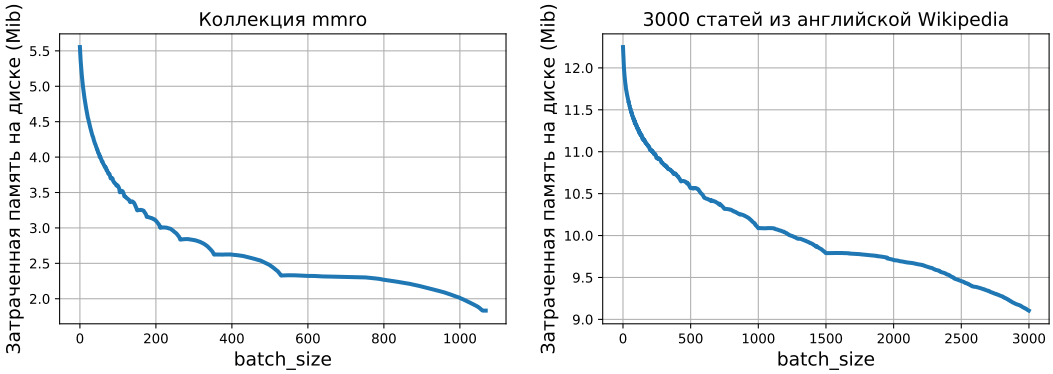
Также было проведено 2 эксперимента с большими коллекциями:

- 8446835 статей английской Википедии, 8272855 уникальных токенов, 3832966193 токенов всего, $width = 1$, $batch\_size = 5000$
- посты социальных сети (4528512 документов), 87494 уникальных токенов, 1613807215 токенов всего, $width = 10$, $batch\_size = 15000$

При значениях остальных параметров как в предыдущих экспериментах время работы составило **3 часа 24 минуты 24 секунды** на Википедии и **5 часов 50 минут 16 секунд** на постах социальных сетей.

5.2 Затраченная память

Значение *batch_size* стоит выбирать большим также из соображений экономии памяти на внешнем устройстве. В этом эксперименте замерялся суммарный размер файлов, которые получались в результате работы первого этапа алгоритма. Параметры алгоритма и коллекции те же, что и экспериментах по замеру времени. Исходный размер коллекции MMRO составлял примерно 13.5 MiB, а урезанная до 3000 статей Википедия занимала примерно 9.7 MiB.



При запуске на больших коллекциях затраченная на внешнем носителе память начинает сильно превосходить размер исходной коллекции: статьи Википедии занимают примерно 20.4 GiB, а при работе алгоритма промежуточные файлы занимают около 1.66 GiB. Коллекция постов социальных сетей занимала исходно около 26 GiB, а промежуточные файлы — около 3.87 GiB.

Оперативной памяти во всех экспериментах было затрачено не больше 8 GiB.

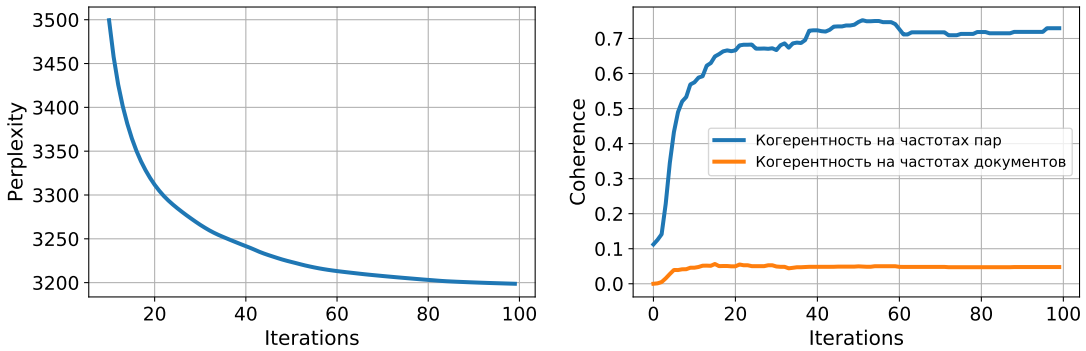
5.3 Выводы

Алгоритм можно запускать на ноутбуке, и за приемлемое время получить результат. На практике со-встречаемости по большой коллекции подсчитываются один раз перед построением тематической модели или проведением других экспериментов с использованием данной информации. Относительно времени, которое ушло на сбор статистики по коллекции, дальнейшие исследования занимают, как правило, намного больше времени. Также дальнейшее обновление статистики в случае пополнения коллекции не вызывает сложностей: надо лишь запустить алгоритм ещё раз на дополнительной части коллекции, объединить файлы со-встречаемостей и пересчитать PPMI.

