BigARTM Documentation

Release 1.0

Konstantin Vorontsov

February 09, 2016

Contents

Getting help

Having trouble? We'd like to help!

- Learn more about BigARTM from our interactive notebooks (in English or in Russian), NLPub.ru, MachineLearning.ru and several Publications.
- Search for information in the archives of the bigartm-users mailing list, or post a question.
- Report bugs with BigARTM in our ticket tracker.
- Try the FAQ – it's got answers to many common questions.
- Looking for specific information? Try the genindex, or search.

CHAPTER 1

Introduction

Warning: Please note that this is a beta version of the BigARTM library which is still undergoing final testing before its official release. Should you encounter any bugs, lack of functionality or other problems with our library, please let us know immediately. Your help in this regard is greatly appreciated.

This is the documentation for the BigARTM library. BigARTM is a tool to infer topic models, based on a novel technique called Additive Regularization of Topic Models. This technique effectively builds multi-objective models by adding the weighted sums of regularizers to the optimization criterion. BigARTM is known to combine well very different objectives, including sparsing, smoothing, topics decorrelation and many others. Such combinations of regularizers significantly improves several quality measures at once almost without any loss of the perplexity.

Online. BigARTM never stores the entire text collection in the main memory. Instead the collection is split into small chunks called 'batches', and BigARTM always loads a limited number of batches into memory at any time.

Parallel. BigARTM can concurrently process several batches, and by doing so it substantially improves the throughput on multi-core machines. The library hosts all computation in several threads withing a single process, which enables efficient usage of shared memory across application threads.

Extensible API. BigARTM comes with an API in Python, but can be easily extended for all other languages that have an implementation of Google Protocol Buffers.

Cross-platform. BigARTM is known to be compatible with gcc, clang and the Microsoft compiler (VS 2012). We have tested our library on Windows, Ubuntu and Fedora.

Open source. BigARTM is released under the New BSD License. If you plan to use our library commercially, please beware that BigARTM depends on ZeroMQ. Please, make sure to review ZeroMQ license.

Acknowledgements. BigARTM project is supported by Russian Foundation for Basic Research (grants 14-07-00847, 14-07-00908, 14-07-31176), Skolkovo Institute of Science and Technology (project 081-R), Moscow Institute of Physics and Technology.

CHAPTER 2

Downloads

- **Windows**
  - Latest 32 bit release: BigARTM_v0.7.4_win32
  - Latest 64 bit release: BigARTM_v0.7.4_x64
  - All previous releases are available at https://github.com/bigartm/bigartm/releases

  Please refer to Basic BigARTM tutorial for Windows users for step by step installation procedure.

- **Linux, Mac OS-X**

  To run BigARTM on Linux and Mac OS-X you need to clone BigARTM repository (https://github.com/bigartm/bigartm) and build it as described in Basic BigARTM tutorial for Linux and Mac OS-X users.

- **Datasets**

  Download one of the following datasets to start experimenting with BigARTM. See Formats page for the description of input data formats.

Task	Source	#Words	#Items	Files
kos	UCI	6906	3430	– docword.kos.txt.gz (1 MB) – vocab.kos.txt (54 KB) – kos_1k (700 KB) – kos_dictionary
nips	UCI	12419	1500	– docword.nips.txt.gz (2.1 MB) – vocab.nips.txt (98 KB) – nips_200 (1.5 MB) – nips_dictionary
enron	UCI	28102	39861	– docword.enron.txt.gz (11.7 MB) – vocab.enron.txt (230 KB) – enron_1k (7.1 MB) – enron_dictionary
nytimes	UCI	102660	300000	– docword.nytimes.txt.gz (223 MB) – vocab.nytimes.txt (1.2 MB) – nytimes_1k (131 MB) – nytimes_dictionary
pubmed	UCI	141043	8200000	– docword.pubmed.txt.gz (1.7 GB) – vocab.pubmed.txt (1.3 MB) – pubmed_10k (1 GB) – pubmed_dictionary
wiki	Gensim	100000	3665223	– enwiki-20141208_10k (1.2 GB) – enwiki-

CHAPTER 3

Formats

This page describes input data formats compatible with BigARTM. Currently all formats correspond to Bag-of-words representation, meaning that all linguistic processing (lemmatization, tokenization, detection of n-grams, etc) needs to be done outside BigARTM.

1. Vowpal Wabbit is a single-format file, based on the following principles:
   - each document is depresented in a single line
   - all tokens are represented as strings (no need to convert them into an integer identifier)
   - token frequency defaults to `1.0`, and can be optionally specified after a colon (:)
   - namespaces (*`Batch.class_id`*) can be identified by a pipe (|)

   Example 1

```
doc1 Alpha Bravo:10 Charlie:5 |author Ola_Nordmann
doc2 Bravo:5 Delta Echo:3 |author Ivan_Ivanov
```

   Example 2

```
user123 |track-like track2 track5 track7 |track-play track1:10 track2:25 track3:2 track7:8 |trac
user345 |track-like track2 track5 track7 |track-play track1:10 track2:25 track3:2 track7:8 |trac
```

2. UCI Bag-of-words format consists of two files - `vocab.*.txt` and `docword.*.txt`. The format of the `docword.*.txt` file is 3 header lines, followed by NNZ triples:

```
D
W
NNZ
docID wordID count
docID wordID count
...
docID wordID count
```

   The file must be sorted on docID. Values of wordID must be unity-based (not zero-based). The format of the `vocab.*.txt` file is line containing wordID=n. Note that words must not have spaces or tabs. In `vocab.*.txt` file it is also possible to specify the namespace (*`Batch.class_id`*) for tokens, as it is shown in this example:

```
token1 @default_class
token2 custom_class
token3 @default_class
token4
```

Use space or tab to separate token from its class. Token that are not followed by class label automatically get ''@default_class'' as a label (see ''token4" in the example).

Unicode support. For non-ASCII characters save `vocab.*.txt` file in **UTF-8** format.

3. Batches (binary BigARTM-specific format).

   This is compact and efficient format, based on several protobuf messages in public BigARTM interface (*Batch*, *Item* and *Field*).

   - A batch is a collection of several items
   - An item is a collection of several fields
   - A field is a collection of pairs `(token_id, token_weight)`.

   The following example shows a Python code that generates a synthetic batch.

```
import artm.messages, random, uuid

num_tokens = 60
num_items = 100
batch = artm.messages.Batch()
batch.id = str(uuid.uuid4())
for token_id in range(0, num_tokens):
    batch.token.append('token' + str(token_id))

for item_id in range(0, num_items):
    item = batch.item.add()
    item.id = item_id
    field = item.field.add()
    for token_id in range(0, num_tokens):
        field.token_id.append(token_id)
        background_count = random.randint(1, 5) if (token_id >= 40) else 0
        topical_count = 10 if (token_id < 40) and ((token_id % 10) == (item_id % 10)) else 0
        field.token_weight.append(background_count + topical_count)
```

   Note that the batch has its local dictionary, `batch.token`. This dictionary which maps `token_id` into the actual token. In order to create a batch from textual files involve one needs to find all distinct words, and map them into sequential indices.

   `batch.id` must be set to a unique GUID in a format of `00000000-0000-0000-0000-000000000000`.

CHAPTER 4

Tutorials

4.1 Basic BigARTM tutorial for Windows users

This tutorial gives guidelines for installing and running existing BigARTM examples via command-line interface and from Python environment.

4.1.1 Download

Download latest binary distribution of BigARTM from https://github.com/bigartm/bigartm/releases. Explicit download links can be found at Downloads section (for 32 bit and 64 bit configurations).

The distribution will contain pre-build binaries, command-line interface and BigARTM API for Python. The distribution also contains a simple dataset and few python examples that we will be running in this tutorial. More datasets in BigARTM-compatible format are available in the Downloads section.

Refer to Windows distribution for details about other files, included in the binary distribution package.

4.1.2 Running BigARTM from command line

No installation steps are required to run BigARTM from command line. After unpacking binary distribution simply open command prompt (`cmd.exe`), change current directory to `bin` folder inside BigARTM package, and run `cpp_client.exe` application as in the following example. As an optional step, we recommend to add `bin` folder of the BigARTM distribution to your `PATH` system variable.

```
>C:\BigARTM\bin>set PATH=%PATH%;C:\BigARTM\bin
>C:\BigARTM\bin>cpp_client.exe -v ../python/examples/vocab.kos.txt -d ../python/examples/docword.kos
Parsing text collection... OK.
Iteration 1 took  197 milliseconds.
    Test perplexity = 7108.35,
    Train perplexity = 7106.18,
    Test spatsity theta = 0,
    Train sparsity theta = 0,
    Spatsity phi = 0.000144802,
    Test items processed = 343,
    Train items processed = 3087,
    Kernel size = 5663,
    Kernel purity = 0.958901,
    Kernel contrast = 0.292389
Iteration 2 took  195 milliseconds.
    Test perplexity = 2563.31,
```

```
    Train perplexity = 2517.07,
    Test spatsity theta = 0,
    Train sparsity theta = 0,
    Spatsity phi = 0.000144802,
    Test items processed = 343,
    Train items processed = 3087,
    Kernel size = 5559.5,
    Kernel purity = 0.956709,
    Kernel contrast = 0.298198
...
#1: november(0.054) poll(0.015) bush(0.013) kerry(0.012) polls(0.012) governor(0.011)
#2: bush(0.0083) president(0.0059) republicans(0.0047) house(0.0042) people(0.0039) administration(0
#3: bush(0.031) iraq(0.018) war(0.012) kerry(0.0096) president(0.0078) administration(0.0076)
#4: kerry(0.018) democratic(0.013) dean(0.012) campaign(0.0097) poll(0.0095) race(0.0082)
ThetaMatrix (last 7 processed documents, ids = 1995,1996,1997,1998,1992,2000,1994):
Topic0: 0.02104 0.02155 0.00604 0.00835 0.00965 0.00006 0.91716
Topic1: 0.15441 0.76643 0.06484 0.11643 0.20409 0.00006 0.00957
Topic2: 0.00399 0.16135 0.00093 0.03890 0.10498 0.00001 0.00037
Topic3: 0.82055 0.05066 0.92819 0.83632 0.68128 0.99987 0.07289
```

We recommend to download larger datasets, available in Downloads section. All `docword` and `vocab` files can be consumed by BigARTM exactly as in the previous example.

Internally BigARTM always parses such files into batches format (for example, enron_1k (7.1 MB)). If you have downloaded such pre-parsed collection, you may feed it into BigARTM as follows:

```
>C:\BigARTM\bin>cpp_client.exe --batch_folder C:\BigARTM\enron
Reuse 40 batches in folder 'enron'
Loading dictionary file... OK.
Iteration 1 took  2502 milliseconds.
```

For more information about `cpp_client.exe` refer to `/ref/cpp_client` section.

4.1.3 Configure BigARTM Python API

1. Install Python, for example from the following links:
   - Python 2.7.9, 64 bit – https://www.python.org/ftp/python/2.7.9/python-2.7.9.amd64.msi, or
   - Python 2.7.9, 32 bit – https://www.python.org/ftp/python/2.7.9/python-2.7.9.msi

   Remember that the version of BigARTM package must match your version Python installed on your machine. If you have 32 bit operating system then you must select 32 bit for Python and BigARTM package. If you have 64 bit operating system then you are free to select either version. However, please note that memory usage of 32 bit processes is limited by 2 GB. For this reason we recommend to select 64 bit configurations.

   Also you need to have several Python libraries to be installed on your machine:
   - numpy >= 1.9.2
   - scipy >= 0.15.0
   - pandas >= 0.16.2
   - scikit-learn >= 0.16.1
2. Add `C:\BigARTM\bin` folder to your `PATH` system variable, and add `C:\BigARTM\python` to your `PYTHONPATH` system variable:

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

Remember to change `C:\BigARTM` and `C:\Python27` with your local folders.

3. Setup *Google Protocol Buffers* library, included in the BigARTM release package.
   - Copy `C:\BigARTM\bin\protoc.exe` file into `C:\BigARTM\protobuf\src` folder
   - Run the following commands from command prompt

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

Avoid `python setup.py test` step, as it produces several confusing errors. Those errors are harmless. For further details about protobuf installation refer to protobuf/python/README.

If you are getting errors when configuring or using Python API, please refer to Troubleshooting chapter in Basic BigARTM tutorial for Linux and Mac OS-X users. The list of issues is common between Windows and Linux.

4.1.4 Running BigARTM from Python API

Refer to ARTM notebook (in Russian or in English), which describes high-level Python API of BigARTM.

4.2 Basic BigARTM tutorial for Linux and Mac OS-X users

Currently there is no distribution package of BigARTM for Linux. BigARTM had been tested on several Linux OS, and it is known to work well, but you have to get the source code and compile it locally on your machine.

4.2.1 Download sources and build

Clone the latest BigARTM code from our github repository, and build it via CMake as in the following script.

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

4.2.2 Running BigARTM from command line

There is a simple utility `bigartm`, which allows you to run BigARTM from command line. To experiment with this tool you need a small dataset, which you can get via the following script. More datasets are available through Downloads page.

```
cd ~/bigartm
mkdir datasets && cd datasets
wget https://s3-eu-west-1.amazonaws.com/artm/docword.kos.txt.gz
wget https://s3-eu-west-1.amazonaws.com/artm/vocab.kos.txt
```

```
gunzip docword.kos.txt.gz
../build/src/bigartm/bigartm -d docword.kos.txt -v vocab.kos.txt
```

4.2.3 Configure BigARTM Python API

For more advanced scenarios you need to configure Python interface for BigARTM. To use BigARTM from Python you need to use Google Protobuf. We recommend to use 'protobuf 2.5.1-pre', included in bigartm/3rdparty.

```
# Step 1 - add BigARTM python bindings to PYTHONPATH
export PYTHONPATH=~/bigartm/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
export ARTM_SHARED_LIBRARY=~/bigartm/build/src/artm/libartm.so        # for linux
export ARTM_SHARED_LIBRARY=~/bigartm/build/src/artm/libartm.dylib     # for Mac OS X
```

At this point you may run examples under `~/bigartm/python/examples`.

