Семинар по использованию библиотеки для тематического моделирования BigARTM

Мурат Апишев
great-mel@yandex.ru

7 апреля 2015

1 Введение

Данный текст описывает процесс установки и базового использования библиотеки для тематического моделирования BigARTM [1]. В её основе лежит теория аддитивной регуляризации (ARTM), ознакомиться с которой можно в [1, 2].

2 Установка библиотеки

[2]

§2.1 Windows

Для установки библиотеки на компьютер с ОС Windows следует выполнить несколько шагов:

1. Скачайте релиз-версию библиотеки подходящей разрядности (http://docs.bigartm.org/en/latest/download.html).
2. Установите Python, скачанный по одной из ссылок (разрядность должна быть той же, что и у библиотеки):
   - https://www.python.org/ftp/python/2.7.9/python-2.7.9.amd64.msi
   - https://www.python.org/ftp/python/2.7.9/python-2.7.9.msi
3. Добавьте директорию 'C:/BigARTM/bin' в системную переменную окружения 'PATH', добавьте директорию 'C:/BigARTM/python' в системную переменную окружения 'PYTHONPATH'

```
set PATH=%PATH%;C:\BigARTM\bin
set PATH=%PATH%;C:\Python27;C:\Python27\Scripts
set PYTHONPATH=%PYTHONPATH%;C:\BigARTM\Python
```

[1] http://bigartm.org

[2] Подробнее об этом можно прочесть по адресу http://docs.bigartm.org/en/latest/tutorials/

Не забудьте заменить 'C:/BigARTM' и 'C:/Python27' на свои локальные пути к директориям.

4. Установите библиотеку Google Protocol Buffers, включённую в релиз-версию BigARTM.

   - Скопируйте файл 'C:/BigARTM/bin/protoc.exe' в директорию C:/BigARTM/protobuf/src'.
   - Запустите в командной строке следующие команды:

```
    cd C:\BigARTM\protobuf\Python
    python setup.py build
    python setup.py install
```

   Пропустите шаг 'python setup.py test', поскольку он приводит к нескольким неясным безвредным ошибкам. Для большей информации прочтите файл 'protobuf/python/README'.

§2.2 Linux & Mac OS X

Для пользователей ОС Linux и Mac OS X последовательность действий такая:

1. Загрузите последнюю версию кода BigARTM из репозитория и соберите его, используя приведённый ниже скрипт:

```
sudo apt get install git make cmake build essential libboost-all-dev
cd ~
git clone https://github.com/bigartm/bigartm.git
cd bigartm
mkdir build && cd build
cmake ..
make
```

2. Настройте Python API BigARTM. Для этого потребуется библиотека Google Protocol Buffers. Лучше всего использовать её версию, лежащую в директории 'bigartm/3rdparty'. Необходимо выполнить скрипт:

```
# Step 1   add BigARTM python bindings to PYTHONPATH
export PYTHONPATH=~/bigartm/src/python:$PYTHONPATH

# Step 2   install google protobuf
cd ~/bigartm
cp build/3rdparty/protobuf cmake/protoc/protoc 3rdparty/protobuf/src/
cd 3rdparty/protobuf/python
python setup.py build
sudo python setup.py install

# Step 3   point ARTM_SHARED_LIBRARY variable
#          to libartm.so (libartm.dylib) location

# for linux
export ARTM_SHARED_LIBRARY=~/bigartm/build/src/artm/libartm.so

```

```
# for Mac OS X
export ARTM_SHARED_LIBRARY=~/bigartm/build/src/artm/libartm.dylib
```

3 Пример пользовательского кода

В этом разделе будет рассмотрен пример использования Python API библиотеки для написания пользовательского скрипта, строящего регуляризованную тематическую модель. Будем рассматривать код по-блочно, подробно описывая смысл каждой строки. Сперва идёт блок, после — комментарии.

```
1  import sys
2  import glob
3  import os
4
5  home_folder = '/home/ubuntu/'
6  #             'C:/' for Windows
7  sys.path.append(home_folder + 'bigartm/src/python')
8  sys.path.append(home_folder + 'bigartm/src/python/artm')
9
10 import artm.messages_pb2
11 import artm.library
```

