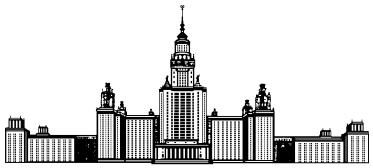


Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Вероятностные тематические модели на основе данных о со-встречаемости слов

Выполнил:
студент 4 курса 417 группы
Солоткий Михаил

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

Москва, 2019

Содержание

Аннотация

Данные о совместной парной встречаемости слов (co-occurrence data) имеют несколько важных применений в вероятностных тематических моделях. Они используются для оценивания когерентности тем, для построения модели битермов (BitermTM) и сетей слов (Word Network Topic Model, WNTM), в регуляризаторах когерентности. В данной работе разработан и реализован параллельный алгоритм для эффективного вычисления статистики со-встречаемостей слов по большой текстовой коллекции. Предлагается несколько способов вычисления когерентности. Исследуется экспериментальные методики подбора коэффициентов регуляризации и числа тем с использованием оценок когерентности.

1 Введение

1.1 Тематическая модель PLSA

Пусть D — конечное множество документов (коллекция, корпус), 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. Следующим модельным 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) [3]:

$$\mathsf{P}(w|d) = \sum_{t\in T} \mathsf{P}(w|t)\,\mathsf{P}(t|d) = \sum_{t\in T} \varphi_{wt}\theta_{td}$$

Каждой позиции каждого токена в каждом документе присваивается своя скрытая переменная t — тема данного токена на данной позиции в данном документе.

Далее будут использоваться следующие обозначения (аналогичные [10]):

n_{dwt} — счётчик, сколько раз токен w был ассоциирован с темой t в документе d;
$n_{wt} = \sum_d n_{dwt}$ — счётчик, сколько раз в коллекции токен w ассоциирован с темой t;
$n_{dt} = \sum_w n_{dwt}$ — количество токенов из документа d, ассоциированные с темой t;
$n_{dw} = \sum_t n_{dwt}$ — количество вхождений токена w в документ d;
$n_d = \sum_{wt} n_{dwt}$ — длина документа d;
$n_w = \sum_{dt} n_{dwt}$ — количество вхождений токена w в коллекцию;
$n_t = \sum_{dw} n_{dwt}$ — количество позиций токенов в коллекции, принадлежащих теме t.

Таким образом можно выразить через эти счётчики элементы матриц Φ, Θ:

$$\varphi_{wt} = \frac{n_{wt}}{n_t}, \qquad \theta_{td} = \frac{n_{td}}{n_d}$$

При этом мы наблюдаем частотные оценки $\hat{\mathsf{P}}(w|d) = \dfrac{n_{dw}}{\sum\limits_{w \in d} n_{dw}} = \dfrac{n_{dw}}{n_d}$. 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 тематической модели $\mathsf{P}(W|D)$ с помощью EM-алгоритма:

$$\mathcal{L}(\Phi, \Theta) = \sum_{d \in D} \sum_{w \in W} n_{dw} \ln \sum_{t \in T} \varphi_{wt} \theta_{td} \to \max_{\Phi, \Theta},$$

при ограничениях неотрицательности и нормировки:

$$\begin{cases} \sum\limits_{w \in W} \varphi_{wt} = 1 & \forall t \in T \\ \sum\limits_{t \in T} \theta_{td} = 1 & \forall d \in D \\ \varphi_{wt} \geqslant 0 & \forall w \in W,\ \forall t \in T \\ \theta_{td} \geqslant 0 & \forall t \in T,\ \forall d \in D \end{cases}$$

Одной из популярных в компьютерной лингвистике мер качества языковых моделей является перплексия. Применительно к модели PLSA она принимает вид:

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

По сути формула представляет собой усреднённый по всем токенам коллекции логарифм правдоподобия, от которого затем взята обратная экспонента.

1.2 Аддитивная регуляризация тематических моделей

Исходная модель PLSA обладает неединственностью решения, так как для матриц Φ, Θ можно подобрать стохастическую матрицу S ранга $|T|$ и объявить матрицы $\Phi' = \Phi S$, $\Theta' = S^{-1}\Theta$ новым решением. Это некорректно поставленная задача, решение которой можно доопределить путём добавления регуляризаторов к исходному функционалу [10]:

$$\mathcal{L}(\Phi, \Theta) = \sum_{d,w} n_{dw} \ln \sum_{t \in T} \varphi_{wt}\theta_{td} + \sum_{i=1}^{k} \tau_i R_i(\Phi, \Theta) \to \max_{\Phi,\Theta},$$

где τ_i — коэффициенты регуляризации.

В литературе в основном решают проблему неединственности решения PLSA с помощью введения априорных распределений на матрицы Φ и Θ и используются техники приближённого байесовского вывода для обучения моделей. Модель называется LDA (Latent Dirichlet Allocation) [1]. Большинство известных тематических моделей, в том числе LDA, допускают переформулировку в терминах аддитивной регуляризации, и могут обучаться посредством максимизации регуляризованного правдоподобия.

1.3 Интерпретируемость тем

Недостатком рассмотренных тематических моделей является то, что они основаны на гипотезе мешка слов и никак не учитывают порядок токенов в документах. Согласно гипотезе дистрибутивности [13], токены близки семантически, если они 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 m наиболее вероятных токенов некоторой темы (обычно $m = 10$) и определяет, насколько они семантически связаны друг с другом. Затем по некоторой шкале выставляет оценку интерпретируемости темы [7, 8].

1.4 Когерентность тем

В [8] сравнивалось 15 автоматически вычисляемых мер интерпретируемости тем по их корреляции с оценками, выставленными экспертами (в смысле корреляции Спирмена). Лучшей мерой интерпретируемости оказалась когерентность — средняя PMI

(Pointwise Mutual Information) [4] по всем парам кроме пар повторяющихся токенов среди m наиболее вероятных токенов темы:

$$C_t = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{PMI}(w_i, w_j)$$

$$\mathrm{PMI}(w_i, w_j) = \log \frac{\mathsf{P}(w_i \cap w_j)}{\mathsf{P}(w_i)\,\mathsf{P}(w_j)},$$

где w_i и w_j — события, заключающиеся в наблюдении токенов w_i и w_j соответственно. Также в [2, 5] было показано, что использование Positive PMI вместо PMI и Shifted Positive PMI улучшает качество в задачах семантической близости слов.

$$\mathrm{PPMI}(w_i, w_j) = \max(0, \mathrm{PMI}(w_i, w_j)),$$

$$\mathrm{SPPMI}_k(w_i, w_j) = \max(0, \mathrm{PMI}(w_i, w_j) - \ln k),$$

Если события w_i и w_j независимы, то вероятность пересечения событий факторизуется в произведение вероятностей по отдельным событиям, и PPMI равна 0. То есть значения больше 0 получают пары, которые встречаются вместе чаще, чем если бы их появление было независимо. Идея Shifted PPMI заключается в отбрасывании пар, которые встречаются вместе лишь немного чаще, чем случайно. Такие пары неинтересны, и за счёт их удаления можно повысить эффективность вычисления когерентности и уменьшить затраты по памяти.