4.2.4 Troubleshooting

```
>python setup.py build
  File "setup.py", line 52
        print "Generating %s..." % output

SyntaxError: Missing parentheses in call to `print`
```

This error may happen during google protobuf installation. It indicates that you are using `Python 3`, which is not supported by BigARTM. (see this question on StackOverflow for more details on the error around *print*). Please use `Python 2.7.9` to workaround this issue.

```
ubuntu@192.168.0.1:~/bigartm/python/examples$ python example01_synthetic_collection.py
Traceback (most recent call last):
  File "example01_synthetic_collection.py", line 6, in <module>
    import artm.messages_pb2, artm.library, random, uuid
ImportError: No module named artm.messages_pb2
```

This error indicate that python is unable to locate `messages_pb2.py and ``library.py` files. Please verify if you executed `Step #1` in the instructions above.

```
ubuntu@192.168.0.1:~/bigartm/python/examples$ python example01_synthetic_collection.py
Traceback (most recent call last):
  File "example01_synthetic_collection.py", line 6, in <module>
    import artm.messages_pb2, artm.library, random, uuid
  File "/home/ubuntu/bigartm/python/messages_pb2.py", line 4, in <module>
    from google.protobuf import descriptor as _descriptor
ImportError: No module named google.protobuf
```

This error indicated that python is unable to locate protobuf library. Please verify if you executed `Step #2` in the instructions above. If you do not have permissions to execute `sudo`

python setup.py install step, you may also try to update PYTHONPATH manually: PYTHONPATH="/home/ubuntu/bigartm/3rdparty/protobuf/python:/home/ubuntu/bigartm/python:$PYTH

```
ubuntu@192.168.0.1:~/bigartm/python/examples$ python example01_synthetic_collection.py
libartm.so: cannot open shared object file: No such file or directory,
fall back to ARTM_SHARED_LIBRARY environment variable
Traceback (most recent call last):
  File "example01_synthetic_collection.py", line 27, in <module>
    with artm.library.MasterComponent() as master:
  File "/home/ubuntu/bigartm/python/artm/library.py", line 179, in __init__
    lib = Library().lib_
  File "/home/ubuntu/bigartm/python/artm/library.py", line 107, in __init__
    self.lib_ = ctypes.CDLL(os.environ['ARTM_SHARED_LIBRARY'])
  File "/usr/lib/python2.7/UserDict.py", line 23, in __getitem__
    raise KeyError(key)
KeyError: 'ARTM_SHARED_LIBRARY'
```

This error indicate that BigARTM's python interface can not locate libartm.so (libartm.dylib) files. Please verify if you executed `Step #3` correctly.

4.2.5 BigARTM on Travis-CI

To get a live usage example of BigARTM you may check BigARTM's .travis.yml script and the latest continuous integration build.

4.3 BigARTM command line utility

This document provides an overview of `bigartm` command-line utility shipped with BigARTM.

For a detailed description of `bigartm` command line interface refer to bigartm.exe notebook (in Russian).

In brief, you need to download some input data (a textual collection represented in bag-of-words format). We recommend to download *vocab* and *docword* files by links provided in Downloads section of the tutorial. Then you can use `bigartm` as described by `bigartm --help`:

```
BigARTM - library for advanced topic modeling (http://bigartm.org):

Input data:
  -c [ --read-vw-corpus ] arg         Raw corpus in Vowpal Wabbit format
  -d [ --read-uci-docword ] arg       docword file in UCI format
  -v [ --read-uci-vocab ] arg         vocab file in UCI format
  --read-cooc arg                     read co-occurrences format
  --batch-size arg (=500)             number of items per batch
  --use-batches arg                   folder with batches to use

Dictionary:
  --dictionary-min-df arg             filter out tokens present in less than N
                                      documents / less than P% of documents
  --dictionary-max-df arg             filter out tokens present in less than N
                                      documents / less than P% of documents
  --use-dictionary arg                filename of binary dictionary file to use

Model:
  --load-model arg                    load model from file before processing
  -t [ --topics ] arg (=16)           number of topics
  --use-modality arg                  modalities (class_ids) and their weights
```

```
  --predict-class arg                 target modality to predict by theta
                                      matrix

Learning:
  -p [ --passes ] arg (=0)            number of outer iterations
  --inner-iterations-count arg (=10)  number of inner iterations
  --update-every arg (=0)             [online algorithm] requests an update of
                                      the model after update_every document
  --tau0 arg (=1024)                  [online algorithm] weight option from
                                      online update formula
  --kappa arg (=0.699999988)          [online algorithm] exponent option from
                                      online update formula
  --reuse-theta                       reuse theta between iterations
  --regularizer arg                   regularizers (SmoothPhi,SparsePhi,SmoothT
                                      heta,SparseTheta,Decorrelation)
  --threads arg (=0)                  number of concurrent processors (default:
                                      auto-detect)
  --async                             invoke asynchronous version of the online
                                      algorithm
  --model-v06                         use legacy model from BigARTM v0.6.4

Output:
  --save-model arg                    save the model to binary file after
                                      processing
  --save-batches arg                  batch folder
  --save-dictionary arg               filename of dictionary file
  --write-model-readable arg          output the model in a human-readable
                                      format
  --write-dictionary-readable arg     output the dictionary in a human-readable
                                      format
  --write-predictions arg             write prediction in a human-readable
                                      format
  --write-class-predictions arg       write class prediction in a
                                      human-readable format
  --write-scores arg                  write scores in a human-readable format
  --force                             force overwrite existing output files
  --csv-separator arg (=;)            columns separator for
                                      --write-model-readable and
                                      --write-predictions. Use \t or TAB to
                                      indicate tab.
  --score-level arg (=2)              score level (0, 1, 2, or 3
  --score arg                         scores (Perplexity, SparsityTheta,
                                      SparsityPhi, TopTokens, ThetaSnippet, or
                                      TopicKernel)
  --final-score arg                   final scores (same as scores)

Other options:
  -h [ --help ]                       display this help message
  --response-file arg                 response file
  --paused                            start paused and waits for a keystroke
                                      (allows to attach a debugger)
  --disable-avx-opt                   disable AVX optimization (gives similar
                                      behavior of the Processor component to
                                      BigARTM v0.5.4)
  --use-dense-bow                     use dense representation of bag-of-words
                                      data in processors
  --time-limit arg (=0)               limit execution time in milliseconds
```

```
Examples:

* Download input data:
  wget https://s3-eu-west-1.amazonaws.com/artm/docword.kos.txt
  wget https://s3-eu-west-1.amazonaws.com/artm/vocab.kos.txt
  wget https://s3-eu-west-1.amazonaws.com/artm/vw.mmro.txt

* Parse docword and vocab files from UCI bag-of-word format; then fit topic model with 20 topics:
  bigartm -d docword.kos.txt -v vocab.kos.txt -t 20 --passes 10

* Parse VW format; then save the resulting batches and dictionary:
  bigartm --read-vw-corpus vw.mmro.txt --save-batches mmro_batches --save-dictionary mmro.dict

* Parse VW format from standard input; note usage of single dash '-' after --read-vw-corpus:
  cat vw.mmro.txt | bigartm --read-vw-corpus - --save-batches mmro2_batches --save-dictionary mmro2.d

* Load and filter the dictionary on document frequency; save the result into a new file:
  bigartm --use-dictionary mmro.dict --dictionary-min-df 5 dictionary-max-df 40% --save-dictionary mm

* Load the dictionary and export it in a human-readable format:
  bigartm --use-dictionary mmro.dict --write-dictionary-readable mmro.dict.txt

* Use batches to fit a model with 20 topics; then save the model in a binary format:
  bigartm --use-batches mmro_batches --passes 10 -t 20 --save-model mmro.model

* Load the model and export it in a human-readable format:
  bigartm --load-model mmro.model --write-model-readable mmro.model.txt

* Load the model and use it to generate predictions:
  bigartm --read-vw-corpus vw.mmro.txt --load-model mmro.model --write-predictions mmro.predict.txt

* Fit model with two modalities (@default_class and @target), and use it to predict @target label:
  bigartm --use-batches <batches> --use-modality @default_class,@target --topics 50 --passes 10 --sav
  bigartm --use-batches <batches> --use-modality @default_class,@target --topics 50 --load-model mode
          --write-predictions pred.txt --csv-separator=tab
          --predict-class @target --write-class-predictions pred_class.txt --score ClassPrecision

* Fit simple regularized model (increase sparsity up to 60-70%):
  bigartm -d docword.kos.txt -v vocab.kos.txt --dictionary-max-df 50% --dictionary-min-df 2
          --passes 10 --batch-size 50 --topics 20 --write-model-readable model.txt
          --regularizer "0.05 SparsePhi" "0.05 SparseTheta"

* Fit more advanced regularize model, with 10 sparse objective topics, and 2 smooth background topics
  bigartm -d docword.kos.txt -v vocab.kos.txt --dictionary-max-df 50% --dictionary-min-df 2
          --passes 10 --batch-size 50 --topics obj:10;background:2 --write-model-readable model.txt
          --regularizer "0.05 SparsePhi #obj"
          --regularizer "0.05 SparseTheta #obj"
          --regularizer "0.25 SmoothPhi #background"
          --regularizer "0.25 SmoothTheta #background"

* Configure logger to output into stderr:
  tset GLOG_logtostderr=1 & bigartm -d docword.kos.txt -v vocab.kos.txt -t 20 --passes 10
```

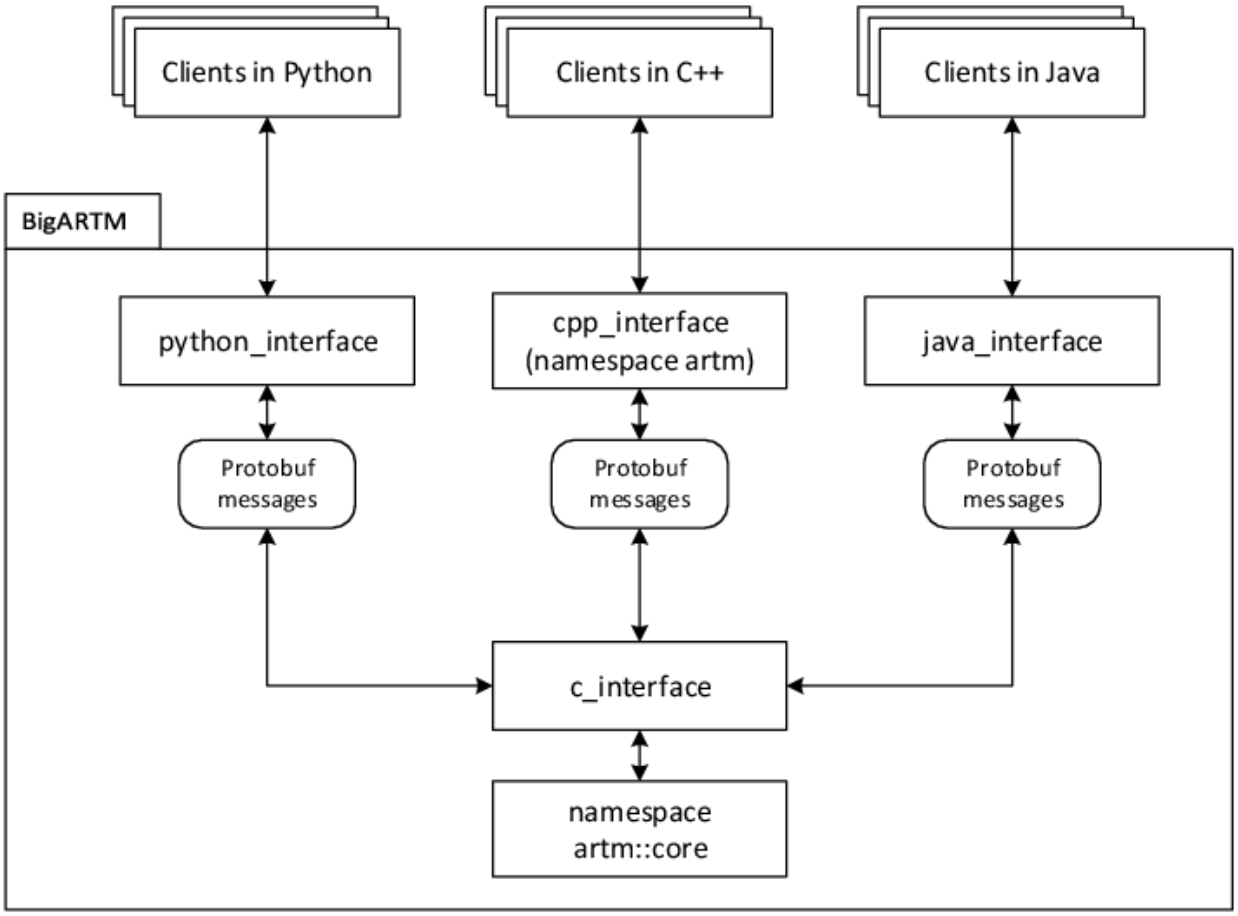
CHAPTER 5

BigARTM FAQ

5.1 Can I use BigARTM from other programming languages (not Python)?

Yes, as long as your language has an implementation of Google Protocol Buffers (the list can be found here). Note that Google officially supports C++, Python and Java.

The following figure shows how to call BigARTM methods directly on `artm.dll` (Windows) or `artm.so` (Linux).



To write your API please refer to Plain C interface of BigARTM.

5.2 How to retrieve Theta matrix from BigARTM

Theta matrix is a matrix that contains the distribution of several items (columns of the matrix) into topics (rows of the matrix). There are three ways to retrieve such information from BigARTM, and the correct way depends on your scenario.

1. **You want to get Theta matrix for the same collection as you have used to infer the topic model.**

   Set *MasterComponentConfig.cache_theta* to *true* prior to the last iteration, and after the iteration use `MasterComponent::GetThetaMatrix()` (in C++) or *MasterComponent.GetThetaMatrix* (in Python) to retrieve Theta matrix.

2. **You want to repeatedly monitor a small portion of the Theta matrix during ongoing iterations.**

   In this case you should create Theta Snippet score, defined via *ThetaSnippetScoreConfig*, and then use `MasterComponent::GetScoreAs<T>()` to retrieve the resulting *ThetaSnippetScore* message.

   This configuration of Theta Snippet score require you to provide *ThetaSnippetScoreConfig.item_id* listing all IDs of the items that should have Theta's collected. If you created the batches manually you should have specified such IDs in *Item.id* field. If you used other methods to parse the collection from disk then you shouldt try using sequential IDs, starting with 1.

   Remember that Theta snippet score is designed to handle only a small number of items. Attemp to retrieve 100+ items will have a negative effect on performance.

3. **You want to classify a new set of items with an existing model.**

   In this case you need to create a *Batch*, containing your new items. Then copy this batch to *GetThetaMatrixArgs.batch* message, specify *GetThetaMatrixArgs.model_name*, and use `MasterComponent::GetThetaMatrix()` (in C++) or *MasterComponent.GetThetaMatrix* (in Python) to retrieve Theta matrix. In this case there is no need set *MasterComponentConfig.cache_theta* to *true*.

Check example11_get_theta_matrix.py for further examples.

CHAPTER 6

BigARTM Developer's Guide

This document describes the development process of BigARTM library.