6 Измерение когерентности

6.1 Обучение модели

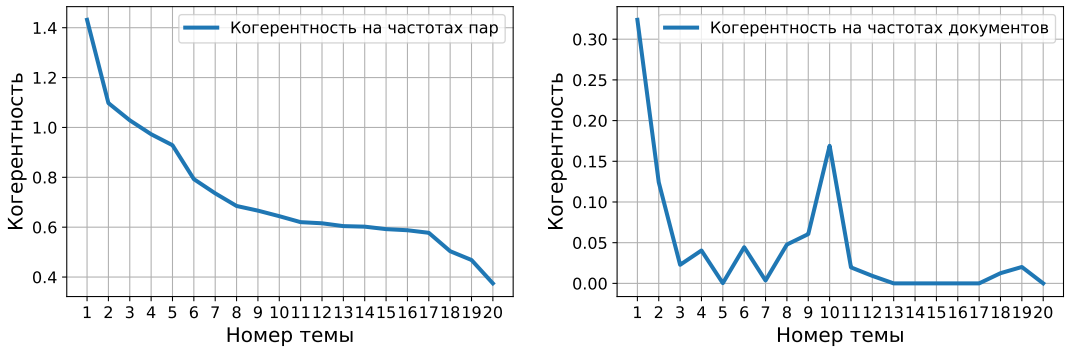
Была обучена модель PLSA на коллекции статей «Постнаука». В коллекции 3446 документов. Была произведена предварительная токенизация, лемматизация и удаление стоп-слов. С помощью описанного выше алгоритма была собрана статистика со-встречаемостей по всем токенам коллекции и посчитаны PPMI. Ограничений на минимальное значение со-встречаемости не выставлялось. Обучение проводилось с помощью библиотеки BigARTM. Количество тем было взято равным 20, количество итераций — 200. После каждой итерации EM-алгоритма замерялась средняя по всем темам когерентность и перплексия модели. Ниже представлены графики перплексии и двух видов когерентности (на частотах пар токенов и на частотах документов) на каждой итерации EM-алгоритма. Когерентность считалась по 10 топ-токенам каждой темы.



6.2 Интерпретируемость тем

Как видно, оба вида когерентности растут по мере восстановления тем. Также видно, что когерентность, для которой использовались частоты пар токенов растёт немонотонно.

Далее отсортируем 2 вектора когерентности тем по когерентности, основанной на частотах пар.



Видно, что темы с высокой когерентностью в смысле попарных частот токенов являются высоко когерентными и в смысле частот документов, однако для промежуточных тем ситуация сильно меняется. К примеру, для тем № 5, 13–17, 20 второй вид когерентности почти равен нулю. Ниже визуализированы топ-слова данных тем.

Тема №1	Тема №7	Тема №10	Тема №20
страна	век	социальный	город
экономический	свой	теория	социальный
экономика	история	мир	общество
рынок	культура	социология	пространство
рост	территория	объект	политический
свой	восток	наука	культура
компания	самый	свой	культурный
цена	остров	событие	свой
большой	народ	действие	группа
деньга	китай	социолог	государство

Заметим, что все из представленных тем почти все интерпретируемы: тема № 1 про экономический рост, тема № 7 явно про историю Востока, тема № 10 про социологию как науку, а тема № 20 похожа на комбинацию тем про политику, культуру и общество. Последней теме сложно дать некоторое осмысленное название, что, как раз коррелирует со значением когерентности, основанной на частотах пар токенов. У второго вида когерентности все значения примерно равны 0 для тем № 7 и 20.

6.3 Выводы

- когерентность возрастает по мере обучения тематической модели;
- когерентность, основанная на попарных частотах токенов лучше коррелирует с интерпретируемостью тем, чем когерентность, основанная на частотах документов в коллекции, в которых встретились пары токенов.

Второй результат можно объяснить тем, что коллекция «Постнаука» — сборник докладов на большое число различных тем, и число документов, в которых встретилась некоторая пара терминов некоторой темы может быть довольно малым (всего в коллекции 3446 документов) для восстановления вероятностей, но количество вхождений некоторой пары токенов в коллекцию намного больше, что позволяет на небольших коллекциях лучше восстанавливать вероятности. Эта гипотеза требует подтверждения.

7 Заключение

7.1 Результаты, выносимые на защиту

- Предложен и реализован алгоритм по сбору статистики со-встречаемостей на текстовых коллекциях неограниченной длины;
- Проведены эксперименты по измерению когерентности тематической модели.

7.2 Направления дальнейших исследований

В описанный алгоритм легко встраивается вычисление большого числа различных метрик на основе попарной статистики токенов. Есть идеи по созданию
n-граммера на основе данного алгоритма, который бы смог за приемлемое время выделять коллокации на таких коллекциях как Википедия. Также остался не отвеченным вопрос: почему некоторые интерпретируемые темы получали очень низкие

оценки когерентности, основанной на частотах документов. Гипотеза заключается в том, что такой результат связан с недостаточной длиной коллекции.

Список литературы

[1] Zellig Harris. Distributional structure. Word, 10(23):146–162, 1954.

[2] 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.

[3] Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. Comput. Linguist. 16, 1 (March 1990), 22–29.

[4] Newman D., Lau J.H., Grieser K., Baldwin T. Automati evaluation of topi oherene // Human Language Tehnologies, HLT-2010.

[5] Bullinaria, J.A. & Levy, J.P. Behavior Research Methods (2007) 39: 510. https://doi.org/10.3758/BF03193020

[6] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In Advances in Neural Information Processing Systems, pages 2177–2185, 2014.

[7] Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving Distributional Similarity with Lessons Learned from Word Embeddings. Transactions Of The Association For Computational Linguistics, 3, 211–225.

[8] Vorontsov 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.