Ясно, что понятие когерентности определено неоднозначно, так как вероятность совместно наблюдать пару токенов можно рассчитывать по-разному в зависимости от модели. Далее рассматриваются некоторые модели со-встречаемостей токенов.

1.5 Модели со-встречаемостей токенов

1.5.1 PPMI, основанная на частотах пар токенов

Рассмотрим в документе d множество позиций токенов. Скажем, что пара позиций $(i,\ j)$ в одном и том же документе 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} — счётчик со-встречаемости токенов u и v, это 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:

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

Заметим, что $n_{uv} = n_{vu}$.
Введём вероятность на множестве пар токенов:

$$\mathsf{P}(u, v) \propto n_{uv}.$$

$\mathsf{P}(u, v)$ — совместная вероятность наблюдения пары токенов (u, v), она же вероятность пересечения событий — наблюдение токена u и наблюдение токена v. Обозначив через $n_u = \sum\limits_v n_{uv}$ и через $n = \sum\limits_u n_u$, получаем:

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

$\mathsf{P}(u)$ примерно равна доле токена u среди всех токенов коллекции, равенство не точное за счёт краевых эффектов на границах документов. Итоговая формула для PPMI:

$$\mathrm{PPMI}(u, v) = \left[\ln \frac{\mathsf{P}(u, v)}{\mathsf{P}(u)\,\mathsf{P}(v)} \right]_+ = \max\left(0, \ln \left[\frac{n_{uv} \cdot n}{n_u \cdot n_v}\right]\right).$$

Данная модель с небольшими отличиями использовалась в [2, 5, 6].

1.5.2 PPMI, основанная на частотах документов

Ещё одиним популярным способом задания вероятностей для PPMI является учёт количества документов, в которых фигурировала хотя бы раз данная пара токенов. В оригинальное статье [7] не использовались окна фиксированной ширины, а учитывались попадание пары токенов в один документ. Этот способ можно обобщить введением окон фиксированной ширины по аналогии с предыдущей моделью, основанной на частотах пар токенов.

Введём величину n_{uv}, которая равна количеству документов, в которых встретились токены u и v хотя бы раз внутри некоторого окна ширины k:

$$n_{uv} = \left| \left\{ d \in D \,\middle|\, \exists\, (i, j) : w_{di} = u,\, w_{dj} = v,\, 0 < |i - j| \leqslant k \right\} \right|.$$

Элементарным исходом является документ. На множестве документов введём равномерное распределение. Тогда вероятность совместно встретить токены u и v внутри некоторого окна в коллекции будет равна доле документов с данным свойством, а, соответственно, вероятность появления токена будет также выражаться в терминах документов — доля документов, в которых встречался данный токен хотя бы раз:

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

То есть количество документов с заданным свойством нормируется на общее количество документов коллекции. Тогда PPMI можно переписать следующим образом:

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

1.6 KL-регуляризатор и регуляризатор когерентности

В [7] была предложена процедура оптимизации когерентности. Идеи, описанные в данной статье, были заложены в основу регуляризатора когерентности в [10]. Процедура оптимизации из [7] может быть приближённо сформулирована в виде следующего регуляризатора:

$$R_{doc\_coher}(\Phi) = \sum_{t \in T} \sum_{(u,v) \in W^2} \frac{N_{uv}}{N_v} n_{vt} \ln \varphi_{vt},$$

где N_{uv} — количество документов, в которых встретились вместе токены u и v (не в окне, а в целом документе), N_u — количество документов, в которых встречается токен u, n_{vt} — количество раз, сколько токен v был связан с темой t в коллекции. Данный регуляризатор является частным случаем более общего:

$$R(\Phi) = -\sum_{t \in T} n_t \, \text{KL}_u\big(\hat{\mathsf{P}}(u|t) \;||\; \varphi_{ut}\big) = -\sum_{t \in T} n_t \sum_{u \in W} \hat{\mathsf{P}}(u|t) \ln \frac{\hat{\mathsf{P}}(u|t)}{\varphi_{ut}}.$$

Регуляризатор минимизирует взвешенную по счётчикам тем n_t KL-дивергенцию между некоторыми оценками вероятности токена при условии темы и столбцами матрицы Φ — параметрами модели. Оценки можно вычислять, задав по формуле полной вероятности оценки $\hat{\mathsf{P}}(u|v)$.

$$\arg\max_{\Phi} R(\Phi) = \arg\max_{\Phi} \left[\sum_{t \in T} n_t \sum_{u \in W} \hat{\mathsf{P}}(u|t) \ln \varphi_{ut} \right] =$$

$$= \left\{ \hat{\mathsf{P}}(u|t) = \sum_{v \in W} \mathsf{P}_{doc}(u|v,t)\, \mathsf{P}(v|t), \;\; \mathsf{P}_{doc}(u|v,t) = \mathsf{P}_{doc}(u|v) \right\} =$$

$$= \arg\max_{\Phi} \left[\sum_{t \in T} n_t \sum_{(u,v) \in W^2} \mathsf{P}_{doc}(u|v)\, \mathsf{P}(v|t) \ln \varphi_{ut} \right] =$$

$$= \arg\max_{\Phi} \left[\sum_{t \in T} n_t \sum_{(u,v) \in W^2} \mathsf{P}_{doc}(u|v)\, \frac{n_{vt}}{n_t}\, \ln \varphi_{ut} \right] =$$

$$= \arg\max_{\Phi} \left[\sum_{t \in T} \sum_{(u,v) \in W^2} \mathsf{P}_{doc}(u|v)\, n_{vt}\, \ln \varphi_{ut} \right] = \left\{ \mathsf{P}_{doc}(u|v) = \frac{N_{uv}}{N_v} \right\} =$$

$$= \arg\max_{\Phi} \left[\sum_{t \in T} \sum_{(u,v) \in W^2} \frac{N_{uv}}{N_v}\, n_{vt}\, \ln \varphi_{ut} \right]$$

Ничего не мешает заменить $\mathsf{P}_{doc}(u|v)$ на условные вероятности другой вероятностной модели, главное — определить в соответствующей модели счётчики n_{uv}:

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

Использование в качестве n_{uv} счётчиков совместной встречаемости, основанных на частотах пар токенов или на частотах пар документов, в наших экспериментах не приводит к улучшению когерентности, посчитанной как усреднённая PPMI.

Можно немного поменять регуляризатор, заменив $\mathsf{P}(u|v)$ на произвольную функцию на парах токенов $f(u,v)$. Регуляризатор таким образом теряет вероятностную интерпретацию, однако появляется возможность оптимизировать именно когерентность, так как функционал $R(\Phi)$ теперь монотонно зависит от значений $\mathrm{PPMI}(u,v)$.