You should not follow this guide if you are using pre-built BigARTM library via command-line interface or from Python environment. (refer to to Basic BigARTM tutorial for Windows users or Basic BigARTM tutorial for Linux and Mac OS-X users depending on your operating system).

6.1 Downloads (Windows)

Download and install the following tools:

- **Git for Windows from http://git-scm.com/download/win**
  - https://github.com/msysgit/msysgit/releases/download/Git-1.9.5-preview20141217/Git-1.9.5-preview20141217.exe
- **Github for Windows from https://windows.github.com/**
  - https://github-windows.s3.amazonaws.com/GitHubSetup.exe
- Visual Studio 2013 Express for Windows Desktop from https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx
- **CMake from http://www.cmake.org/download/**
  - http://www.cmake.org/files/v3.0/cmake-3.0.2-win32-x86.exe
- **Prebuilt Boost binaries from http://sourceforge.net/projects/boost/files/boost-binaries/, for example these two:**
  - http://sourceforge.net/projects/boost/files/boost-binaries/1.57.0/boost_1_57_0-msvc-12.0-32.exe/download
  - http://sourceforge.net/projects/boost/files/boost-binaries/1.57.0/boost_1_57_0-msvc-12.0-64.exe/download
- **Python from https://www.python.org/downloads/**
  - 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
- (optional) If you plan to build documentation, download and install sphinx-doc as described here: http://sphinx-doc.org/latest/index.html
- (optional) 7-zip – http://www.7-zip.org/a/7z920-x64.msi

- (optional) Putty – http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe

All explicit links are given just for convenience if you are setting up new environment. You are free to choose other versions or tools, and most likely they will work just fine for BigARTM. Remember to match the following: * Visual Studio version must match Boost binaries version, unless you build Boost yourself * Use the same configuration (32 bit or 64 bit) for your Python and BigARTM binaries

6.2 Source code

BigARTM is hosted in public GitHub repository:

https://github.com/bigartm/bigartm

We maintain two branches: master and stable. *master* branch is the latest source code, potentially including some unfinished features. *stable* branch will be lagging behind *master*, and moved forward to *master* as soon as mainteiners decide that it is ready. Typically this should happen at the end of each month. At the same point we will introduce a new tag (something like v0.7.3) and produce a new release for Windows. In addition, *stable* branch also might receive small urgent fixes in between releases, typically to address critical issues reported by our users. Such fixes will be also included in *master* branch.

To contribute a fix you should fork the repository, code your fix and submit a pull request. against *master* branch. All pull requests are regularly monitored by BigARTM maintainers and will be soon merged. Please, keep monitoring the status of your pull request on travis, which is a continuous integration system used by BigARTM project.

6.3 Build C++ code on Windows

The following steps describe the procedure to build BigARTM's C++ code on Windows.

- Download and install GitHub for Windows.
- Clone https://github.com/bigartm/bigartm/ repository to any location on your computer. This location is further refered to as `$(BIGARTM_ROOT)`.
- Download and install Visual Studio 2012 or any newer version. BigARTM will compile just fine with any edition, including any Visual Studio Express edition (available at www.visualstudio.com).
- Install CMake (tested with cmake-3.0.1, Win32 Installer).

  Make sure that CMake executable is added to the `PATH` environmental variable. To achieve this either select the option *"Add CMake to the system PATH for all users"* during installation of CMake, or add it to the `PATH` manually.
- Download and install Boost 1.55 or any newer version.

  We suggest to use the Prebuilt Windows Binaries. Make sure to select version that match your version of Visual Studio. You may choose to work with either x64 or Win32 configuration, both of them are supported.
- Configure system variables `BOOST_ROOT` and `Boost_LIBRARY_DIR`.

  If you have installed boost from the link above, and used the default location, then the setting should look similar to this:

```
setx BOOST_ROOT C:\local\boost_1_56_0
setx BOOST_LIBRARYDIR C:\local\boost_1_56_0\lib32-msvc-12.0
```

  For all future details please refer to the documentation of FindBoost module. We also encourage new CMake users to step through CMake tutorial.

- Install Python 2.7 (tested with Python 2.7.6).

  You may choose to work with either x64 or Win32 version of the Python, but make sure this matches the configuration of BigARTM you have choosed earlier. The x64 installation of python will be incompatible with 32 bit BigARTM, and virse versus.

- Use CMake to generate Visual Studio projects and solution files. To do so, open a command prompt, change working directory to `$(BIGARTM_ROOT)` and execute the following commands:

```
mkdir build
cd build
cmake ..
```

  You might have to explicitly specify the cmake generator, especially if you are working with x64 configuration. To do so, use the following syntax:

```
cmake .. -G"Visual Studio 12 Win64"
```

  CMake will generate Visual Studio under `$(BIGARTM_ROOT)/build/`.

- Open generated solution in Visual Studio and build it as you would usually build any other Visual Studio solution. You may also use MSBuild from Visual Studio command prompt.

  The build will output result into the following folders:

  - `$(BIGARTM_ROOT)/build/bin/[Debug|Release]` — binaries (.dll and .exe)
  - `$(BIGARTM_ROOT)/build/lib/[Debug|Release]` — static libraries

At this point you should be able to run BigARTM tests, located here: `$(BIGARTM_ROOT)/build/bin/*/artm_tests.exe`.

6.4 Python code on Windows

- Install Python 2.7 (this step is already done if you are following the instructions above),
- Add Python to the `PATH` environmental variable

  http://stackoverflow.com/questions/6318156/adding-python-path-on-windows-7

- Follow the instructions in `README` file in directory `$(BIGARTM_ROOT)/3rdparty/protobuf/python/`. In brief, this instructions ask you to run the following commands:

```
python setup.py build
python setup.py test
python setup.py install
```

  On second step you fill see two failing tests:

```
Ran 216 tests in 1.252s
FAILED (failures=2)
```

  This 2 failures are OK to ignore.

At this point you should be able to run BigARTM tests for Python, located under `$(BIGARTM_ROOT)/python/tests/`.

- [Optional] Download and add to MSVS Python Tools 2.0. All necessary instructions can be found at https://pytools.codeplex.com/. This will allow you debug you Python scripts using Visual Studio. You may start with the following solution: `$(BIGARTM_ROOT)/src/artm_vs2012.sln`.

6.5 Build C++ code on Linux

Refer to Basic BigARTM tutorial for Linux and Mac OS-X users.

6.6 Working with iPython notebooks remotely

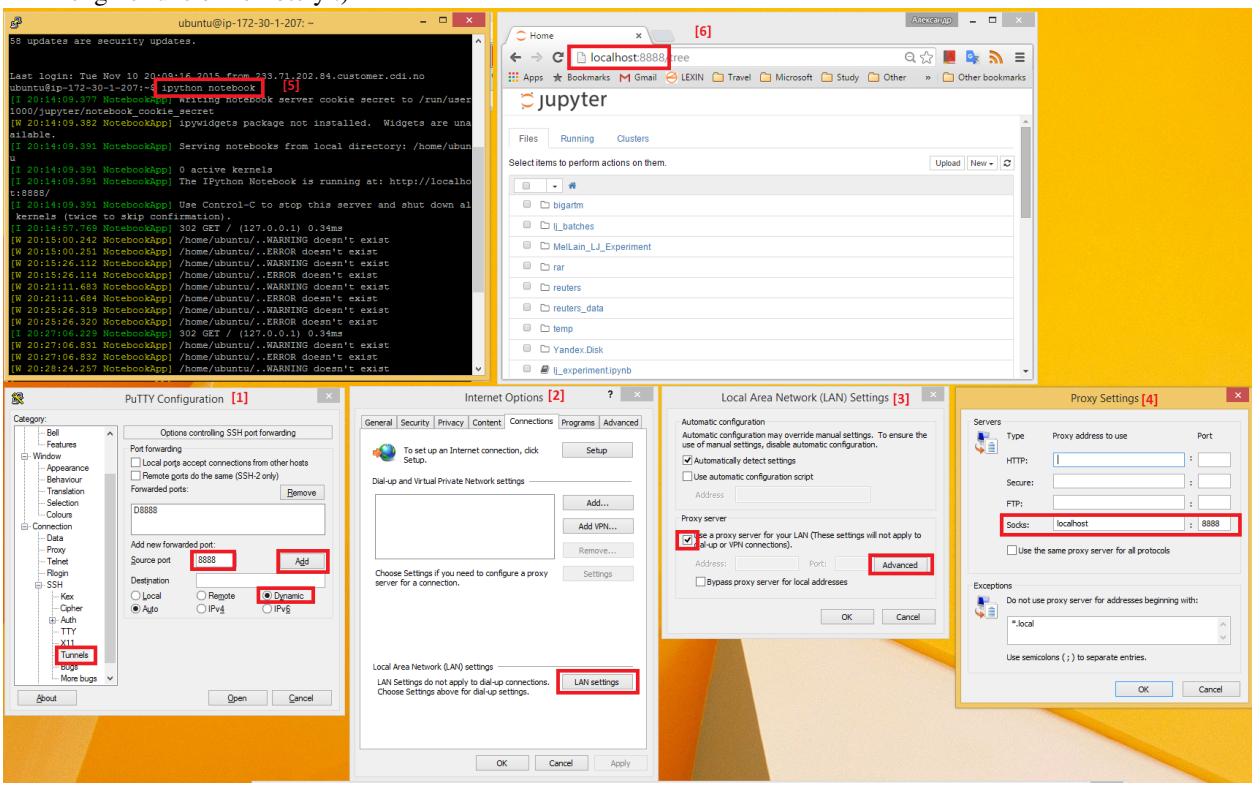
It turned out to be common scenario to run BigARTM on a Linux server (for example on Amazon EC2), while connecting to it from Windows through `putty`. Here is a convenient way to use `ipython notebook` in this scenario:

1. Connect to the Linux machine via putty. Putty needs to be configured with dynamic tunnel for port `8888` as describe here on this page (port `8888` is a default port for `ipython notebook`). The same page describes how to configure internet properties:

   Clicking on Settings in Internet Explorer, or Proxy Settings in Google Chrome, should open this dialogue. Navigate through to the Advanced Proxy section and add localhost:9090 as a SOCKS Proxy.

2. Start `ipython notebook` in your putty terminal.

3. Open your favourite browser on Windows, and go to http://localhost:8888. Enjoy your notebook while the engine runs on remotely :)



6.7 Compiling .proto files on Windows

1. Open a new command prompt
2. Copy the following file into `$(BIGARTM_ROOT)/src/`

- `$(BIGARTM_ROOT)/build/bin/CONFIG/protoc.exe`

Here CONFIG can be either Debug or Release (both options will work equally well).

3. Change working directory to `$(BIGARTM_ROOT)/src/`
4. Run the following commands

```
.\protoc.exe --cpp_out=. --python_out=. .\artm\messages.proto
.\protoc.exe --cpp_out=. .\artm\core\internals.proto
```

6.8 Code style

Configure Visual Studio

Open *Tools / Text Editor / All languages / Tabs* and configure as follows:
- Indenting - smart,
- Tab size - 2,
- Indent size - 2,
- Select "insert spaces".

We also suggest to configure Visual Studio to show space and tab crlf characters (shortcut: Ctrl+R, Ctrl+W), and enable vertical line at 120 characters.

In the code we follow google code style with the following changes:

- Exceptions are allowed
- Indentation must be 2 spaces. Tabs are not allowed.
- No lines should exceed 120 characters.

All .h and .cpp files under `$(BIGARTM_ROOT)/src/artm/` must be verified for code style with cpplint.py script. Files, generated by protobuf compiler, are the only exceptions from this rule.

To run the script you need some version of Python installed on your machine. Then execute the script like this:

```
python cpplint.py --linelength=120 <filename>
```

On Windows you may run this master-script to check all required files:

```
$(BIGARTM_ROOT/utils/cpplint_all.bat.
```

CHAPTER 7

Release Notes

7.1 BigARTM v0.7.0 Release notes

We are happy to introduce BigARTM v0.7.0, which brings you the following changes:

- New-style models
- Network modus operandi is removed
- Coherence regularizer and scores (experimental)

7.1.1 New-style models

BigARTM v0.7.0 exposes new APIs to give you additional control over topic model inference:

- ProcessBatches
- MergeModel
- RegularizeModel
- NormalizeModel

Besides being more flexible, new APIs bring many additional benefits:

- Fully deterministic inference, no dependency on threads scheduling or random numbers generation
- Less bottlenecks for performance (DataLoader and Merger threads are removed)
- Phi-matrix regularizers can be implemented externally
- Capability to output Phi matrices directly into your NumPy matrices (scheduled for BigARTM v0.7.2)
- Capability for store Phi matrices in sparse format (scheduled for BigARTM v0.7.3)
- Capability for async ProcessBatches and non-blocking online algorithm (BigARTM v0.7.4)
- Form solid foundation for high performance networking (BigARTM v0.8.X)

The picture below illustrates scalability of BigARTM v0.7.0 vs v0.6.4. Top chart (in green) corresponds to CPU usage at 28 cores on machine with 32 virtual cores (16 physical cores + hyper threading). As you see, new version is much more stable. In addition, new version consumes less memory.