В первую очередь (после подключения необходимых пакетов Python) требуется добавить в `path` пути к библиотеке. Здесь необходимо изменить переменную `home_folder` в соответствии с расположением библиотеки на пользовательской машине. Затем происходит подключение файла с описаниями protobuf-сообщений для Python-интерфейса библиотеки и файла с интерфейсом `library.py`.

```
12 batches_disk_path = 'kos'
13 unique_tokens = artm.library.Library().LoadDictionary(\
14                  os.path.join(batches_disk_path, 'dictionary'))
```

Прежде, чем настраивать библиотеку, следует обозначить местоположение обрабатываемых данных. Без ограничения общности будем считать, что данные хранятся на диске в формате батчей BigARTM [3].

Путь к директории с данными прописывается в строке 12 [4]. Кроме самих батчей в папке с данными находится файл `dictionary`, содержащий словарь коллекции, собранный в процессе парсинга и создания батчей. Этот словарь может использоваться для того, чтобы инициализировать матрицу Φ (если этого не сделать, матрица будет наполняться новыми словами по мере их нахождения в обрабатываемых батчах) [5]. Он считывается в переменную `unique_tokens`.

[3] В BigARTM имеются утилиты-парсеры для создания батчей из нескольких популярных форматов Bag-of-Words. Кроме того, нужный парсер можно написать самостоятельно, однако это выходит за рамки этого занятия.

[4] Текстовые коллекции, уже в формате BigARTM, можно найти по адресу https://docs.bigartm.org/en/latest/download.html

[5] На самом деле, словари в BigARTM имеют самое широкое применение в механизме регуляризации. Словарь, помимо самих слов, может содержать много полезных данных, связанных с ними. Такими данными являются частоты слов в коллекции, их переводы на других языках и т.п. Такая информация бывает очень полезной при работе различных регуляризаторов.

```
15 with artm.library.MasterComponent() as master:
16     master.config().processors_count = 2
17     master.Reconfigure()
18     dictionary = master.CreateDictionary(unique_tokens)
```

Наконец, создадим объект `MasterComponent`, который представляет собой базовую сущность в BigARTM. Задача этого объекта — управлять всем процессом построения тематических моделей. Весь последующий код будет выполнятся в рамках этого компонента. Библиотека позволяет производить обучение произвольного числа моделей по одним и тем же данным. Это может быть полезно при использовании различных регуляризаторов и траекторий регуляризации. На рис. 1 показаны соотношения различных компонентов библиотеки в процессе работы алгоритма. Видно, что помимо моделей, `MasterComponent` также содержит в себе регуляризаторы (`Regularizer`), словари (`Dictionary`) и функционалы (`Score`), подключаемые к моделям по модульному принципу. В каждой модели с подключенным к ней регуляризатором ассоциирован свой коэффициент регуляризации τ.

В строчке 16 производится замена конфигурации `MasterComponent` (а именно, задаётся новое число потоков-обработчиков), после чего компонент обновляется. В 18 строчке на основе данных из переменной `unique_tokens` создаётся объект словаря `Dictionary`.

```
19     background_topics = []
20     objective_topics = []
21     all_topics = []
22
23     for i in range(0, 16):
24         topic_name = "topic" + str(i)
25         all_topics.append(topic_name)
26         if i < 14:
27             objective_topics.append(topic_name)
28         else:
29             background_topics.append(topic_name)
```

Есть предположение о том, что все темы модели разумно поделить на две группы: предметные (`objective`) и фоновые (`background`). Задача последних — собирать все общеупотребительные слова коллекции (фон), что достигается путём сглаживания этих тем. Предметные темы — это фактически результат тематического моделирования. К ним, наоборот, лучше применять стратегию разреживания, позволяющую достигнуть существенной разреженности этих тем, а также их высокой взаимной различности. Темы в BigARTM идентифицируются своими именами. В описанном выше блоке кода создаются два списка имён, соответствующих предметным и фоновым темам.

```
30     perplexity_score          = master.CreatePerplexityScore()
31     top_tokens_score          = master.CreateTopTokensScore()
32     sparsity_theta_objective = master.CreateSparsityThetaScore(\
33                                topic_names=objective_topics)
34     sparsity_phi_objective    = master.CreateSparsityPhiScore(\
35                                topic_names=objective_topics)
```