Далее в тексте будем называть **KL-регуляризатором** функционал

$$R(\Phi) = -\sum_{t \in T} n_t \; \mathrm{KL}_u\big(\hat{\mathsf{P}}(u|t) \;||\; \varphi_{ut}\big),$$

регуляризатором когерентности — функционал

$$R(\Phi) = \tau \sum_{t \in T} \sum_{(u,v) \in W^2} \mathrm{PPMI}(u,v)\, n_{vt}\, \ln \varphi_{ut},$$

который является частным случаем **обобщённого регуляризатора когерентности**:

$$R(\Phi) = \tau \sum_{t \in T} \sum_{(u,v) \in W^2} f(u,v)\, n_{vt}\, \ln \varphi_{ut}.$$

2 Алгоритм подсчёта со-встречаемостей

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

- асинхронная обработка входной коллекции и промежуточных данных;
- возможность обработки коллекций потенциально неограниченных по числу документов;
- возможность улучшения производительности за счёт увеличения числа потоков-обработчиков на одном вычислительном узле.

Условно алгоритм можно разбить на три этапа:

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

2.1 Обработка входной коллекции

Обработка входной коллекции происходит по пакетам (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. Предполагается, что размер каждого документа ограничен общей константой, иначе нельзя было бы гарантировать, что каждый документ можно считать в оперативную память. 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. $vocab$ может представлять собой словарь коллекции W, очищенный от стоп-слов и низкочастотных токенов, или он может состоять из токенов некоторого узкого набора тем. Для больших коллекций рекомендуется задавать $vocab$. Поскольку множество всех токенов может оказаться большим — около миллиона токенов или больше, и на хранение статистики по каждой паре в пакете будет расходоваться много памяти. Таким образом, если задать большой $vocab$ или не задавать его вовсе, не получится использовать большие пакеты, так как при этом не хватило бы памяти для хранения большого количества различных пар токенов, встретившихся в пакете. Статистика со-встречаемостей по каждому пакету сохраняется в файл в отсортированном формате

для того, чтобы потом можно было легко объединить файлы, соответствующие всем различным пакетам, в один. Также для вычисления PPMI нужно значение n — общее количество рассмотренных пар в коллекции. Это значение можно вычислить на первом этапе алгоритма. Обработка пакетов происходит параллельно асинхронно в n_{jobs} потоков по одному пакету на поток. Для каждого документа подсчитываются частоты со-встречаемостей пар токенов внутри некоторого окна ширины $window\_width$, которая является параметром алгоритма. Если некоторое окно выходит за границы документа, оно усекается до соответствующих границ.

Этап 1. Обработка входной коллекции.

Вход: коллекция D, $vocab$, $batch\_size$, $window\_width$, 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 $j = 1, \ldots, window\_width : \ i + j \leqslant n_d$
          $r = i + j$;
          $C[w_{di}, w_{dr}] \leftarrow C[w_{di}, w_{dr}] + 1$;
          $C[w_{dr}, w_{di}] \leftarrow C[w_{dr}, w_{di}] + 1$;
          $n \leftarrow n + 2$
    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: вначале по ключу (токену u) находятся все токены, с которыми он встречался в коллекции внутри некоторого окна, 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 пары, а затем все, с которыми он встретился и значение со-встречаемости. Красно-чёрное дерево было выбрано, так как оно позволяет осуществлять быстрый доступ к данным и поддерживает их отсортированными.

2.2 Агрегация собранной статистики

Основой второго этапа является сортировка во внешней памяти с помощью алгоритма k-way merge [1] . Алгоритм в случае $k = 2$ совпадает со слиянием 2 отсортированных массивов, а в случае $k > 2$ строит бинарную кучу 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 пакете, 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 одну запись для каждого файла. Так как число файлов потенциально неограничено, 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 стандартного k-way merge до тех пор, пока количество массивов не станет меньше заданной константы. Ситуация, когда количество файлов слишком велико и происходит предварительное слияние некоторых порций файлов, на практике 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 максимального количества открытых файлов в одном процессе. Эта константа определяется в ядре операционной системы. Если бы количество файлов превышало эта константу, пришлось бы производить открытие-закрытие файлов, что сильно сильно замедлило бы работу алгоритма, так системные вызовы обрабатываются долго.

Все слияния можно производить многопоточно: каждый поток будет иметь свой локальный набор файлов. Многопоточное слияние нужно прекратить по достижении какого-то критически малого количества файлов и произвести окончательное однопоточное слияние оставшихся 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 на третьем этапе алгоритма. Также для экономии внешней памяти по завершении слияния некоторой порции файлов и записи объединённых данных в новый, предыдущие файлы можно сразу удалить.

[1] `https://en.wikipedia.org/wiki/K-way_merge_algorithm`

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;

пока $|F| > min\_merge$
  $F' \leftarrow \{\}$;
  пока $|F| > 0$
    для всех $i = 1, \ldots, n\_jobs$
      $batch\_size \leftarrow \min(\lfloor \frac{max\_open}{n_{jobs}} \rfloor, |F|)$;
      $batch \leftarrow F_1, \ldots, F_{batch\_size}$;
      $F \leftarrow F \setminus batch$;
      $f \leftarrow k$-way merge$(batch)$;
      Добавить f в F';
  $F \leftarrow F'$
$f \leftarrow k$-way merge(F, min_cooc);

2.3 Вычисление метрик

Последний этап алгоритма заключается в вычислении метрик по имеющейся статистике 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 $(u, v, n_{uv}) \in f$
  $ratio \leftarrow \frac{n_{uv} n}{n_u n_v}$;
  если $ratio > 1$ **то**
    PPMI $\leftarrow \log(ratio)$;
    $f' \leftarrow$ PPMI;

2.4 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) оперативной памяти хватает для хранения словаря *vocab*;

(2) любой документ целиком можно записать в оперативную память;

(3) оперативной памяти хватает для хранения записей из промежуточных файлов (по одной записи на каждый открытый файл);

(4) внешней памяти хватает для хранения всех промежуточных файлов;

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

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

Достаточно выполнения ограничений (1 — 4) и любого из (5), (6).