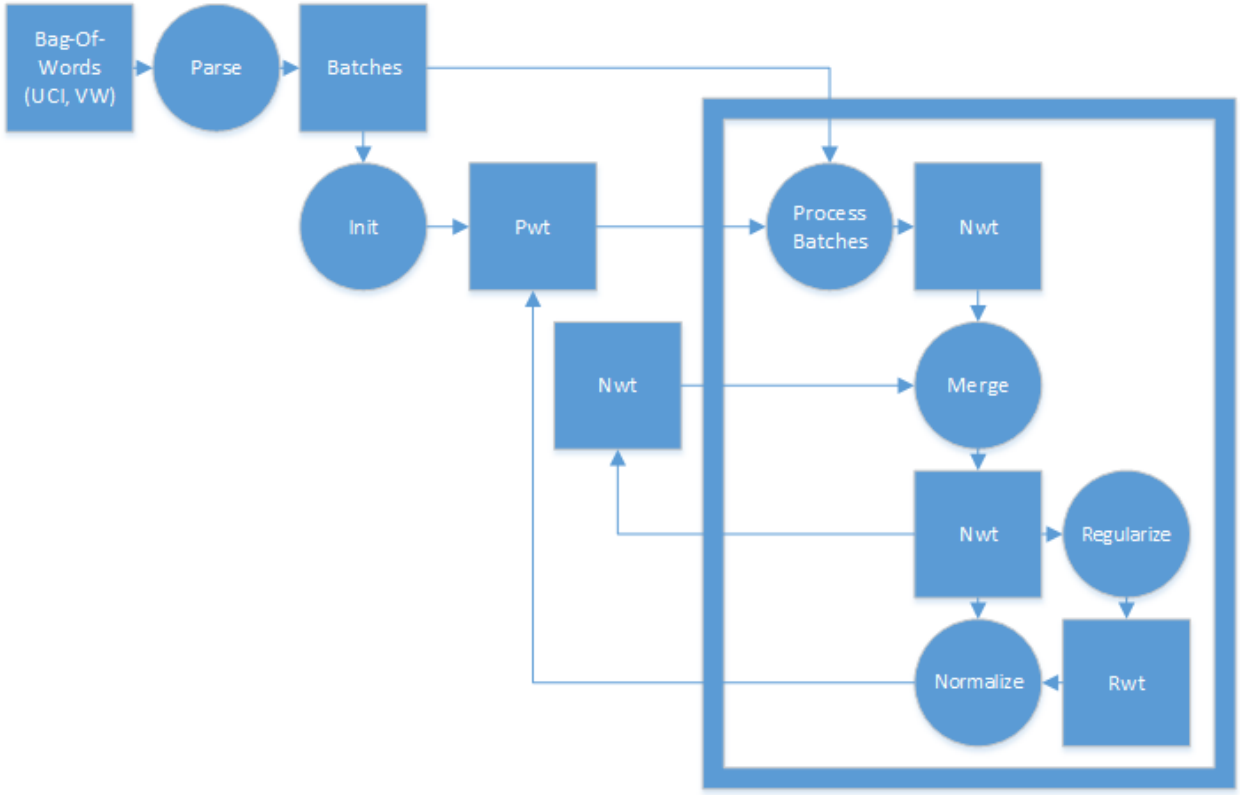
Refer to the following examples that demonstrate usage of new APIs for offline, online and regularized topic modelling:

- example17_process_batches.py
- example18_merge_model.py
- example19_regularize_model.py

Models, tuned with the new API are referred to as *new-style models*, as opposite to *old-style models* inferred with AddBatch, InvokeIteration, WaitIdle and SynchronizeModel APIs.

Warning: For BigARTM v0.7.X we will continue to support old-style models. However, you should consider upgrading to new-style models because old APIs (AddBatch, InvokeIteration, WaitIdle and SynchronizeModel) are likely to be removed in future releases.

The following flow chart gives a typical use-case on new APIs in online regularized algorithm:



Notes on upgrading existing code to new-style models

1. New APIs can only read batches from disk. If your current script passes batches via memory (in AddBatchArgs.batch field) then you need to store batches on disk first, and then process them with ProcessBatches method.
2. Initialize your model as follows:
   - For python_interface: using MasterComponent.InitializeModel method
   - For cpp_interface: using MasterComponent.InitializeModel method
   - For c_interface: using ArtmInitializeModel method

   Remember that you should not create ModelConfig in order to use this methods. Pass your topics_count (or topic_name list) as arguments to InitializeModel method.
3. Learn the difference between Phi and Theta scores, as well as between Phi and Theta regularizes. The following table gives an overview:

Object	Theta	Phi
Scores	• Perplexity • SparsityTheta • ThetaSnippet • ItemsProcessed	• SparsityPhi • TopTokens • TopicKernel
Regularizers	• SmoothSparseTheta	• DecorrelatorPhi • ImproveCoherencePhi • LabelRegularizationPhi • SmoothSparsePhi • SpecifiedSparsePhi

Phi regularizers needs to be calculated explicitly in RegularizeModel, and then applied in NormalizeModel (via optional *rwt* argument). Theta regularizers needs to be enabled in ProcessBatchesArgs. Then they will be automatically calculated and applied during ProcessBatches.

Phi scores can be calculated at any moment based on the new-style model (same as for old-style models). Theta scores can be retrieved in two equivalend ways:

```
pwt_model = "pwt"
master.ProcessBatches(pwt_model, batches, "nwt")
perplexity_score.GetValue(pwt_model).value
```

or

```
pwt_model = "pwt"
process_batches_result = master.ProcessBatches(pwt_model, batches, "nwt")
perplexity_score.GetValue(scores = process_batches_result).value
```

Second way is more explicit. However, the first way allows you to combine aggregate scores accross multiple ProcessBatches calls:

```
pwt_model = "pwt"
master.ProcessBatches(pwt_model, batches1, "nwt")
master.ProcessBatches(pwt_model, batches2, "nwt", reset_scores=False)
perplexity_score.GetValue(pwt_model).value
```

This works because BigARTM caches the result of ProcessBatches together (in association with pwt_model). The *reset_scores* switch disables the default behaviour, which is to reset the cache for pwt_model at the beginning of each ProcessBatch call.

4. Continue using GetThetaMatrix and GetTopicModel to retrieve results from the library. For GetThetaMatrix to work you still need to enable cache_theta in master component. Remember to use the same model in GetThetaMatrix as you used as the input to ProcessBatches. You may also omit "target_nwt" argument in ProcessBatches if you are not interested in getting this output.

```
master.ProcessBatches("pwt", batches)
theta_matrix = master.GetThetaMatrix("pwt")
```

5. Stop using certain APIs:
   - For python_interface: stop using class Model and ModelConfig message
   - For cpp_interface: stop using class Model and ModelConfig message
   - For c_interface: stop using methods ArtmCreateModel, ArtmReconfigureModel, ArtmInvokeIteration, ArtmAddBatch, ArtmWaitIdle, ArtmSynchronizeModel

Notes on models handling (reusing, sharing input and output, etc)

Is allowed to output the result of ProcessBatches, NormalizeModel, RegularizeModel and MergeModel into an existing model. In this case the existing model will be fully overwritten by the result of the operation. For all operations except ProcessBatches it is also allowed to use the same model in inputs and as an output. For example, typical usage of MergeModel involves combining "nwt" and "nwt_hat" back into "nwt". This scenario is fully supported. The output and input of ProcessBatches must refer to two different models. Finally, note that MergeModel will ignore all non-existing models in the input (and log a warning). However, if none of the input models exist then MergeModel will thrown an error.

Known differences

1. Decorrelator regularizer will give slightly different result in the following scenario:

```
master.ProcessBatches("pwt", batches, "nwt")
master.RegularizeModel("pwt", "nwt", "rwt", phi_regularizers)
master.NormalizeModel("nwt", "pwt", "rwt")
```

   To get the same result as from model.Synchronize() adjust your script as follows:

```
master.ProcessBatches("pwt", batches, "nwt")
master.NormalizeModel("nwt", "pwt_temp")
master.RegularizeModel("pwt_temp", "nwt", "rwt", phi_regularizers)
master.NormalizeModel("nwt", "pwt", "rwt")
```

2. You may use GetThetaMatrix(pwt) to retrieve Theta-matrix, previously calculated for new-style models inside ProcessBatches. However, you can not use GetThetaMatrix(pwt, batch) for new models. They do not have corresponding ModelConfig, and as a result you need to go through ProcessBatches to pass all parameters.

7.1.2 Network modus operandi is removed

Network modus operandi had been removed from BigARTM v0.7.0.

This decision had been taken because current implementation struggle from many issues, particularly from poor performance and stability. We expect to re-implement this functionality on top of new-style models.

Please, let us know if this caused issues for you, and we will consider to re-introduce networking in v0.8.0.

7.1.3 Coherence regularizer and scores (experimental)

Refer to example in example16_coherence_score.py.

7.2 BigARTM v0.7.1 Release notes

We are happy to introduce BigARTM v0.7.1, which brings you the following changes:

- BigARTM noteboks — new source of information about BigARTM
- ArtmModel — a brand new Python API
- Much faster retrieval of Phi and Theta matrices from Python
- Much faster dictionary imports from Python
- Auto-detect and use all CPU cores by default
- Fixed Import/Export of topic models (was broken in v0.7.0)

- New capability to implement Phi-regularizers in Python code
- Improvements in Coherence score

Before you upgrade to BigARTM v0.7.1 please review the changes that *break backward compatibility*.

7.2.1 BigARTM notebooks

BigARTM notebooks is your go-to links to read more ideas, examples and other information around BigARTM:

- BigARTM notebooks in English
- BigARTM notebooks in Russian

7.2.2 ArtmModel

Best thing about ArtmModel is that this API had been designed by BigARTM users. Not by BigARTM programmers. This means that BigARTM finally has a nice, clean and easy-to-use programming interface for Python. Don't believe it? Just take a look and some examples:

- ArtmModel examples in English
- ArtmModel examples in Russian

That is cool, right? This new API allows you to load input data from several file formats, infer topic model, find topic distribution for new documents, visualize scores, apply regularizers, and perform many other actions. Each action typically takes one line to write, which allows you to work with BigARTM interactively from Python command line.

ArtmModel exposes most of BigARTM functionality, and it should be sufficiently powerful to cover 95% of all BigARTM use-cases. However, for the most advanced scenarios you might still need to go through the previous API (artm.library). When in doubt which API to use, ask bigartm-users@googlegroups.com — we are there to help!

7.2.3 Coding Phi-regularizers in Python code

This is of course one of those very advanced scenarios where you need to go down to the old API :) Take a look at this example:

- example19_regularize_model
- example20_attach_model

First one tells how to use Phi regularizers, built into BigARTM. Second one provides a new capability to manipulate Phi matrix from Python. We call this **Attach** numpy matrix to the model, because this is similar to attaching debugger (like gdb or Visual Studio) to a running application.

To implement your own Phi regularizer in Python you need to to **attach** to `rwt` model from the first example, and update its values.

7.2.4 Other changes

Fast retrieval of Phi and Theta matrices. In BigARTM v0.7.1 dense Phi and Theta matrices will be retrieved to Python as numpy matrices. All copying work will be done in native C++ code. This is much faster comparing to current solution, where all data is transferred in a large Protobuf message which needs to be deserialized in Python. ArtmModel already takes advantage of this performance improvements.

Fast dictionary import. BigARTM core now supports importing dictionary files from disk, so you no longer have to load them to Python. ArtmModel already take advantage of this performance improvement.

Auto-detect number of CPU cores. You no longer need to specify `num_processors` parameter. By default BigARTM will detect the number of cores on your machine and load all of them. `num_processors` still can be used to limit CPU resources used by BigARTM.

Fixed Import/Export of topic models. Export and Import of topic models will now work. As simple as this:

```
master.ExportModel("pwt", "file_on_disk.model")
master.ImportModel("pwt", "file_on_disk.model")
```

This will also take care of very large models above 1 GB that does not fit into single protobuf message.

Coherence scores. Ask bigartm-users@googlegroups.com if you are interested :)

7.2.5 Breaking changes

- **Changes in Python methods `MasterComponent.GetTopicModel` and `MasterComponent.GetThetaMatrix`**

  From BigARTM v0.7.1 and onwards method `MasterComponent.GetTopicModel` of the low-level Python API will return a tuple, where first argument is of type TopicModel (protobuf message), and second argument is a numpy matrix. TopicModel message will keep all fields as usual, except token_weights field which will became empty. Information from token_weights field had been moved to numpy matrix (rows = tokens, columns = topics).

  Similarly, `MasterComponent.GetThetaMatrix` will also return a tuple, where first argument is of type ThetaMatrix (protobuf message), and second argument is a numpy matrix. ThetaMatrix message will keep all fields as usual, except item_weights field which will became empty. Information from item_weights field had been moved to numpy matrix (rows = items, columns = topics).

  Updated examples:

  - example11_get_theta_matrix.py
  - example12_get_topic_model

  Warning: Use the followign syntax to restore the old behaviour:
  - MasterComponent.GetTopicModel(use_matrix = False)
  - MasterComponent.GetThetaMatrix(use_matrix = False)

  This will return a complete protobuf message, without numpy matrix.

- **Python method ParseCollectionOrLoadDictionary is now obsolete**
  - Use ParseCollection method to convert collection into a set of batches
  - Use MasterComponent.ImportDictionary to load dictionary into BigARTM
  - Updated example: example06_use_dictionaries.py

7.3 BigARTM v0.7.2 Release notes

We are happy to introduce BigARTM v0.7.2, which brings you the following changes:

- Enhancements in high-level python API (`ArtmModel` -> `ARTM`)
- Enhancements in low-level python API (`library.py` -> `master_component.py`)
- Enhancements in CLI interface (`cpp_client`)
- Status and information retrievals from BigARTM

- Allow float token counts (`token_count` -> `token_weight`)
- Allow custom weights for each batch (`ProcessBatchesArgs.batch_weight`)
- Bug fixes and cleanup in the online documentation

7.3.1 Enhancements in Python APIs

Note that `ArtmModel` had been renamed to `ARTM`. The naming conventions follow the same pattern as in scikit learn (e.g. `fit`, `transform` and `fit_transform` methods).

Also note that all input data is now handled by `BatchVectorizer` class.

Refer to noteboods in English and in Russian for further details about `ARTM` interface.

Also note that previous low-level python API `library.py` is superseeded by a new API `master_component.py`. For now both APIs are available, but the old one will be removed in future releases. Refer to this folder for futher examples of the new low-level python API.

Remember that any use of low-level APIs is discouraged. Our recommendation is to always use the high-level python API `ARTM`, and e-mail us know if some functionality is not exposed there.

7.3.2 Enhancements in CLI interface

BigARTM command line interface `cpp_client` had been enhanced with the following options:

- `--load_model` - to load model from file before processing
- `--save_model` - to save the model to binary file after processing
- `--write_model_readable` - to output the model in a human-readable format (CSV)
- `--write_predictions` - to write prediction in a human-readable format (CSV)
- `--dictionary_min_df` - to filter out tokens present in less than N documents / less than P% of documents
- `--dictionary_max_df` - filter out tokens present in less than N documents / less than P% of documents
- `--tau0` - an option of the online algorith, describing the weight parameter in the online update formula. Optional, defaults to `1024`.
- `--kappa` - an option of the online algorithm, describing the exponent parameter in the online update formula. Optional, defaults to `0.7`.

Note that for `--dictionary_min_df` and `--dictionary_max_df` can be treated as number, fraction, percent.

- Use a percentage `%` sign to specify percentage value
- Use a floating value in `[0, 1)` range to specify a fraction
- Use an integer value (`1` or greater) to indicate a number

7.4 BigARTM v0.7.3 Release notes

BigARTM v0.7.3 releases the following changes:

- New command line tool for BigARTM
- Support for classification in bigartm CLI
- Support for asynchronous processing of batches

- Improvements in coherence regularizer and coherence score
- New *TopicMass* score for phi matrix
- Support for documents markup
- New API for importing batches through memory

7.4.1 New command line tool for BigARTM

New CLI is named `bigartm` (or `bigrtm.exe` on Windows), and it supersedes previous CLI named `cpp_client`. New CLI has the following features:

- Parse collection in one of the Formats
- Load dictionary
- Initialize a new model, or import previously created model
- Perform EM-iterations to fit the model
- Export predicted probabilities for all documents into CSV file
- Export model into a file

All command-line options are listed here, and you may see several exampels on BigARTM page at github. At the moment full documentation is only available in Russian.