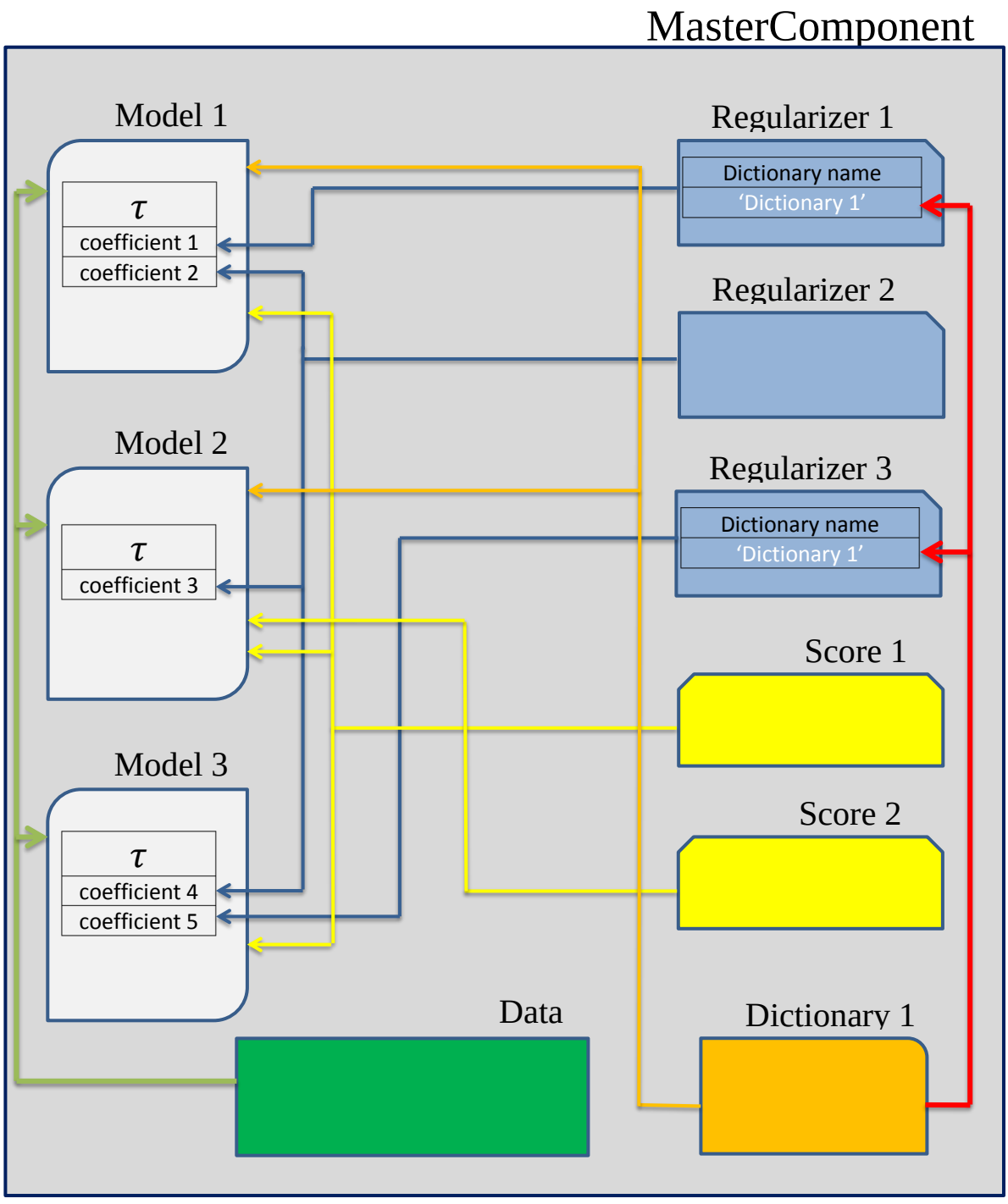

Рис. 1. Пример схемы обучения моделей в BigARTM.