2.5 Анализ сложности

2.5.1 Анализ первого этапа

В самом начале алгоритма в память записывается словарь *vocab* для отслеживания релевантных токенов. Также каждому токену из vocab сопоставляется число и после обработки коллекции работа происходит не с токенами, а с числами для эффективного хранения статистики со-встречаемостей. Отображение токенов в числа и обратно хранятся в хеш-таблице и в массиве соответственно. Построение этих структур занимает $\mathcal{O}(|vocab|)$ времени, а обращение к данным по ключу — $\mathcal{O}(1)$. Дальнейшая часть первого первого этапа алгоритма работает за линейное время от общей длины коллекции и ширины окна, причём эти параметры входят как множители в оценку, так как для почти всех токенов коллекции (за исключением первых и последних $window\_width$ токенов в каждом документе) необходимо просмотреть $window\_width$ 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 $\mathcal{O}\big(N \cdot window\_width \cdot \log(n_{pairs})\big)$, где N — длина коллекции в токенах, а n_{pairs} — верхняя оценка на количество пар в красно-чёрном дереве. Так как единовременно в память пишется $batch\_size$ документов, можно утверждать $n_{pairs} = window\_width \cdot \sum_{i=1}^{batch\_size} n_d = \{\text{ограничение (4)}\} = const$. С другой стороны, можно получить другую верхнюю оценку $\hat{n}_{pairs} = |vocab|^2 = \{\text{ограничение (5)}\} = const$. В итоге время оценивается как $\mathcal{O}\big(|vocab| + N \cdot window\_width\big)$. Можно также оценивать через $\mathcal{O}\big(|vocab| + |D| \cdot window\_width\big)$, так как размеры всех документов ограничены одной константой.

Оперативная память на первом этапе тратится на содержание пакета документов и красно-чёрного дерева. Отсюда условия (2, 5, 6). Также так как в самом начале этапа в память был записан словарь, создана хеш-таблица и массив обратных индексов, памяти требуется как минимум $\mathcal{O}\big(|vocab|\big)$. При выполнении условий (5) или (6) можно гарантировать, что затраты по памяти будут $\mathcal{O}\big(|vocab| + n_{jobs} \cdot batch\_size\big)$, так как $\mathcal{O}\big(|vocab|\big)$ тратится на содержание $vocab$, а $\mathcal{O}\big(n_{jobs} \cdot batch\_size\big)$ — на хранение документов в памяти, а расходы на хранение красно-чёрного дерева занимают константу. Выполнения условия (5) 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.

Суммарный объём файлов на внешнем устройстве можно оценить как $\mathcal{O}\big(|D|\big)$, так как размер каждого документа заранее ограничен.

2.5.2 Анализ второго этапа

В стандартном k-way merge время доступа к минимальному элементу есть $\mathcal{O}\big(1\big)$, 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 $\mathcal{O}\big(N \cdot \log k \cdot |vocab|\big)$, так как для слияния может понадобиться $\mathcal{O}\big(|vocab|\big)$ операций. Здесь N — верхняя оценка на количество записей в файлах. В данном случае записью является последовательность из пар с одинаковым первым токеном. Так как в данном алгоритме используется модификация k-way merge, оценки сложности немного отличаются от стандартного случая. На первом проходе алгоритма всё множество файлов разбивается на непересекающиеся части, которые обрабатываются параллельно. Всего частей $\frac{|D| \cdot n_{jobs}}{batch\_size \cdot max\_open} + \mathcal{O}(1)$, и n_{jobs} частей можно обрабатывать одновременно. То есть время работы на первом этапе оценивается как $\mathcal{O}\big(\frac{|D| \cdot n_{jobs}}{batch\_size \cdot max\_open} \cdot \frac{1}{n_{jobs}} \cdot T\big)$, где T — время обработки одной части, которое оценивается для стандартного k-way merge как $\mathcal{O}\big(|vocab| \cdot \log \frac{max\_open}{n_{jobs}} \cdot |vocab|\big)$, и итоговая сложность на первом этапе равна $\mathcal{O}\big(\frac{|D| \cdot |vocab|^2}{batch\_size}\big)$. Остальные переменные в формуле оцениваются сверху константой. Всего проходов по файлам до достижения минимального количества происходит $\lceil \log_{max\_open} \frac{|F|}{min\_merge} \rceil$, поэтому общее время работы можно оценить как $\mathcal{O}\big(\frac{|D| \cdot \log |D| \cdot |vocab|^2}{batch\_size}\big)$.

На втором этапе требуется хранить для каждого открытого файла по одной записи из файла, что потенциально может занимать $\mathcal{O}\big(|vocab| \cdot max\_open\big)$ оперативной памяти. Здесь необходимо воспользоваться условием (3).

Так как на втором этапе файлы объединяются посредством слияния, а слияние может происходить одновременно только по max_open файлам, можно для дополнительно затраченной внешней памяти ввести оценку $\mathcal{O}\big(|vocab|^2 \cdot max\_open\big)$, так как каждый новый файл будет содержать не более $|vocab|^2$ записей константной длины. Также справедлива оценка с первого этапа: $\mathcal{O}\big(|D|\big)$, так как после слияния суммарный размер полученных дополнительных файлов не будет превышать суммарного размера исходных файлов.

2.5.3 Анализ третьего этапа

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

2.5.4 Выводы и возможные модификации

Алгоритм имеет время работы $\mathcal{O}\big(|D| \cdot log|D|\big)$ от размера коллекции и $\mathcal{O}\big(|vocab|^2\big)$ от размера $vocab$ и требует $\mathcal{O}\big(1\big)$ оперативной памяти, а также $\mathcal{O}\big(|D|\big)$ внешней памяти, что делает возможный обработку больших коллекций при соблюдении условий (1 — 4) и одного из (5) или (6). За счёт того, что внешней памяти требуется не константа, алгоритм имеет ограничение на объём обрабатываемых коллекций, однако можно его модифицировать: применять к коллекции некоторого размера по частям, получать словари со-встречаемостей на больших частях коллекции и потом делать слияние получившихся файлов со-встречаемостей, тогда размер потребляемой внешней памяти не будет зависеть от длины коллекции, а будет зависит от размера $vocab$ как $\mathcal{O}\big(|vocab|^2\big)$

2.6 Реализация алгоритма

Описанный выше алгоритм был реализован в библиотеке тематического моделирования с открытым кодом BigARTM [9] [2] [3] [4] на языке C++. Язык был выбран из соображений эффективности и простоты использования стандартных структур данных. Например, в качестве красно-чёрного дерева использовался контейнер std::map.