7.4.2 Support for classification in BigARTM CLI

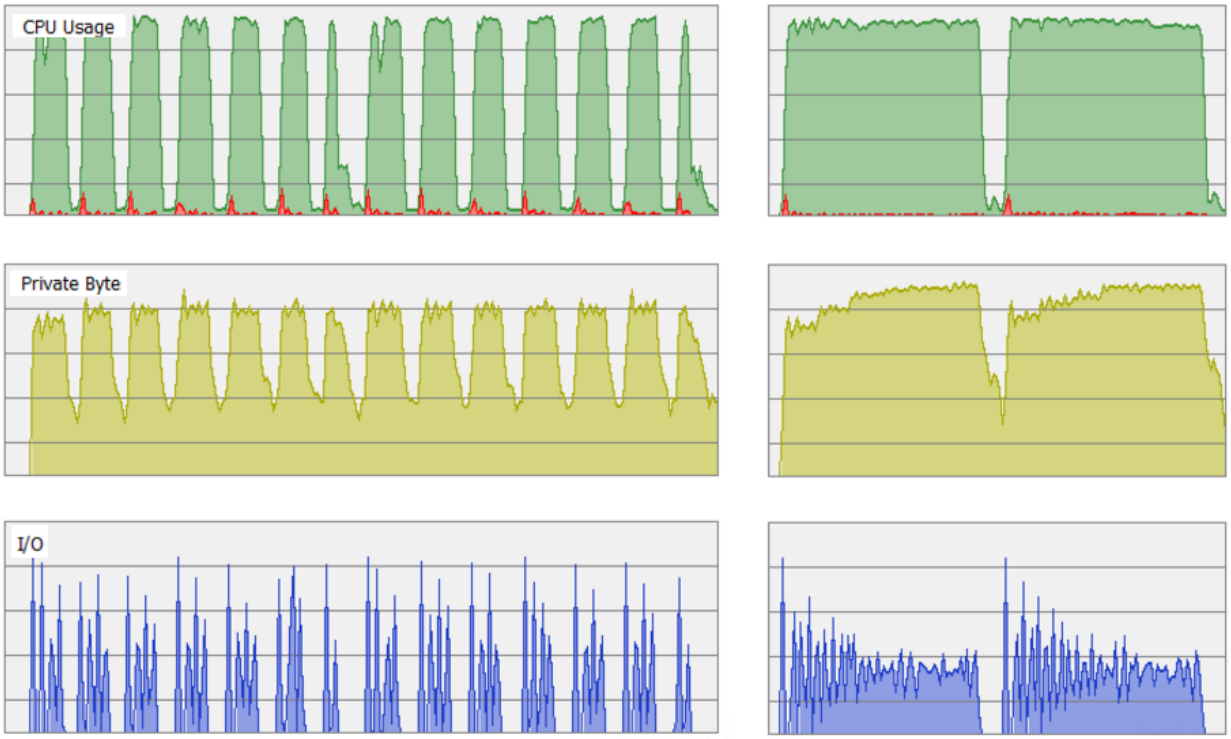
BigARTM CLI is now able to perform classification. The following example assumes that your batches have `target_class` modality in addition to the default modality (`@default_class`).

```
# Fit model
bigartm.exe --use-batches <your batches>
            --use-modality @default_class,target_class
            --topics 50
            --dictionary-min-df 10
            --dictionary-max-df 25%
            --save-model model.bin

# Apply model and output to text files
bigartm.exe --use-batches <your batches>
            --use-modality @default_class,target_class
            --topics 50
            --passes 0
            --load-model model.bin
            --predict-class target_class
            --write-predictions pred.txt
            --write-class-predictions pred_class.txt
            --csv-separator=tab
            --score ClassPrecision
```

7.4.3 Support for asynchronous processing of batches

Asynchronous processing of batches enables applications to overlap EM-iterations better utilize CPU resources. The following chart shows CPU utilization of `bigartm.exe` with (left-hand side) and without async flag (right-hand side).



7.4.4 TopicMass score for phi matrix

Topic mass score calculates cumulated topic mass for each topic. This is a useful metric to monitor balance between topics.

7.4.5 Support for documents markup

Document markup provides topic distribution for each word in a document. Since BigARTM v0.7.3 it is posible to extract this information to use it. A potential application includes color-highlighted maps of the document, where every work is colored according to the most probable topic of the document.

In the code this feature is refered to as `ptdw` matrix. It is possible to extract and regularizer `ptdw` matrices. In future versions it will be also possible to calculate scores based on `ptdw` matrix.

7.4.6 New API for importing batches through memory

New low-level APIs `ArtmImportBatches` and `ArtmDisposeBatches` allow to import batches from memory into BigARTM. Those batches are saved in BigARTM, and can be used for batches processing.

7.5 BigARTM v0.7.4 Release notes

BigARTM v0.7.4 is a big release that includes major rework of dictionaries and MasterModel.

7.5.1 *bigartm/stable* branch

Up until now BigARTM has only one `master` branch, containing the latest code. This branch potentially includes untested code and unfinished features. We are now introducing `bigartm/stable` branch, and encourage all users to stop using `master` and start fetching from `stable`. `stable` branch will be lagging behind `master`, and moved forward to `master` as soon as mainteiners decide that it is ready. At the same point we will introduce a new tag (something like v0.7.3) and produce a new release for Windows. In addition, `stable` branch also might receive small urgent fixes in between releases, typically to address critical issues reported by our users. Such fixes will be also included in `master` branch.

7.5.2 MasterModel

MasterModel is a new set of low-level APIs that allow users of C-interface to infer models and apply them to new data. The APIs are `ArtmCreateMasterModel`, `ArtmReconfigureMasterModel`, `ArtmFitOfflineMasterModel`, `ArtmFitOnlineMasterModel` and `ArtmRequestTransformMasterModel`, togehter with corresponding protobuf messages. For a usage example see `src/bigartm/srcmain.cc`.

This APIs should be easy to understand for the users who are familiar with Python interface. Basically, we take `ARTM` class in Python, and push it down to the core. Now users can create their model via `MasterModelConfig` (protobuf message), fit via `ArtmFitOfflineMasterModel` or `ArtmFitOnlineMasterModel`, and apply to the new data via `ArtmRequestTransformMasterModel`. This means that the user no longer has to orchestrate low-level building blocks such as `ArtmProcessBatches`, `ArtmMergeModel`, `ArtmRegularizeModel` and `ArtmNormalizeModel`.

`ArtmCreateMasterModel` is similar to `ArtmCreateMasterComponent` in a sence that it returns `master_id`, which can be later passed to all other APIs. This mean that most APIs will continue working as before. This applies to `ArtmRequestThetaMatrix`, `ArtmRequestTopicModel`, `ArtmRequestScore`, and many others.

7.5.3 Rework of dictionaries

Previous implementation of the dictionaries was really messy, and we are trying to clean this up. This effort is not finished yet, however we decided to release current version because it is a major improvement comparing to the previous version. At the low-level (`c_interface`), we now have the following methods to work with dictionaries:

- `ArtmGatherDictionary` collects a dictionary based on a folder with batches,
- `ArtmFilterDictionary` filter tokens from the dictinoary based on their term frequency or document frequency,
- `ArtmCreateDictionary` creates a dictionary from a custom `DictionaryData` object (protobuf message),
- `ArtmRequestDictionary` retrieves a dictionary as `DictionaryData` object (protobuf message),
- `ArtmDisposeDictionary` deletes dictionary object from BigARTM,
- `ArtmImportDictionary` import dictionary from binary file,
- `ArtmExportDictionary` expor tdictionary into binary file.

All dictionaries are identified by a string ID (`dictionary_name`). Dictionaries can be used to initialize the model, in regularizers or in scores.

Note that `ArtmImportDictionary` and `ArtmExportDictionary` now uses a different format. For this reason we require that all imported or exported files end with `.dict` extension. This limitation is only introduced to make users aware of the change in binary format.

Warning: Please note that you have to re-generate all dictionaries, created in previous BigARTM versions. To force this limitation we decided that `ArtmImportDictionary` and `ArtmExportDictionary` will require all imported or exported files end with `.dict` extension. This limitation is only introduced to make users aware of the change in binary format.
Please note that in the next version (*BigARTM v0.8.0*) we are planing to break dictionary format once again. This is because we will introduce `boost.serialize` library for all import and export methods. From that point `boost.serialize` library will allow us to upgrade formats without breaking backwards compatibility.

The following example illustrate how to work with new dictionaries from Python.

```
# Parse collection in UCI format from D:\Datasets\docword.kos.txt and D:\Datasets\vocab.kos.txt
# and store the resulting batches into D:\Datasets\kos_batches
batch_vectorizer = artm.BatchVectorizer(data_format='bow_uci',
                                        data_path=r'D:\Datasets',
                                        collection_name='kos',
                                        target_folder=r'D:\Datasets\kos_batches')

# Initialize the model. For now dictionaries exist within the model,
# but we will address this in the future.
model = artm.ARTM(...)


# Gather dictionary named `dict` from batches.
# The resulting dictionary will contain all distinct tokens that occur
# in those batches, and their term frequencies
model.gather_dictionary("dict", "D:\Datasets\kos_batches")

# Filter dictionary by removing tokens with too high or too low term frequency
# Save the result as `filtered_dict`"
model.filter_dictionary(dictionary_name='dict',
                        dictionary_target_name='filtered_dict',
                        min_df=10, max_df_rate=0.4)

# Initialize model from `diltered_dict`
model.initialize("filtered_dict")

# Import/export functionality
model.save_dictionary("filtered_dict", "D:\Datasets\kos.dict")
model.load_dictionary("filtered_dict2",  "D:\Datasets\kos.dict")
```

7.5.4 Changes in the infrastructure

- Static linkage for bigartm command-line executable on Linux. To disable static linkage use `cmake -DBUILD_STATIC_BIGARTM=OFF ..`
- Install BigARTM python API via `python setup.py install`

7.5.5 Changes in core functionality

- Custom transform function for KL-div regularizers
- Ability to initialize the model with custom seed
- `TopicSelection` regularizers
- `PeakMemory` score (Windows only)

- Different options to name batches when parsing collection (`GUID` as today, and `CODE` for sequential numbering)

7.5.6 Changes in Python API

- `ARTM.dispose()` method for managing native memory
- `ARTM.get_info()` method to retrieve internal state
- Performance fixes
- Expose class prediction functionality

7.5.7 Changes in C++ interface

- Consume `MasterModel` APIs in C++ interface. Going forward this is the only C++ interface that we will support.

7.5.8 Changes in console interface

- Better options to work with dictionaries
- `--write-dictionary-readable` to export dictionary
- `--force` switch to let user overwrite existing files
- `--help` generates much better examples
- `--model-v06` to experiment with old APIs (`ArtmInvokeIteration` / `ArtmWaitIdle` / `ArtmSynchronizeModel`)
- `--write-scores` switch to export scores into file
- `--time-limit` option to time-box model inference(as an alternative to `--passes` switch)

CHAPTER 8

Publications

- Vorontsov K., Potapenko A., Plavin A. Additive Regularization of Topic Models for Topic Selection and Sparse Factorization. // Statistical Learning and Data Sciences. 2015 — pp. 193-202. PDF in English.
- 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. Slides in English.
- Vorontsov K. V. Additive Regularization for Topic Models of Text Collections // Doklady Mathematics. 2014, Pleiades Publishing, Ltd. — Vol. 89, No. 3, pp. 301–304. PDF in English, PDF in Russian.
- Vorontsov K. V., Potapenko A. A. Tutorial on Probabilistic Topic Modeling: Additive Regularization for Stochastic Matrix Factorization // AIST'2014, Analysis of Images, Social networks and Texts. Springer International Publishing Switzerland, 2014. Communications in Computer and Information Science (CCIS). Vol. 436. pp. 29–46. PDF in English.
- Vorontsov K. V., Potapenko A. A. Additive Regularization of Topic Models // Machine Learning Journal, Special Issue "Data Analysis and Intelligent Optimization", Springer, 2014. PDF in English, PDF in Russian.

CHAPTER 9

Legacy documentation pages

Legacy pages are kept to preserve existing user's links (favourites in browser, etc).

9.1 Typical python example

This page is obsolete, please use the high-level API described in ARTM notebook (in Russian or in English).

9.1.1 Examples of low-level API

Folder `C:\BigARTM\python\examples` contains several toy examples:

- example01_synthetic_collection.py
- example02_parse_collection.py
- example03_concurrency.py
- example04_online_algorithm.py
- example05_train_and_test_stream.py
- example06_use_dictionaries.py
- example09_regularizers.py
- example10_multimodal.py
- example11_get_theta_matrix.py
- example12_get_topic_model.py
- example13_overwrite_topic_model.py
- example14_initialize_topic_model.py
- example15_import_export_topic_model.py
- example17_process_batches.py
- example18_merge_model.py
- example19_regularize_model.py
- example20_attach_model.py

All examples does not have any parameters, and you may run them without arguments:

```
C:\BigARTM\python\examples>python example02_parse_collection.py

 No batches found, parsing them from textual collection...  OK.
 Iter#0 : Perplexity = 6885.223 , Phi sparsity = 0.050 , Theta sparsity = 0.012
 Iter#1 : Perplexity = 2409.510 , Phi sparsity = 0.113 , Theta sparsity = 0.063
 Iter#2 : Perplexity = 2075.445 , Phi sparsity = 0.203 , Theta sparsity = 0.174
 Iter#3 : Perplexity = 1855.196 , Phi sparsity = 0.293 , Theta sparsity = 0.261
 Iter#4 : Perplexity = 1728.749 , Phi sparsity = 0.370 , Theta sparsity = 0.302
 Iter#5 : Perplexity = 1661.044 , Phi sparsity = 0.429 , Theta sparsity = 0.317
 Iter#6 : Perplexity = 1621.851 , Phi sparsity = 0.475 , Theta sparsity = 0.327
 Iter#7 : Perplexity = 1596.965 , Phi sparsity = 0.511 , Theta sparsity = 0.331

 Top tokens per topic:
 Topic#1: poll(0.05) iraq(0.04) people(0.02) news(0.02) john(0.01) media(0.01)
 Topic#2: republican(0.02) party(0.02) state(0.02) general(0.01) democrats(0.01)
 Topic#3: dean(0.04) edwards(0.02) percent(0.02) primary(0.02) clark(0.02)
 Topic#4: forces(0.01) baghdad(0.01) iraqis(0.01) coburn(0.01) carson(0.01)
 Topic#5: military(0.01) officials(0.01) intelligence(0.01) american(0.01)
 Topic#6: electoral(0.04) labor(0.02) culture(0.02) exit(0.02) scoop(0.01)
 Topic#7: law(0.01) court(0.01) marriage(0.01) gay(0.01) amendment(0.01)
 Topic#8: president(0.03) administration(0.02) campaign(0.01) million(0.01)
 Topic#9: years(0.01) ballot(0.01) rights(0.01) nader(0.01) life(0.01)
 Topic#10: house(0.08) war(0.03) republicans(0.02) voting(0.02) vote(0.02)

 Snippet of theta matrix:
 Item#3000: 0.432 0.507 0.059 0.000 0.000 0.000 0.000 0.000 0.002 0.000
 Item#2991: 0.249 0.382 0.269 0.000 0.000 0.025 0.016 0.034 0.000 0.026
 Item#2992: 0.000 0.001 0.000 0.000 0.000 0.000 0.000 0.851 0.000 0.147
 Item#2993: 0.358 0.058 0.030 0.141 0.152 0.000 0.002 0.248 0.000 0.010
 Item#2994: 0.051 0.142 0.056 0.000 0.000 0.146 0.000 0.000 0.000 0.604
 Item#2995: 0.004 0.593 0.000 0.000 0.128 0.005 0.168 0.040 0.030 0.033
 Item#2996: 0.069 0.063 0.054 0.000 0.000 0.107 0.008 0.004 0.000 0.696
 Item#2997: 0.000 0.194 0.000 0.000 0.043 0.000 0.471 0.228 0.062 0.002
 Item#2998: 0.026 0.085 0.042 0.001 0.180 0.000 0.146 0.485 0.022 0.012
 Item#2999: 0.312 0.547 0.099 0.000 0.000 0.004 0.008 0.017 0.013 0.000
```

This simple example loads a text collection from disk and uses iterative scans over the collection to infer a topic model. Then it outputs top words in each topic and topic distributions of last processed documents. For further information about this example refer to Typical python example.