В строках 30–36 в `MasterComponent` добавляются функционалы качества. Делается это простым вызовом соответсвующих методов [6]. Здесь, сверху вниз, создаются

[6]Эти методы являются «синтаксическим сахаром», создающим объекты с параметрами по-умолчанию. На самом деле, при создании фугкционалов, регуляризаторов и т.п., можно задавать их конфигурации гораздо более детально.

следующие метрики качества: перплексия на обучении, самые вероятные слова в каждой теме, разреженность матрицы Θ, разреженность матрицы Φ. Последним в качестве параметра передаются имена предметных тем, что означает, что данные функционалы будут рабоать только с этими темами (считать разреженность сглаживаемых фоновых тем бессмысленно, в них отсутствуют нулевые значения).

```
36      theta_objective  = master.CreateSmoothSparseThetaRegularizer(\
37                         topic_names=objective_topics)
38
39      theta_background = master.CreateSmoothSparseThetaRegularizer(\
40                         topic_names=background_topics)
41
42      phi_objective    = master.CreateSmoothSparsePhiRegularizer(\
43                         topic_names=objective_topics)
44
45      phi_background   = master.CreateSmoothSparsePhiRegularizer(\
46                         topic_names=background_topics)
47
48      decorrelator     = master.CreateDecorrelatorPhiRegularizer(\
49                         topic_names=objective_topics)
```

По тому же принципу, по которому выше создавались функционалы, в этом блоке кода создаются регуляризаторы. Они делятся на две группы. Первая «работает» с предметными темами (что опять-таки специфицируется списком имён тем, передаваемым в качестве параметра), вторая — с фоновыми. Сверху вниз создаются: регуляризатор разреживани предметных тем в матрице Θ, регуляризатор сглаживания фоновых тем в Θ, два аналогичных регуляризатора для Φ и декоррелятор предметных тем в Φ [7].

```
50      model = master.CreateModel(inner_iterations_count=20,\
51                                 topic_names=all_topics)
52      model.EnableScore(perplexity_score)
53      model.EnableScore(sparsity_theta_objective)
54      model.EnableScore(sparsity_phi_objective)
55      model.EnableScore(top_tokens_score)
56      model.EnableScore(theta_snippet_score)
57      model.EnableRegularizer(theta_objective, -1.0)
58      model.EnableRegularizer(theta_background, 0.5)
59      model.EnableRegularizer(phi_objective, -1.0)
60      model.EnableRegularizer(phi_background, 1.0)
61      model.EnableRegularizer(decorrelator, 1000000)
62      model.Initialize(dictionary)
```

После того, как все необходимые компоненты были созданы в `MasterComponent`, пришло время создать собственно модель `Model`, которая и будет производить обучение (для прозрачности кода настраиваться будет только одна модель, создать вторую и последующие можно по аналогии). В соответствии с диаграммой 1 модель является таким же компонентом, как и регуляризаторы и функционалы качества, поэтому её создание производится в строках 50 и 51 в `MasterComponent`. Параметрами здесь являются число внутренних итераций алгоритма (т.е. число проходов по каждому документу) и список с именами всех тем. Далее в модель добавляются функциона-

[7] Задача регуляризатора декорреляции тем — повышение их различности. По своей сути так же является разреживающим регуляризатором.

лы и регуляризаторы, делается это методами `EnableScore()` `EnableRegularizer()`, получающими на вход ссылки на созданные ранее объекты. Стоит обратить внимание на то, что каждый метод создания регуляризатора, помимо ссылки, получает на вход также коэффициент регуляризации данного регуляризатора. Этот коэффициент определяет степень и «направление» процесса регуляризации (например, сглаживание и разреживание делаются одним регуляризатором и отличаются только знаком своего коэффициента). Шкала значений коэффициента для каждого регуляризатора своя, кроме того, она зависит от коллекции (например, для одного регуляризатора коэффициент 10.0 будет считаться очень большим, для другого — пренебрежительно малым). Обычно подбор коэффициентов производится эмпирически.