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

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

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

3 Влияние параметров алгоритма

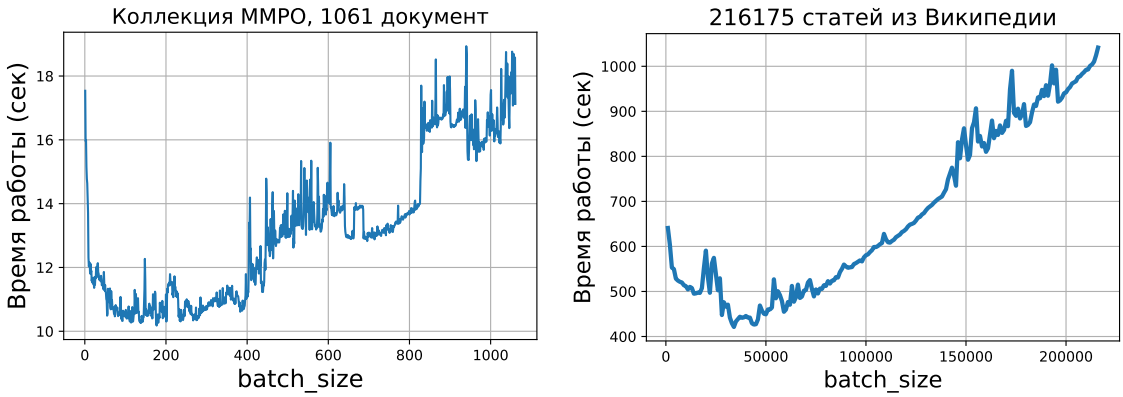
В данном разделе приводятся результаты экспериментов по измерению потребления ресурсов при различных значениях параметров. Все эксперименты проводились на ноутбуке с SSD и 8-ядерным процессором 2,3 GHz Intel Core i5.

3.1 Параметр batch_size

Некоторые пары токенов встречаются в большом количестве документов, поэтому если пара (i, j) встретилась в разных пакетах, она будет записана в разные файлы несколько раз. Чтобы избежать частого 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.

3.1.1 Реальное время работы при разных значениях batch_size

Проводилось несколько экспериментов по замеру реального времени работы алгоритма на коллекциях малого, среднего и большого размеров. В качестве малой коллекций была взята коллекция записей выступлений на конференции ММРО (1069 документов, 7805 уникальных токенов и 804423 токенов всего), а в качестве коллекции среднего размера — набор статей из русскоязычной и англоязычной Википедии (216175 документов, 196749 уникальных токенов и 70536525 токенов всего). Словари $vocab$ обеих коллекций совпадали с полными словарями коллекций W, а параметр ширины окна $window\_width = 10$. Для определения зависимости времени работы от параметра $batch\_size$ алгоритм запускался для каждого значения от 1 до $|D|$ включительно с шагом 1 для маленькой коллекции и с шагом 1000 — для большой. Полученные графики времени работы представлены ниже. Для статей из Википедии на графике указаны замеры для значений $batch\_size$, начинающихся с 1001, так как для $batch\_size = 1$ время работы около 2000 секунд.



Видно, что на заключительном участке время растёт, то есть эффект от параллелизма исчезает, а время обработки коллекции становится временем обработки самого большого пакета. На начальном участке время работы становится меньше с увеличением размера пакета, а глобальный минимум достигается на коллекции ММРО при значении $batch\_size =$ и на коллекции статей из русскоязычной и англоязычной Википедии при $batch\_size = 176$, то есть когда $batch\_size \approx \frac{|D|}{8}$, что согласуется с ожиданиями, так как продолжение наращивания $batch\_size$ ведёт к тому, что первый этап алгоритма работает дольше. Также эксперименты проводились при параметре $n_{jobs} \in [1, 8]$, результаты такие же: довольно хорошим приближением к оптимальному значению $batch\_size$ будет значение $\frac{|D|}{n_{jobs}}$. Соответственно рекомендация состоит в том, чтобы выбирать настолько большое значение, которое позволяет оперативная память, но не больше, чем $\frac{|D|}{n_{jobs}}$. Значения больше, чем $\frac{|D|}{n_{jobs}}$ есть смысл выставлять, только если есть жёсткие ограничения на объём доступного места на внешнем носителе.

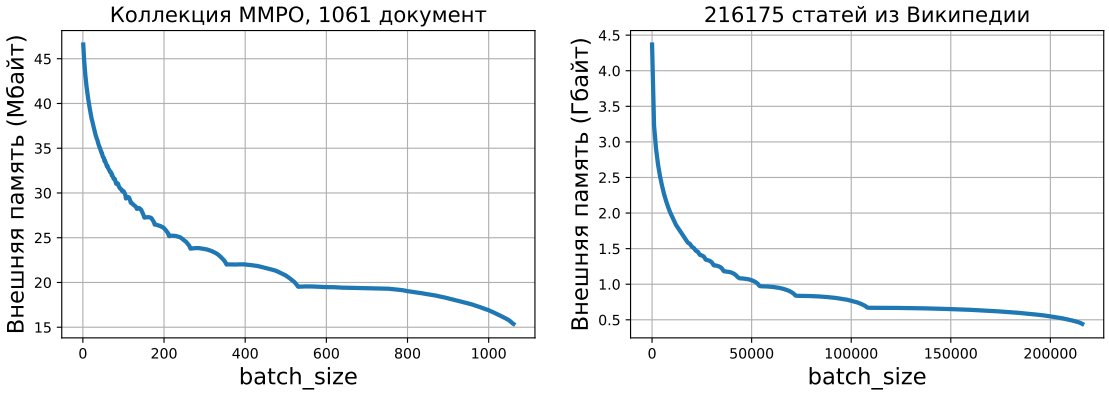
Также было проведено 2 эксперимента с большими коллекциями:

- 8446835 статей англоязычной Википедии, 8272855 уникальных токенов, 3832966193 токенов всего, $window\_width = 10$, $batch\_size = 2500$
- посты социальных сети (4528512 документов), 87494 уникальных токенов, 1613807215 токенов всего, $window\_width = 10$, $batch\_size = 10000$

При значениях остальных параметров как в предыдущих экспериментах время работы составило **4 часа 18 минут** на Википедии и **2 часа 45 минут** на постах социальных сетей. Оперативной памяти было затрачено во всех экспериментах не более **16 GB**.

3.1.2 Затраченная внешняя память при разных значениях batch_size

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 ММРО составляет примерно 14.2 MB, а коллекция статей из русскоязычной и англоязычной Википедии занимает примерно 795.6 MB.



Если $vocab = W$, то при запуске на больших коллекциях затраченная на внешнем носителе память может сильно превышать размер исходной коллекции (как в эксперименте со статьями из Википедии, результаты ниже), поэтому рекомендуется урезать *vocab* посредством, к примеру, удаления низкочастотных токенов.

В экспериментах с большими коллекциями алгоритм запускался с теми же параметрами, как и эксперименте по замеру времени. Статьи Википедии занимают около **21.94 GB**, а при работе алгоритма промежуточные файлы занимают около **131.17 GB**. Коллекция постов социальных сетей занимала исходно около **27.9 GB**, а промежуточные файлы — около **59.2 GB**. Оперативной памяти во всех экспериментах было затрачено не больше **16 GB**.