9.1.2 Parse collection step

The following python script parses `docword.kos.txt` and `vocab.kos.txt` files and converts them into a set of binary-serialized *batches*, stored on disk. In addition the script creates a *dictionary* with all unique tokens in the collection and stored it on disk. The script also detects if it had been already executed, and in this case it just loads the dictionary and save it in *unique_tokens* variable.

The same logic is implemented in a helper-method *`ParseCollectionOrLoadDictionary`* method.

```
data_folder = sys.argv[1] if (len(sys.argv) >= 2) else ''
target_folder = 'kos'
collection_name = 'kos'

batches_found = len(glob.glob(target_folder + "/*.batch"))
if batches_found == 0:
  print "No batches found, parsing them from textual collection...",
  parser_config = artm.messages_pb2.CollectionParserConfig();
```

```
  parser_config.format = artm.library.CollectionParserConfig_Format_BagOfWordsUci

  parser_config.docword_file_path = data_folder + 'docword.'+ collection_name + '.txt'
  parser_config.vocab_file_path = data_folder + 'vocab.'+ collection_name + '.txt'
  parser_config.target_folder = target_folder
  parser_config.dictionary_file_name = 'dictionary'
  unique_tokens = artm.library.Library().ParseCollection(parser_config);
  print " OK."
else:
  print "Found " + str(batches_found) + " batches, using them."
  unique_tokens  = artm.library.Library().LoadDictionary(target_folder + '/dictionary');
```

You may also download larger collections from the following links. You can get the original collection (docword file and vocab file) or an already precompiled batches and dictionary.

9.1.3 MasterComponent

Master component is you main entry-point to all BigARTM functionality. The following script creates master component and configures it with several regularizers and score calculators.

```
with artm.library.MasterComponent(disk_path = target_folder) as master:
  perplexity_score     = master.CreatePerplexityScore()
  sparsity_theta_score = master.CreateSparsityThetaScore()
  sparsity_phi_score   = master.CreateSparsityPhiScore()
  top_tokens_score     = master.CreateTopTokensScore()
  theta_snippet_score  = master.CreateThetaSnippetScore()

  dirichlet_theta_reg  = master.CreateDirichletThetaRegularizer()
  dirichlet_phi_reg    = master.CreateDirichletPhiRegularizer()
  decorrelator_reg     = master.CreateDecorrelatorPhiRegularizer()
```

Master component must be configured with a disk path, which should contain a set of batches produced in the previous step of this tutorial.

Score calculators allows you to retrieve important quality measures for your topic model. Perplexity, sparsity of theta and phi matrices, lists of tokens with highest probability within each topic are all examples of such scores. By default BigARTM does not calculate any scores, so you have to create in master component. The same is true for regularizers, that allow you to customize your topic model.

For further details about master component refer to *MasterComponentConfig*.

9.1.4 Configure Topic Model

Topic model configuration defins the number of topics in the model, the list of scores to be calculated, and the list of regularizers to apply to the model. For further details about model configuration refer to *ModelConfig*.

```
model = master.CreateModel(topics_count = 10, inner_iterations_count = 10)
model.EnableScore(perplexity_score)
model.EnableScore(sparsity_phi_score)
model.EnableScore(sparsity_theta_score)
model.EnableScore(top_tokens_score)
model.EnableScore(theta_snippet_score)
model.EnableRegularizer(dirichlet_theta_reg, -0.1)
model.EnableRegularizer(dirichlet_phi_reg, -0.2)
model.EnableRegularizer(decorrelator_reg, 1000000)
model.Initialize(unique_tokens)     # Setup initial approximation for Phi matrix.
```

Note that on the last step we configured the initial approximation of Phi matrix. This step is optional — BigARTM is able to collect all tokens dynamically during first scan of the collection. However, a deterministic initial approximation helps to reproduce the same results from run to run.

9.1.5 Invoke Iterations

The following script performs several scans over the set of batches. Depending on the size of the collection this step might be quite time-consuming. It is good idea to output some information after every step.

```
for iter in range(0, 8):
  master.InvokeIteration(1)         # Invoke one scan of the entire collection...
  master.WaitIdle();                # and wait until it completes.
  model.Synchronize();              # Synchronize topic model.
  print "Iter#" + str(iter),
  print ": Perplexity = %.3f" % perplexity_score.GetValue(model).value,
  print ", Phi sparsity = %.3f" % sparsity_phi_score.GetValue(model).value,
  print ", Theta sparsity = %.3f" % sparsity_theta_score.GetValue(model).value
```

If your collection is very large you may want to utilize online algorithm that updates topic model several times during each iteration, as it is demonstrated by the following script:

```
master.InvokeIteration(1)         # Invoke one scan of the entire collection...
while True:
  done = master.WaitIdle(100)     # wait 100 ms
  model.Synchronize(0.9)          # decay weights in current topic model by 0.9,
  if (done):                      # append all increments and invoke all regularizers.
        break;
```

9.1.6 Retrieve and visualize scores

Finally, you are interested in retrieving and visualizing all collected scores.

```
artm.library.Visualizers.PrintTopTokensScore(top_tokens_score.GetValue(model))
artm.library.Visualizers.PrintThetaSnippetScore(theta_snippet_score.GetValue(model))
```

9.2 Enabling Basic BigARTM Regularizers

This paper describes the experiment with topic model regularization in BigARTM library using experiment02_artm.py. The script provides the possibility to learn topic model with three regularizers (sparsing Phi, sparsing Theta and pairwise topic decorrelation in Phi). It also allows the monitoring of learning process by using quality measures as hold-out perplexity, Phi and Theta sparsity and average topic kernel characteristics.

> **Warning:** Note that perplexity estimation can influence the learning process in the online algorithm, so we evaluate perplexity only once per 20 synchronizations to avoid this influence. You can change the frequency using `test_every` variable.

We suggest you to have BigARTM installed in `$YOUR_HOME_DIRECTORY`. To proceed the experiment you need to execute the following steps:

1. Download the collection, represented as BigARTM batches:
   - https://s3-eu-west-1.amazonaws.com/artm/enwiki-20141208_1k.7z
   - https://s3-eu-west-1.amazonaws.com/artm/enwiki-20141208_10k.7z

This data represents a complete dump of the English Wikipedia (approximately 3.7 million documents). The size of one batch in first version is 1000 documents and 10000 in the second one. We used 10000. The decompressed folder with batches should be put into `$YOUR_HOME_DIRECTORY`. You also need to move there the dictionary file from the batches folder.

The batch, you'd like to use for hold-out perplexity estimation, also must be placed into `$YOUR_HOME_DIRECTORY`. In our experiment we used the batch named `243af5b8-beab-4332-bb42-61892df5b044.batch`.

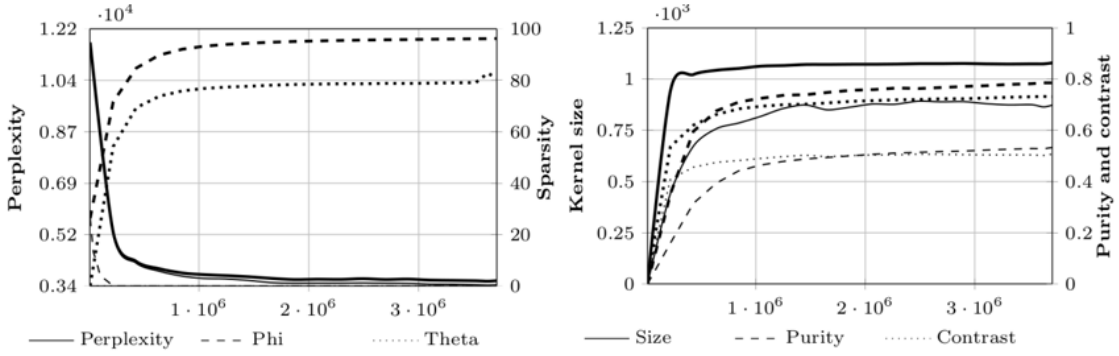
2. The next step is the script preparation. Open it's code and find the declaration(-s) of variable(-s)
   - `home_folder` (line 8) and assign it the path `$YOUR_HOME_DIRECTORY`;
   - `batch_size` (line 28) and assign it the chosen size of batch;
   - `batches_disk_path` (line 36) and replace the string 'wiki_10k' with the name of your directory with batches;
   - `test_batch_name` (line 43) and replace the string with direct batch's name with the name of your test batch;
   - `tau_decor`, `tau_phi` and `tau_theta` (lines 57-59) and substitute the values you'd like to use.
3. If you want to estimate the final perplexity on another, larger test sample, put chosen batches into test folder (in `$YOUR_HOME_DIRECTORY` directory). Then find in the code of the script the declaration of variable `save_and_test_model` (line 30) and assign it `True`.
4. After all launch the script. Current measures values will be printed into console. Note, that after synchronizations without perplexity estimation it's value will be replaced with string 'NO'. The results of synchronizations with perplexity estimation in addition will be put in corresponding files in results folder. The file format is general for all measures: the set of strings «(accumulated number of processed documents, measure value)»:

```
(10000, 0.018)
(220000, 0.41)
(430000, 0.456)
(640000, 0.475)
...
```

These files can be used for plot building.

If desired, you can easy change values of any variable in the code of script since it's sense is clearly commented. If you used all parameters and data identical our experiment you should get the results, close to these ones

Model/Functional	$\mathcal{P}_{10k}$	$\mathcal{P}_{100k}$	$\mathcal{S}_\Phi$	$\mathcal{S}_\Theta$	$\mathcal{K}_s$	$\mathcal{K}_p$	$\mathcal{K}_c$
LDA	3436	3801	0.0	0.0	873	0.533	0.507
ARTM	3577	3947	96.3	80.9	1079	0.785	0.731

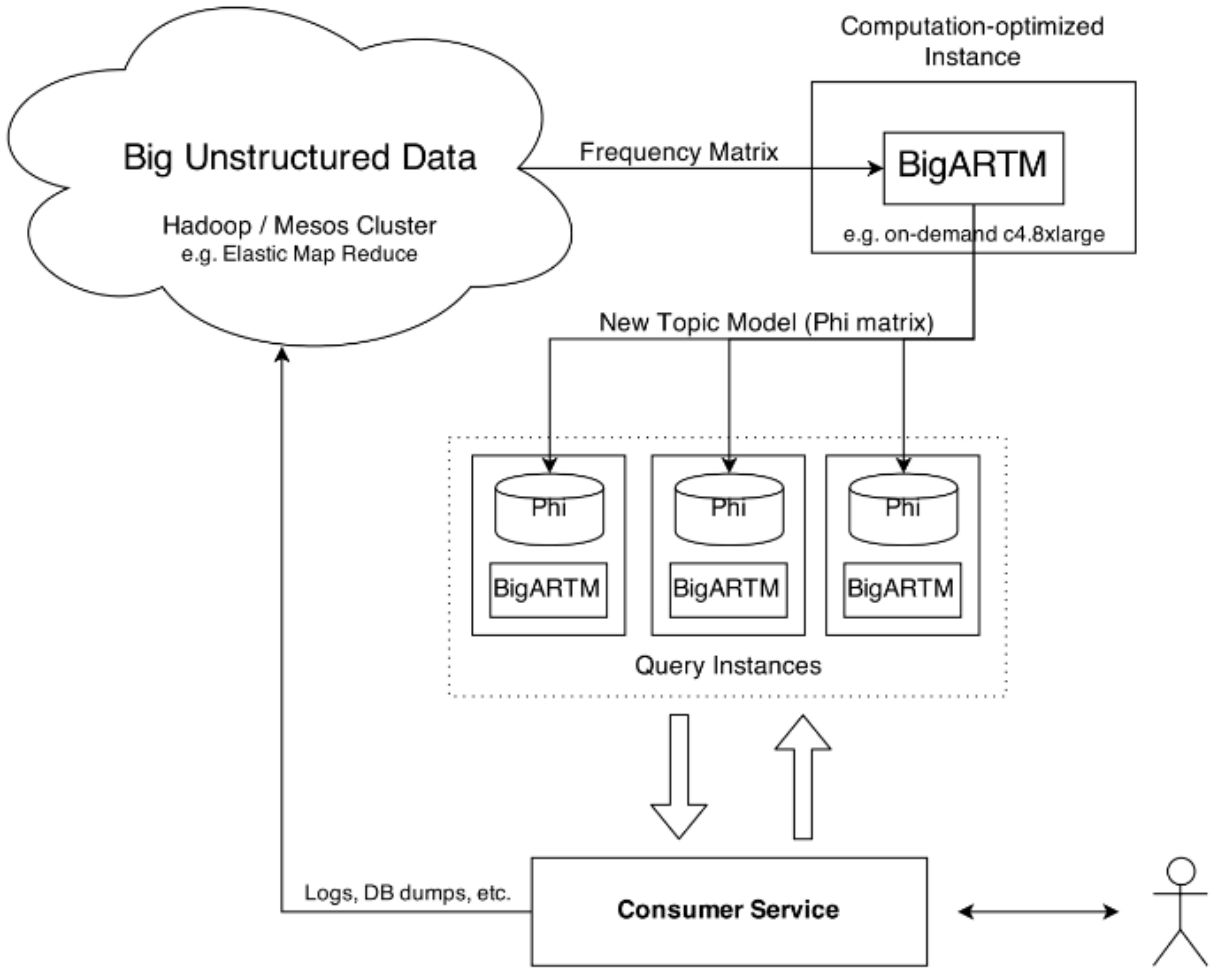


Here you can see the results of comparison between ARTM and LDA models. To make the experiment with LDA instead of ARTM you only need to change the values of variables tau_decor, tau_phi and tau_theta to 0, 1 / topics_count and 1 / topics_count respectively and run the script again.