В последней 62 строке производится инициализации тематической модели словарём коллекции (выше уже говорилось о том, что это такое).

```
63      batches = glob.glob(batches_disk_path + "/*.batch")
64      for iteration in range(0, 40):
65          for batch_index, batch_filename in enumerate(batches):
66              master.AddBatch(batch_filename=batch_filename)
67          master.WaitIdle()
68          model.Synchronize(decay_weight=0, apply_weight=1)
```

Этот фрагмент кода является ключевым и описывает собственно процесс построения модели. В 63 строчке из директории с батчами считываются имена всех имеющихся там батчей с документами. Строка 64 описывает цикл, каждая итерация которого — один проход по всей коллекции. Здесь указано число проходов, равное 40, обычно оно подбирается опытным путём и является структурным параметром алгоритма. В следующих двух строках описан цикл по всем именам батчей, в теле которого батчи отправляются на обработку. Следуется обратить внимание, что используемая для этого функция `AddBatch()` опять-таки является методом `MasterComponent`, а не модели `Model`, она (функция) инициирует обработку батча сразу всеми имеющимися в `MasterComponent` моделями.

В строчке 67 производится вызов метода `WaitIdle()`, задача которого — дожидаться окончания обработки коллекции. Её следует вызывать всегда перед любой выгрузкой данных из BigARTM (выгрузка значений функционалов, выгрузка матриц Φ и Θ и т.п.). Вызываемая в последней строчке функция `Syncronize()` занимается применением к матрице Φ всех полученных в результате работы алгоритма обновления. Её параметры отвечают за то, насколько сильно должны учитываться старые данные и новые поступившие инкременты (это зависит от специфики коллекции — если обрабатывается новостной поток, то разумно учитывать новую информацию с большим весом). Важно: описанный здесь алгоритм не является полностью онлайновым, поскольку обновления производятся только раз за проход по коллекции, а не через определённое количество обработанных документов.

```
69          print "Perplexity = %.3f"\
70                % perplexity_score.GetValue(model).value,
71
72          print ", Phi obj. sparsity = %.3f"\
73                % sparsity_phi_objective.GetValue(model).value,
74
75          print ", Theta obj. sparsity = %.3f"\
76                % sparsity_theta_objective.GetValue(model).value
77
```

```
78      artm.library.Visualizers.PrintTopTokensScore(\
79                  top_tokens_score.GetValue(model))
```

После того, как очередной проход по коллекции был завершён, разумно вывести на экран значения вычисляемых функционалов качества. Это позволяет, например, оценивать, насколько хорошо/плохо повлияла регуляризация на модель и сделать выводы о том, как следует изменить коэффициенты регуляризации, чтобы улучшить результат [8]. Извлечение значений всех функционалов производится одним и тем же способом: у переменной-ссылки, соответствующей данному функционалу, вызывается метод `GetValue()`. Поскольку один и тот же функционал может оценивать качество разных моделей, параметром метода является ссылка на нужный объект `Model`. Результат вызова метода — структура, полями которой являются различные величины, соответствующие данному функционалу (например, в подобной структуре для функционала, оценивающего разреженность Ф, хранятся число нулевых элементов, процент нулевых элементов, общее число элементов). Основное значение всегда содержится в поле `value`, и именно оно в описанном коде и выводится на экран.

В последней строчке, после последнего прохода по коллекции, с помощью описанного в Python API класса-утилиты `Visualizers`, на экран выводятся самые вероятные слова в каждой теме.