3.2 Параметр min_merge и константа max_open

Параметр *min_merge* отвечает за то минимальное количество файлов, при котором слияние производится параллельно. Если количество файлов меньше данного, слияние производится последовательно. Выбор оптимального значения параметра *min_merge* неочевиден: с одной стороны использование вычислительных мощностей может помочь уменьшить время слияния файлов, с другой стороны, файлы будут больше раз записываться во внешнюю память и считываться из неё. Здесь всё зависит от скорости записи и чтения с внешнего носителя, от общего объёма данных,

количества ядер процессора. Также наши эксперименты показывают, что на больший коллекциях есть смысл выставлять этот параметр максимально возможным, поэтому в реализации в BigARTM этот параметр зафиксирован. Значение параметра min_merge не является фактически максимальным, которое позволяет операционная система, но это некоторая нижняя оценка на максимальное значение. Само максимальное значение не используется, так как процесс при этом может обрабатывать другие файлы.

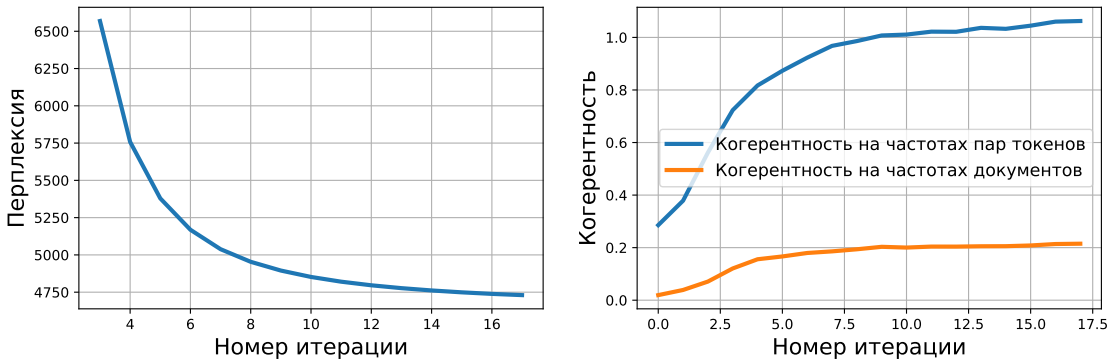
3.3 Выводы по алгоритму

Алгоритм можно запускать на ноутбуке и за приемлемое время получать 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. Также использование Shifted PPMI может уменьшить потребляемую внешнюю память на хранение итогового словаря, а методы, использующие этот словарь в дальнейшем могут более эффективно работать.

4 Увеличение когерентности

4.1 Обучение модели PLSA

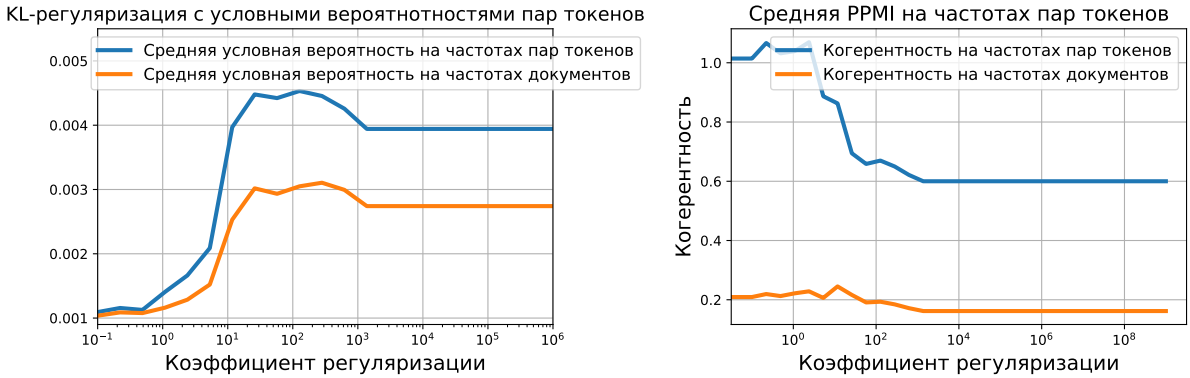
Была обучена модель PLSA на коллекции новостных статей «NY Times». В коллекции 300000 документов. Была произведена предварительная токенизация, лемматизация и удаление стоп-слов. С помощью описанного выше алгоритма была предварительно собрана статистика со-встречаемостей на англоязычной Википедии и посчитаны PPMI. Обучение проводилось с помощью библиотеки BigARTM. Количество тем было взято равным 25. Это значение было взято произвольно, точное количество «тем» в коллекции неизвестно. После каждой итерации EM-алгоритма замерялась средняя по всем темам когерентность и перплексия модели. Ниже представлены графики перплексии и двух видов когерентности (основанная на частотах пар токенов и на частотах документов соответственно) после каждой итерации EM-алгоритма. Когерентность считалась по 10 наиболее вероятным токенам каждой темы.

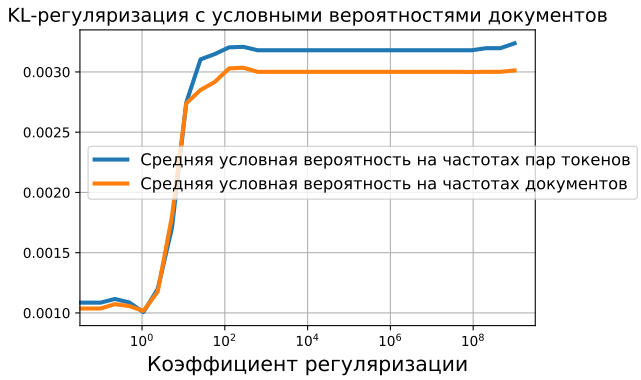


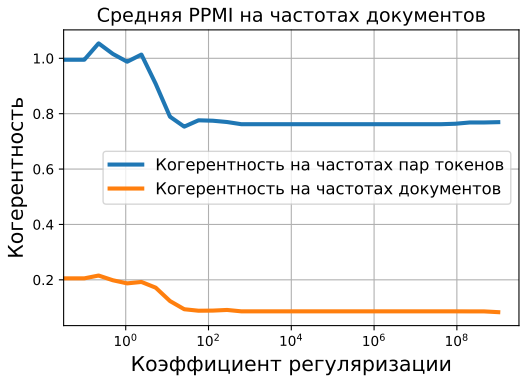
Видно, что оба вида когерентности растут по мере увеличения правдоподобия в процессе обучения.

4.2 Использование KL-регуляризатора с условными вероятностями

Как уже было сказано в разделе про регуляризатор когерентности, использование без изменений регуляризатора [10] (в котором $f(u,v) = \hat{\mathsf{P}}(u|v)$) не улучшает когерентность как среднюю PPMI. Покажем это экспериментально на коллекции «NY Times» с теми же значениями гиперпараметров, как и в разделе про обучение модели PLSA: 25 тем, когерентность считается по 10 наиболее вероятным токенам каждой темы. Можно рассчитывать когерентность двумя способами, так как есть два способа вычислить PPMI. Ещё можно подавать вместо $f(u,v)$ условные вероятности, посчитанные двумя способами. Ниже представлены графики описанных величин в результате обучения в зависимости от коэффициента регуляризации. Значения коэффициента регуляризации перебирались по логарифмической шкале.