Warning: Note, that we used machine with 8 cores and 15 Gb RAM for our experiment.

9.3 BigARTM as a Service

The following diagram shows a suggested topology for a query service that involve topic modelling on Big Data.



Here the main use for Hadoop / MapReduce is to process your Big Unstructured Data into a compact bag-of-words representation. Due to out-of-core design and extreme performance BigARTM will be able to handle this data on a single compute-optimized node. The resulting topic model should be replicated on all query instances that serve user requests.

To avoid query-time dependency on BigARTM component you may want to infer topic distributions `theta_{td}` for new documents in your code. This can be done as follows. Start from uniform topic assigment `theta_{td} = 1 / |T|` and update it in the following loop:

initialize θ_{td} for all $t \in T$;
repeat
  $Z_w := \sum_{t \in T} \phi_{wt}^{i-1} \theta_{td}$ for all $w \in d$;
  $\theta_{td} := \frac{1}{n_d} \sum_{w \in d} n_{dw} \phi_{wt}^{i-1} \theta_{td} / Z_w$ for all $t \in T$;
until θ_d converges;

where `n_dw` is the number of word `w` occurences in document `d`, `phi_wt` is an element of the Phi matrix. In BigARTM the loop is repeated *`ModelConfig.inner_iterations_count`* times (defaulst to `10`). To precisely replicate BigARTM behavior one needs to account for class weights and include regularizers. Please contact us if you need more details.

9.4 BigARTM: The Algorithm Under The Hood

ToDo: link BigARTM to online batch PLSA algorithm.

ToDo: explain the notation in the algorithm.

ToDo: update the algortihm with regularization.

Algorithm 1 BigARTM's algorithm

1: Initialize ϕ_{wt}^0 for all $w \in W$ and $t \in T$;
2: **for all** $i = 1, \ldots, I$ **do**
3: $n_{wt}^i := 0, n_t^i := 0$ for all $w \in W$ and $t \in T$;
4: **for all** batches D_j, j = 1,...,J **do**
5: $\tilde{n}_{wt} := 0, \tilde{n}_t := 0$ for all $w \in W$ and $t \in T$;
6: **for all** $d \in D_j$ **do**
7: initialize θ_{td} for all $t \in T$;
8: **repeat**
9: $Z_w := \sum_{t \in T} \phi_{wt}^{i-1} \theta_{td}$ for all $w \in d$;
10: $\theta_{td} := \frac{1}{n_d} \sum_{w \in d} n_{dw} \phi_{wt}^{i-1} \theta_{td} / Z_w$ for all $t \in T$;
11: **until** θ_d converges;
12: increment $\tilde{n}_{wt}, \tilde{n}_t$ by $n_{dw} \phi_{wt}^{i-1} \theta_{td} / Z_w$ for all $w \in W$ and $t \in T$;
13: $n_{wt}^i := n_{wt}^i + \tilde{n}_{wt}^i$ for all $w \in W$ and $t \in T$;
14: $n_t^i := n_t^i + \tilde{n}_t$ for all $t \in T$;
15: $\phi_{wt}^i := \frac{n_{wt}^i}{n_t^i}$ for all $w \in W$ and $t \in T$;

In this algorithm most CPU resources are consumed on steps 8-11 to infer topic distribution for each document. This operation can be executed concurrently across documents or batches. In BigARTM this parallelization is done across batches to avoid splitting the work into too small junks.

Processing each batch produces counters $tilde n_{wt}$ and $tilde n_{t}$, which should be then merged with the corresponding counters coming from other batches. Since this information is produced by multiple concurrent threads the merging process should be thread-safe and properly synchronised. Our solution is to store all counters $tilde

n_{wt}$ and $tilde n_{t}$ into a single queue, from where they can be picked up by a single *merger thread*. This thread will then accumulate the counters without any locking.

Further in this text the term *outer iteration loop* stands for the loop at the step 2, and the term emph{inner iteration loop} stands for the loop at step 8. Instead of "repeat until it converges" criteria current implementation uses a fixed number of iterations, which is configured manually by the user.

Step 15 is incorporated into all steps that require phi_{wt} (e.g. into steps 9, 10 and 11). These steps utilize counters from the previous iteration (n^{i-1}_wt and n^{i-1}_t), which are no longer updated by the merger thread, hence they represent read-only data and can be accessed from multiple threads without any synchronization. At the same time the merger thread will accumulate counters for n^i_{wt} and n^i_t for the current iteration, again in a lock-free manner.

9.5 Messages

This document explains all protobuf messages that can be transfered between the user code and BigARTM library.

> **Warning:** Remember that all fields is marked as *optional* to enhance backwards compatibility of the binary protobuf format. Some fields will result in run-time exception when not specified. Please refer to the documentation of each field for more details.

Note that we discourage any usage of fields marked as *obsolete*. Those fields will be removed in future releases.

9.5.1 DoubleArray

class messages_pb2.**DoubleArray**

Represents an array of double-precision floating point values.

```
message DoubleArray {
  repeated double value = 1 [packed = true];
}
```

9.5.2 FloatArray

class messages_pb2.**FloatArray**

Represents an array of single-precision floating point values.

```
message FloatArray {
  repeated float value = 1 [packed = true];
}
```

9.5.3 BoolArray

class messages_pb2.**BoolArray**

Represents an array of boolean values.

```
message BoolArray {
  repeated bool value = 1 [packed = true];
}
```

9.5.4 IntArray

class messages_pb2.**IntArray**

Represents an array of integer values.

```
message IntArray {
  repeated int32 value = 1 [packed = true];
}
```

9.5.5 Item

class messages_pb2.**Item**

Represents a unit of textual information. A typical example of an item is a document that belongs to some text collection.

```
message Item {
  optional int32 id = 1;
  repeated Field field = 2;
  optional string title = 3;
}
```

Item.**id**
An integer identifier of the item.

Item.**field**
A set of all fields withing the item.

Item.**title**
An optional title of the item.

9.5.6 Field

class messages_pb2.**Field**

Represents a field withing an item. The idea behind fields is that each item might have its title, author, body, abstract, actual text, links, year of publication, etc. Each of this entities should be represented as a Field. The topic model defines how those fields should be taken into account when BigARTM infers a topic model. Currently each field is represented as "bag-of-words" — each token is listed together with the number of its occurrences. Note that each Field is always part of an Item, Item is part of a Batch, and a batch always contains a list of tokens. Therefore, each Field just lists the indexes of tokens in the Batch.

```
message Field {
  optional string name = 1 [default = "@body"];
  repeated int32 token_id = 2;
  repeated int32 token_count = 3;
  repeated int32 token_offset = 4;

  optional string string_value = 5;
  optional int64 int_value = 6;
  optional double double_value = 7;
  optional string date_value = 8;

  repeated string string_array = 16;
  repeated int64 int_array = 17;
  repeated double double_array = 18;
```

```
  repeated string date_array = 19;
}
```

9.5.7 Batch

class messages_pb2.**Batch**

Represents a set of items. In BigARTM a batch is never split into smaller parts. When it comes to concurrency this means that each batch goes to a single processor. Two batches can be processed concurrently, but items in one batch are always processed sequentially.

```
message Batch {
  repeated string token = 1;
  repeated Item item = 2;
  repeated string class_id = 3;
  optional string description = 4;
  optional string id = 5;
}
```

Batch.**token**
: A set value that defines all tokens than may appear in the batch.

Batch.**item**
: A set of items of the batch.

Batch.**class_id**
: A set of values that define for classes (modalities) of tokens. This repeated field must have the same length as *token*. This value is optional, use an empty list indicate that all tokens belong to the default class.

Batch.**description**
: An optional text description of the batch. You may describe for example the source of the batch, preprocessing technique and the structure of its fields.

Batch.**id**
: Unique identifier of the batch in a form of a GUID (example: 4fb38197-3f09-4871-9710-392b14f00d2e). This field is required.

9.5.8 Stream

class messages_pb2.**Stream**

Represents a configuration of a stream. Streams provide a mechanism to split the entire collection into virtual subsets (for example, the 'train' and 'test' streams).

```
message Stream {
  enum Type {
    Global = 0;
    ItemIdModulus = 1;
  }

  optional Type type = 1 [default = Global];
  optional string name = 2 [default = "@global"];
  optional int32 modulus = 3;
  repeated int32 residuals = 4;
}
```

Stream.**type**
A value that defines the type of the stream.

Global	Defines a stream containing all items in the collection.
ItemIdModulus	Defines a stream containing all items with ID that matches modulus and residuals. An item belongs to the stream iff the modulo reminder of item ID is contained in the residuals field.

Stream.**name**
A value that defines the name of the stream. The name must be unique across all streams defined in the master component.

9.5.9 MasterComponentConfig

class messages_pb2.**MasterComponentConfig**

Represents a configuration of a master component.

```
message MasterComponentConfig {
  optional string disk_path = 2;
  repeated Stream stream = 3;
  optional bool compact_batches = 4 [default = true];
  optional bool cache_theta = 5 [default = false];
  optional int32 processors_count = 6 [default = 1];
  optional int32 processor_queue_max_size = 7 [default = 10];
  optional int32 merger_queue_max_size = 8 [default = 10];
  repeated ScoreConfig score_config = 9;
  optional bool online_batch_processing = 13 [default = false];  // obsolete in BigARTM v0.5.8
  optional string disk_cache_path = 15;
}
```

MasterComponentConfig.**disk_path**
A value that defines the disk location to store or load the collection.

MasterComponentConfig.**stream**
A set of all data streams to configure in master component. Streams can overlap if needed.

MasterComponentConfig.**compact_batches**
A flag indicating whether to compact batches in AddBatch() operation. Compaction is a process that shrinks the dictionary of each batch by removing all unused tokens.

MasterComponentConfig.**cache_theta**
A flag indicating whether to cache theta matrix. Theta matrix defines the discrete probability distribution of each document across the topics in topic model. By default BigARTM infers this distribution every time it processes the document. Option 'cache_theta' allows to cache this theta matrix and re-use theha values when the same document is processed on the next iteration. This option must be set to 'true' before calling method *ArtmRequestThetaMatrix()*.

MasterComponentConfig.**processors_count**
A value that defines the number of concurrent processor components. The number of processors should normally not exceed the number of CPU cores.

MasterComponentConfig.**processor_queue_max_size**
A value that defines the maximal size of the processor queue. Processor queue contains batches, prefetch from disk into memory. Recommendations regarding the maximal queue size are as follows:

- the queue size should be at least as large as the number of concurrent processors;

MasterComponentConfig.**merger_queue_max_size**
A value that defines the maximal size of the merger queue. Merger queue size contains an incremental updates of topic model, produced by processor components. Try reducing this parameter if BigARTM consumes too much memory.

MasterComponentConfig.**score_config**
A set of all scores, available for calculation.

MasterComponentConfig.**online_batch_processing**
Obsolete in BigARTM v0.5.8.

MasterComponentConfig.**disk_cache_path**
A value that defines a writtable disk location where this master component can store some temporary files. This can reduce memory usage, particularly when *cache_theta* option is enabled. Note that on clean shutdown master component will will be cleaned this folder automatically, but otherwise it is your responsibility to clean this folder to avoid running out of disk.

9.5.10 ModelConfig

class messages_pb2.**ModelConfig**

Represents a configuration of a topic model.

```
message ModelConfig {
  optional string name = 1 [default = "@model"];
  optional int32 topics_count = 2 [default = 32];
  repeated string topic_name = 3;
  optional bool enabled = 4 [default = true];
  optional int32 inner_iterations_count = 5 [default = 10];
  optional string field_name = 6 [default = "@body"];  // obsolete in BigARTM v0.5.8
  optional string stream_name = 7 [default = "@global"];
  repeated string score_name = 8;
  optional bool reuse_theta = 9 [default = false];
  repeated string regularizer_name = 10;
  repeated double regularizer_tau = 11;
  repeated string class_id = 12;
  repeated float class_weight = 13;
  optional bool use_sparse_bow = 14 [default = true];
  optional bool use_random_theta = 15 [default = false];
  optional bool use_new_tokens = 16 [default = true];
  optional bool opt_for_avx = 17 [default = true];
}
```

ModelConfig.**name**
A value that defines the name of the topic model. The name must be unique across all models defined in the master component.

ModelConfig.**topics_count**
A value that defines the number of topics in the topic model.

ModelConfig.**topic_name**
A repeated field that defines the names of the topics. All topic names must be unique within each topic model. This field is optional, but either *topics_count* or *topic_name* must be specified. If both specified, then *topics_count* will be ignored, and the number of topics in the model will be based on the length of *topic_name* field. When *topic_name* is not specified the names for all topics will be autogenerated.

ModelConfig.**enabled**
A flag indicating whether to update the model during iterations.

ModelConfig.**inner_iterations_count**
A value that defines the fixed number of iterations, performed to infer the theta distribution for each document.

ModelConfig.**field_name**
Obsolete in BigARTM v0.5.8

ModelConfig.**stream_name**
A value that defines which stream the model should use.

ModelConfig.**score_name**
A set of names that defines which scores should be calculated for the model.

ModelConfig.**reuse_theta**
A flag indicating whether the model should reuse theta values cached on the previous iterations. This option require cache_theta flag to be set to 'true' in MasterComponentConfig.

ModelConfig.**regularizer_name**
A set of names that define which regularizers should be enabled for the model. This repeated field must have the same length as *regularizer_tau*.

ModelConfig.**regularizer_tau**
A set of values that define the regularization coefficients of the corresponding regularizer. This repeated field must have the same length as *regularizer_name*.

ModelConfig.**class_id**
A set of values that define for which classes (modalities) to build topic model. This repeated field must have the same length as *class_weight*.

ModelConfig.**class_weight**
A set of values that define the weights of the corresponding classes (modalities). This repeated field must have the same length as *class_id*. This value is optional, use an empty list to set equal weights for all classes.