На скриншотах ниже можно увидеть результаты выполнения программы.

```
Perplexity = 6694.837 , Phi obj. sparsity = 0.565 , Theta obj. sparsity = 0.700
Perplexity = 4008.423 , Phi obj. sparsity = 0.801 , Theta obj. sparsity = 0.795
Perplexity = 3231.960 , Phi obj. sparsity = 0.871 , Theta obj. sparsity = 0.825
Perplexity = 2856.601 , Phi obj. sparsity = 0.903 , Theta obj. sparsity = 0.840
Perplexity = 2637.115 , Phi obj. sparsity = 0.921 , Theta obj. sparsity = 0.849
Perplexity = 2493.149 , Phi obj. sparsity = 0.932 , Theta obj. sparsity = 0.855
Perplexity = 2391.303 , Phi obj. sparsity = 0.940 , Theta obj. sparsity = 0.860
Perplexity = 2315.482 , Phi obj. sparsity = 0.945 , Theta obj. sparsity = 0.863
Perplexity = 2256.902 , Phi obj. sparsity = 0.949 , Theta obj. sparsity = 0.866
Perplexity = 2210.345 , Phi obj. sparsity = 0.953 , Theta obj. sparsity = 0.869
Perplexity = 2172.546 , Phi obj. sparsity = 0.956 , Theta obj. sparsity = 0.871
Perplexity = 2141.312 , Phi obj. sparsity = 0.958 , Theta obj. sparsity = 0.873
Perplexity = 2115.091 , Phi obj. sparsity = 0.960 , Theta obj. sparsity = 0.875
Perplexity = 2092.766 , Phi obj. sparsity = 0.962 , Theta obj. sparsity = 0.876
Perplexity = 2073.525 , Phi obj. sparsity = 0.964 , Theta obj. sparsity = 0.878
Perplexity = 2056.753 , Phi obj. sparsity = 0.965 , Theta obj. sparsity = 0.879
Perplexity = 2041.992 , Phi obj. sparsity = 0.967 , Theta obj. sparsity = 0.881
Perplexity = 2028.886 , Phi obj. sparsity = 0.968 , Theta obj. sparsity = 0.882
Perplexity = 2017.161 , Phi obj. sparsity = 0.969 , Theta obj. sparsity = 0.883
Perplexity = 2006.606 , Phi obj. sparsity = 0.970 , Theta obj. sparsity = 0.884
Perplexity = 1997.058 , Phi obj. sparsity = 0.971 , Theta obj. sparsity = 0.885
Perplexity = 1988.378 , Phi obj. sparsity = 0.971 , Theta obj. sparsity = 0.886
Perplexity = 1980.455 , Phi obj. sparsity = 0.972 , Theta obj. sparsity = 0.887
Perplexity = 1973.198 , Phi obj. sparsity = 0.973 , Theta obj. sparsity = 0.888
Perplexity = 1966.535 , Phi obj. sparsity = 0.973 , Theta obj. sparsity = 0.888
Perplexity = 1960.403 , Phi obj. sparsity = 0.974 , Theta obj. sparsity = 0.889
Perplexity = 1954.744 , Phi obj. sparsity = 0.974 , Theta obj. sparsity = 0.890
Perplexity = 1949.504 , Phi obj. sparsity = 0.975 , Theta obj. sparsity = 0.891
Perplexity = 1944.640 , Phi obj. sparsity = 0.975 , Theta obj. sparsity = 0.891
Perplexity = 1940.114 , Phi obj. sparsity = 0.975 , Theta obj. sparsity = 0.892
Perplexity = 1935.894 , Phi obj. sparsity = 0.976 , Theta obj. sparsity = 0.893
Perplexity = 1931.950 , Phi obj. sparsity = 0.976 , Theta obj. sparsity = 0.893
Perplexity = 1928.255 , Phi obj. sparsity = 0.976 , Theta obj. sparsity = 0.894
Perplexity = 1924.786 , Phi obj. sparsity = 0.977 , Theta obj. sparsity = 0.894
Perplexity = 1921.524 , Phi obj. sparsity = 0.977 , Theta obj. sparsity = 0.895
Perplexity = 1918.451 , Phi obj. sparsity = 0.977 , Theta obj. sparsity = 0.895
Perplexity = 1915.551 , Phi obj. sparsity = 0.977 , Theta obj. sparsity = 0.895
Perplexity = 1912.810 , Phi obj. sparsity = 0.977 , Theta obj. sparsity = 0.896
Perplexity = 1910.217 , Phi obj. sparsity = 0.978 , Theta obj. sparsity = 0.896
Perplexity = 1907.760 , Phi obj. sparsity = 0.978 , Theta obj. sparsity = 0.897
```

[8] В рамках этого занятия замена коэффициентов регуляризаторов, как и реконфигурация всех компонентов алгоритма не рассматривается. Найти информацию об этом можно в документации.

Список литературы

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

[2] *Vorontsov K. V., Potapenko A. A.* (2014). Additive Regularization of Topic Models, // Machine Learning Journal. Special Issue «Data Analysis and Intelligent Optimization with Applications».