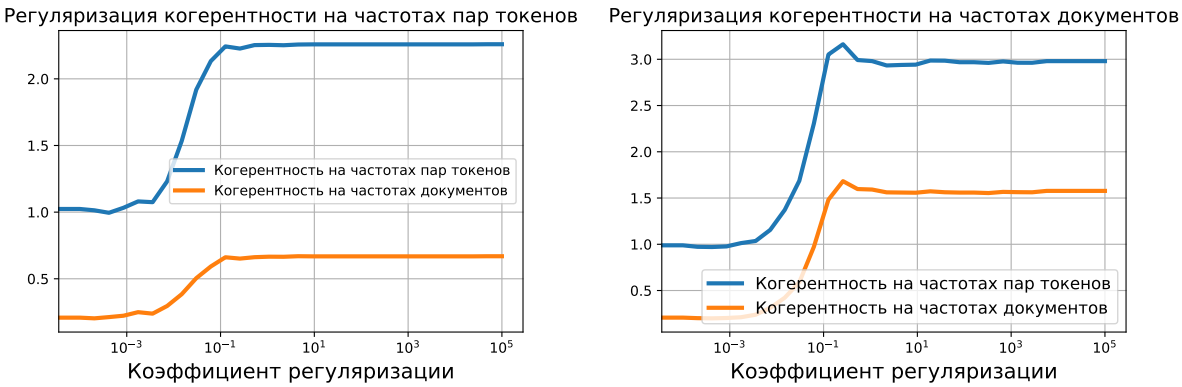


При этом перплексия в результате обучения в зависимости от коэффициента регуляризации во всех случаях выглядит как на следующем графике.

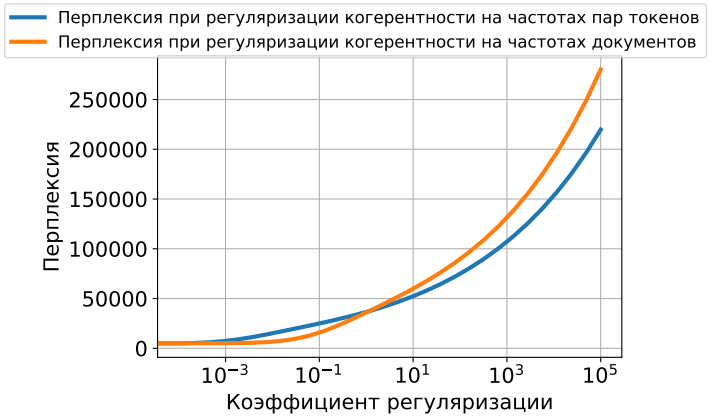
Вывод: для того, чтобы увеличивать когерентность в смысле средней по парам токенов тем PPMI, нужно подавать в функцию $f(u, v)$ значения PPMI, а если вместо $f(u, v)$ подавать $\hat{\mathsf{P}}(u|v)$, то будет увеличиваться величина, похожая на когерентность, но рассчитанная по-другому — $\hat{\mathsf{P}}(u|v)$ вместо PPMI. Также эта величина не всегда возрастает, начиная с некоторого значения, падает, а потом выходит на асимптоту.

4.3 Регуляризация когерентности

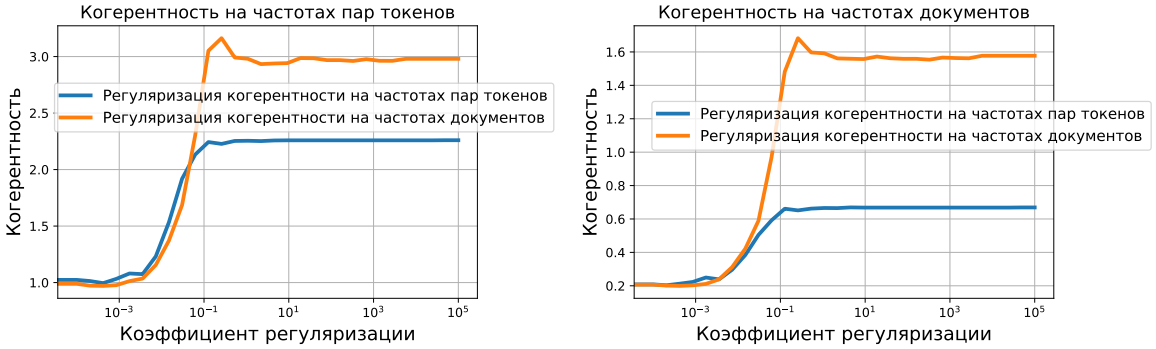
В регуляризаторе когерентности для максимизации средней PPMI используются заранее посчитанные значения PPMI для пар токенов. Итоговый функционал для регуляризатора представляет собой средне-взвешенную PPMI. Ниже представлены графики когерентности в результате обучения в зависимости от коэффициента регуляризации. Значения коэффициента регуляризации перебирались по логарифмической шкале.



Таким образом для регуляризации когерентности нужно использовать PPMI в качестве функции $f(u, v)$. Также если использовать когерентность на частотах документов, можно получать потенциально более высокие значения когерентности обоих типов. При этом перплексия моделей растёт экспоненциально при увеличении коэффициента регуляризации.



Ниже сравниваются способы получения наиболее высоких значений когерентности: регуляризация когерентности на частотах пар токенов и регуляризация когерентности на частотах документов.



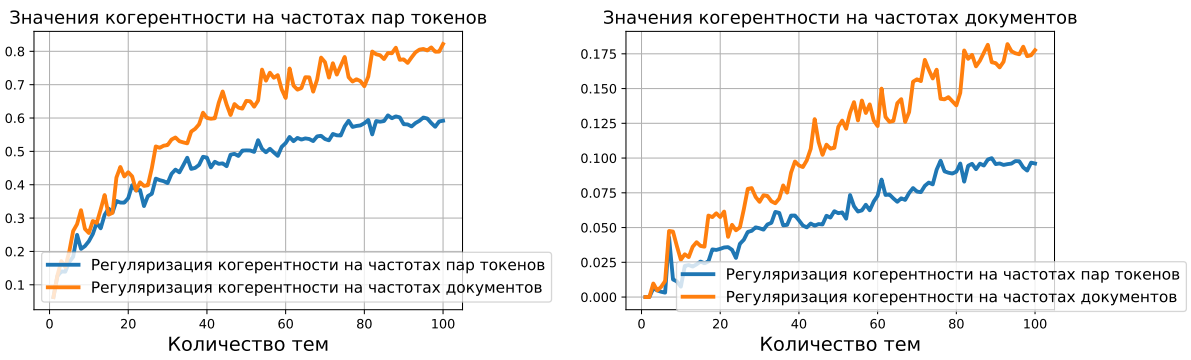
При низких значениях коэффициента регуляризации величины взаимозаменяемы, а при высоких — выгоднее регуляризовывать когерентность на частотах документов.

С другой стороны, при высоких значениях коэффициента регуляризации ухудшается перплексия и данный результат выглядит бесполезным, однако если удастся каким-то образом значительно уменьшить перплексию модели, то более выгодно всё же окажется использовать именно когерентность на частотах документов, а когерентность на частотах пар можно даже не вычислять. К тому же из этих двух видов когерентности последний эффективнее рассчитывается.

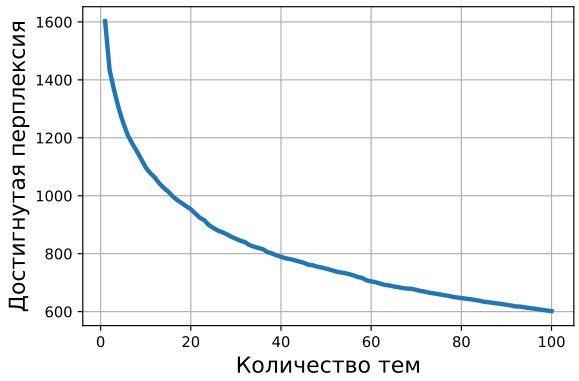
4.4 Зависимость от количества тем

В предыдущих экспериментах удавалось улучшить когерентность за счёт роста перплексии, причём она росла очень быстро. Оказывается, с выбором избыточного количества тем можно найти отрезок для коэффициентов регуляризации такой, что перплексия будет несущественно выше соответствующего значения без регуляризации. Зафиксируем уровень 10 %, скажем, что несущественно выше значит, что перплексия поднимется не больше, чем на 10 % от исходного значения без регуляризации. Значения коэффициента регуляризации перебирались по логарифмической шкале на отрезке $[10^{-4}; 10^{5}]$, бралось 30 значений.

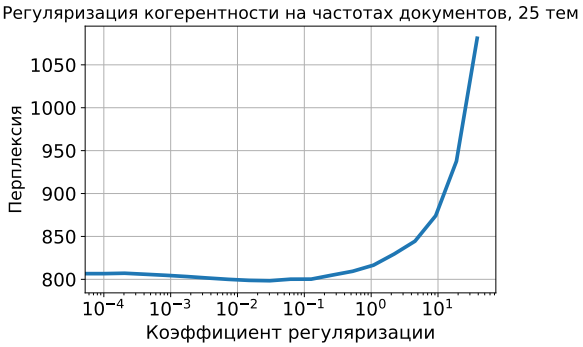
Следующий эксперимент проводился на коллекции выступлений на конференции «ММРО». Проводились замеры двух типов когерентности на правых границах описанного выше отрезка при значении количества тем на отрезке [1; 100]. После каждого прохода по коллекции коэффициент регуляризации уменьшался посредством умножения на множитель $\lambda < 1$, в наших экспериментах $\lambda = 0.6$. Затухание коэффициента регуляризации было введено для того, чтобы была возможность восстановить перплексию модели после регуляризации. В случае $\lambda = 1$ получается очень быстрое возрастание перплексии и отрезок приемлемых значений коэффициента регуляризации становится очень узким, что неудобно при подборе коэффициента. Такой способ уменьшения коэффициента регуляризации похож на метод Тихонова по сведению некорректно поставленных задач к корректно поставленным.



Ниже приведен график максимально достигнутой перплексии (не больше, чем 110 % от исходной перплексии, которая была достигнута без регуляризации). График не зависит от способа регуляризации, так как значения вычисляются в зависимости от значений перплексии без регуляризации.



Перплексия при применении регуляризации может даже падать относительно исходного значения, полученного без регуляризации. Ниже приведён график логарифма перплексии в зависимости от коэффициента регуляризации при количестве тем, равном 25.



Как можно увидеть, даже при малых значениях коэффициента регуляризации, когерентность на частотах документов выгоднее (в смысле когерентности и в смысле перплексии) регуляризовывать, чем когерентность на частотах пар токенов. При этом, если проводить регуляризацию на частотах документов, когерентность можно увеличить в несколько раз по сравнению с вариантом без регуляризации. Ниже приведены графики, полученные в результате регуляризации когерентности на частотах документов.

5 Результаты и выводы

Было предложено два способа вычисления и регуляризации когерентности: когерентность, основанная на частотах пар токенов и основанная на частотах документов. Эксперименты показали, что выгоднее с точки зрения перплексии, когерентности и вычислительной эффективности проводить регуляризацию на частотах документов.

Предложенный метод регуляризации позволяет существенно увеличивать когерентность тематических моделей без сильного увеличения перплексии при правильном подборе коэффициента регуляризации, количества тем и экспоненциальном убывании коэффициента регуляризации. Если брать коэффициент регуляризации слишком большим, перплексия модели становится очень большой, темы при этом могут быть когерентными, но они уже плохо описывают исходную коллекцию. При увеличении числа тем когерентность растёт, а перплексия — падает.

Предложен алгоритм эффективного подсчёта статистики совместной встречаемости токенов в больших текстов коллекциях. Было показано, что его можно использовать для коллекций типа англоязычной Википедии, и расчёт можно проводить на ноутбуке за приемлемое время и с небольшим расходом внешней и оперативной памяти. Статистика со-встречаемости может быть вычислена до проведения экспериментов по регуляризации или вычислению когерентности тематических моделей. Также когерентность можно использовать для построения моделей битермов [11] и модели WNTM (Word Network Topic Model) [12].

Список литературы

[1] Blei, D.M., Ng,A.Y., Jordan,M.I.: Latent dirichlet allocation. the Journal of machine Learning research 3, 993–1022 (2003)

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

[3] Hofmann, T.: Probabilistic latent semantic analysis. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence. pp. 289–296. UAI'99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)

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

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

[6] 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.

[7] Mimno, D., Wallach, H.M., Talley, E., Leenders, M., McCallum, A.: Optimizing seman- tic coherence in topic models. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. pp. 262–272. EMNLP '11, Association for Computational Linguistics, Stroudsburg, PA, USA (2011)

[8] Newman, D., Lau, J.H., Grieser, K., Baldwin, T.: Automatic evaluation of topic coherence. In: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics. pp. 100–108. HLT '10, Association for Computational Linguistics, Stroudsburg, PA, USA (2010)

[9] 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.

[10] Vorontsov, K., Potapenko, A.: Additive regularization of topic models. Machine Learning 101(1), 303–323 (2015)

[11] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, Xueqi Cheng: A Biterm Topic Model for Short Texts. WWW 2013.

[12] Yuan Zuo, Jichang Zhao, Ke Xu. Word Network Topic Model: a simple but general solution for short and imbalanced texts. 2014.

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