ModelConfig.**use_sparse_bow**
A flag indicating whether to use sparse representation of the Bag-of-words data. The default setting (*use_sparse_bow* = *true*) is best suited for processing textual collections where every token is represented in a small fraction of all documents. Dense representation (*use_sparse_bow* = *false*) better fits for non-textual collections (for example for matrix factorization).

Note that *class_weight* and *class_id* must not be used together with *use_sparse_bow*=*false*.

ModelConfig.**use_random_theta**
A flag indicating whether to initialize `p(t|d)` distribution with random uniform distribution. The default setting (*use_random_theta* = *false*) sets `p(t|d) = 1/T`, where `T` stands for *topics_count*. Note that *reuse_theta* flag takes priority over *use_random_theta* flag, so that if *reuse_theta* = *true* and there is a cache entry from previous iteration the cache entry will be used regardless of *use_random_theta* flag.

ModelConfig.**use_new_tokens**
A flag indicating whether to automatically include new tokens into the topic model. This setting is set to *True* by default. As a result, every new token observed in batches is automatically incorporated into topic model during the next model synchronization (*ArtmSynchronizeModel()*). The `n_wt_` weights for new tokens randomly generated from `[0..1]` range.

ModelConfig.**opt_for_avx**
: An experimental flag that allows to disable AVX optimization in processor. By default this option is enabled as on average it adds ca. 40% speedup on physical hardware. You may want to disable this option if you are running on Windows inside virtual machine, or in situation when BigARTM performance degrades from iteration to interation.

  This option does not affect the results, and is only intended for advanced users experimenting with BigARTM performance.

9.5.11 RegularizerConfig

class messages_pb2.**RegularizerConfig**

Represents a configuration of a general regularizer.

```
message RegularizerConfig {
  enum Type {
    SmoothSparseTheta = 0;
    SmoothSparsePhi = 1;
    DecorrelatorPhi = 2;
    LabelRegularizationPhi = 4;
  }

  optional string name = 1;
  optional Type type = 2;
  optional bytes config = 3;
}
```

RegularizerConfig.**name**
: A value that defines the name of the regularizer. The name must be unique across all names defined in the master component.

RegularizerConfig.**type**
: A value that defines the type of the regularizer.

SmoothSparseTheta	Smooth-sparse regularizer for theta matrix
SmoothSparsePhi	Smooth-sparse regularizer for phi matrix
DecorrelatorPhi	Decorrelator regularizer for phi matrix
LabelRegularizationPhi	Label regularizer for phi matrix

RegularizerConfig.**config**
: A serialized protobuf message that describes regularizer config for the specific regularizer type.

9.5.12 SmoothSparseThetaConfig

class messages_pb2.**SmoothSparseThetaConfig**

Represents a configuration of a SmoothSparse Theta regularizer.

```
message SmoothSparseThetaConfig {
  repeated string topic_name = 1;
  repeated float alpha_iter = 2;
}
```

SmoothSparseThetaConfig.**topic_name**
: A set of topic names that defines which topics in the model should be regularized. This value is optional, use an empty list to regularize all topics.

SmoothSparseThetaConfig.**alpha_iter**
: A field of the same length as *ModelConfig.inner_iterations_count* that defines relative regularization weight for every iteration inner iterations. The actual regularization value is calculated as product of alpha_iter[i] and *ModelConfig.regularizer_tau*.

  To specify different regularization weight for different topics create multiple regularizers with different *topic_name* set, and use different values of *ModelConfig.regularizer_tau*.

9.5.13 SmoothSparsePhiConfig

class messages_pb2.**SmoothSparsePhiConfig**

Represents a configuration of a SmoothSparse Phi regularizer.

```
message SmoothSparsePhiConfig {
  repeated string topic_name = 1;
  repeated string class_id = 2;
  optional string dictionary_name = 3;
}
```

SmoothSparsePhiConfig.**topic_name**
: A set of topic names that defines which topics in the model should be regularized. This value is optional, use an empty list to regularize all topics.

SmoothSparsePhiConfig.**class_id**
: This set defines which classes in the model should be regularized. This value is optional, use an empty list to regularize all classes.

SmoothSparsePhiConfig.**dictionary_name**
: An optional value defining the name of the dictionary to use. The entries of the dictionary are expected to have *DictionaryEntry.key_token*, *DictionaryEntry.class_id* and *DictionaryEntry.value* fields. The actual regularization value will be calculated as a product of *DictionaryEntry.value* and *ModelConfig.regularizer_tau*.

  This value is optional, if no dictionary is specified than all tokens will be regularized with the same weight.

9.5.14 DecorrelatorPhiConfig

class messages_pb2.**DecorrelatorPhiConfig**

Represents a configuration of a Decorrelator Phi regularizer.

```
message DecorrelatorPhiConfig {
  repeated string topic_name = 1;
  repeated string class_id = 2;
}
```

DecorrelatorPhiConfig.**topic_name**
: A set of topic names that defines which topics in the model should be regularized. This value is optional, use an empty list to regularize all topics.

DecorrelatorPhiConfig.**class_id**
: This set defines which classes in the model should be regularized. This value is optional, use an empty list to regularize all classes.

9.5.15 LabelRegularizationPhiConfig

class messages_pb2.**LabelRegularizationPhiConfig**

Represents a configuration of a Label Regularizer Phi regularizer.

```
message LabelRegularizationPhiConfig {
  repeated string topic_name = 1;
  repeated string class_id = 2;
  optional string dictionary_name = 3;
}
```

LabelRegularizationPhiConfig.**topic_name**
: A set of topic names that defines which topics in the model should be regularized.

LabelRegularizationPhiConfig.**class_id**
: This set defines which classes in the model should be regularized. This value is optional, use an empty list to regularize all classes.

LabelRegularizationPhiConfig.**dictionary_name**
: An optional value defining the name of the dictionary to use.

9.5.16 RegularizerInternalState

class messages_pb2.**RegularizerInternalState**

Represents an internal state of a general regularizer.

```
message RegularizerInternalState {
  enum Type {
    MultiLanguagePhi = 5;
  }

  optional string name = 1;
  optional Type type = 2;
  optional bytes data = 3;
}
```

9.5.17 DictionaryConfig

class messages_pb2.**DictionaryConfig**

Represents a static dictionary.

```
message DictionaryConfig {
  optional string name = 1;
  repeated DictionaryEntry entry = 2;
  optional int32 total_token_count = 3;
  optional int32 total_items_count = 4;
}
```

DictionaryConfig.**name**
: A value that defines the name of the dictionary. The name must be unique across all dictionaries defined in the master component.

DictionaryConfig.**entry**
: A list of all entries of the dictionary.

DictionaryConfig.**total_token_count**
A sum of *DictionaryEntry.token_count* across all entries in this dictionary. The value is optional and might be missing when all entries in the dictionary does not carry the *DictionaryEntry.token_count* attribute.

DictionaryConfig.**total_items_count**
A sum of *DictionaryEntry.items_count* across all entries in this dictionary. The value is optional and might be missing when all entries in the dictionary does not carry the *DictionaryEntry.items_count* attribute.

9.5.18 DictionaryEntry

class messages_pb2.**DictionaryEntry**

Represents one entry in a static dictionary.

```
message DictionaryEntry {
  optional string key_token = 1;
  optional string class_id = 2;
  optional float value = 3;
  repeated string value_tokens = 4;
  optional FloatArray values = 5;
  optional int32 token_count = 6;
  optional int32 items_count = 7;
}
```

DictionaryEntry.**key_token**
A token that defines the key of the entry.

DictionaryEntry.**class_id**
The class of the *DictionaryEntry.key_token*.

DictionaryEntry.**value**
An optional generic value, associated with the entry. The meaning of this value depends on the usage of the dictionary.

DictionaryEntry.**token_count**
An optional value, indicating the overall number of token occurrences in some collection.

DictionaryEntry.**items_count**
An optional value, indicating the overall number of documents containing the token.

9.5.19 ScoreConfig

class messages_pb2.**ScoreConfig**

Represents a configuration of a general score.

```
message ScoreConfig {
  enum Type {
    Perplexity = 0;
    SparsityTheta = 1;
    SparsityPhi = 2;
    ItemsProcessed = 3;
    TopTokens = 4;
    ThetaSnippet = 5;
    TopicKernel = 6;
  }
```

```
  optional string name = 1;
  optional Type type = 2;
  optional bytes config = 3;
}
```

ScoreConfig.**name**
: A value that defines the name of the score. The name must be unique across all names defined in the master component.

ScoreConfig.**type**
: A value that defines the type of the score.

Perplexity	Defines a config of the Perplexity score
SparsityTheta	Defines a config of the SparsityTheta score
SparsityPhi	Defines a config of the SparsityPhi score
ItemsProcessed	Defines a config of the ItemsProcessed score
TopTokens	Defines a config of the TopTokens score
ThetaSnippet	Defines a config of the ThetaSnippet score
TopicKernel	Defines a config of the TopicKernel score

ScoreConfig.**config**
: A serialized protobuf message that describes score config for the specific score type.

9.5.20 ScoreData

class messages_pb2.**ScoreData**

Represents a general result of score calculation.

```
message ScoreData {
  enum Type {
    Perplexity = 0;
    SparsityTheta = 1;
    SparsityPhi = 2;
    ItemsProcessed = 3;
    TopTokens = 4;
    ThetaSnippet = 5;
    TopicKernel = 6;
  }

  optional string name = 1;
  optional Type type = 2;
  optional bytes data = 3;
}
```

ScoreData.**name**
: A value that describes the name of the score. This name will match the name of the corresponding score config.

ScoreData.**type**
: A value that defines the type of the score.

Perplexity	Defines a Perplexity score data
SparsityTheta	Defines a SparsityTheta score data
SparsityPhi	Defines a SparsityPhi score data
ItemsProcessed	Defines a ItemsProcessed score data
TopTokens	Defines a TopTokens score data
ThetaSnippet	Defines a ThetaSnippet score data
TopicKernel	Defines a TopicKernel score data

ScoreData.**data**
A serialized protobuf message that provides the specific score result.

9.5.21 PerplexityScoreConfig

class messages_pb2.**PerplexityScoreConfig**

Represents a configuration of a perplexity score.

```
message PerplexityScoreConfig {
  enum Type {
    UnigramDocumentModel = 0;
    UnigramCollectionModel = 1;
  }

  optional string field_name = 1 [default = "@body"];  // obsolete in BigARTM v0.5.8
  optional string stream_name = 2 [default = "@global"];
  optional Type model_type = 3 [default = UnigramDocumentModel];
  optional string dictionary_name = 4;
  optional float theta_sparsity_eps = 5 [default = 1e-37];
  repeated string theta_sparsity_topic_name = 6;
}
```

PerplexityScoreConfig.**field_name**
Obsolete in BigARTM v0.5.8

PerplexityScoreConfig.**stream_name**
A value that defines which stream should be used in perplexity calculation.

9.5.22 PerplexityScore

class messages_pb2.**PerplexityScore**

Represents a result of calculation of a perplexity score.

```
message PerplexityScore {
  optional double value = 1;
  optional double raw = 2;
  optional double normalizer = 3;
  optional int32 zero_words = 4;
  optional double theta_sparsity_value = 5;
  optional int32 theta_sparsity_zero_topics = 6;
  optional int32 theta_sparsity_total_topics = 7;
}
```

PerplexityScore.**value**
A perplexity value which is calculated as exp(-raw/normalizer).

PerplexityScore.**raw**
A numerator of perplexity calculation. This value is equal to the likelihood of the topic model.

PerplexityScore.**normalizer**
: A denominator of perplexity calculation. This value is equal to the total number of tokens in all processed items.

PerplexityScore.**zero_words**
: A number of tokens that have zero probability p(w|t,d) in a document. Such tokens are evaluated based on to unigram document model or unigram colection model.

PerplexityScore.**theta_sparsity_value**
: A fraction of zero entries in the theta matrix.

9.5.23 SparsityThetaScoreConfig

class messages_pb2.**SparsityThetaScoreConfig**

Represents a configuration of a theta sparsity score.

```
message SparsityThetaScoreConfig {
  optional string field_name = 1 [default = "@body"];  // obsolete in BigARTM v0.5.8
  optional string stream_name = 2 [default = "@global"];
  optional float eps = 3 [default = 1e-37];
  repeated string topic_name = 4;
}
```

SparsityThetaScoreConfig.**field_name**
: Obsolete in BigARTM v0.5.8

SparsityThetaScoreConfig.**stream_name**
: A value that defines which stream should be used in theta sparsity calculation.

SparsityThetaScoreConfig.**eps**
: A small value that defines zero threshold for theta probabilities. Theta values below the threshold will be counted as zeros when calculating theta sparsity score.

SparsityThetaScoreConfig.**topic_name**
: A set of topic names that defines which topics should be used for score calculation. The names correspond to *ModelConfig.topic_name*. This value is optional, use an empty list to calculate the score for all topics.

9.5.24 SparsityThetaScore

class messages_pb2.**SparsityThetaScoreConfig**

Represents a result of calculation of a theta sparsity score.

```
message SparsityThetaScore {
  optional double value = 1;
  optional int32 zero_topics = 2;
  optional int32 total_topics = 3;
}
```

SparsityThetaScore.**value**
: A value of theta sparsity that is calculated as zero_topics / total_topics.

SparsityThetaScore.**zero_topics**
: A numerator of theta sparsity score. A number of topics that have zero probability in a topic-item distribution.

SparsityThetaScore.**total_topics**
: A denominator of theta sparsity score. A total number of topics in a topic-item distributions that are used in theta sparsity calculation.

9.5.25 SparsityPhiScoreConfig

class messages_pb2.**SparsityPhiScoreConfig**

Represents a configuration of a sparsity phi score.

```
message SparsityPhiScoreConfig {
  optional float eps = 1 [default = 1e-37];
  optional string class_id = 2;
  repeated string topic_name = 3;
}
```

SparsityPhiScoreConfig.**eps**
: A small value that defines zero threshold for phi probabilities. Phi values below the threshold will be counted as zeros when calculating phi sparsity score.

SparsityPhiScoreConfig.**class_id**
: A value that defines the class of tokens to use for score calculation. This value corresponds to *ModelConfig.class_id* field. This value is optional. By default the score will be calculated for the default class ('@default_class').

SparsityPhiScoreConfig.**topic_name**
: A set of topic names that defines which topics should be used for score calculation. This value is optional, use an empty list to calculate the score for all topics.

9.5.26 SparsityPhiScore

class messages_pb2.**SparsityPhiScore**

Represents a result of calculation of a phi sparsity score.

```
message SparsityPhiScore {
  optional double value = 1;
  optional int32 zero_tokens = 2;
  optional int32 total_tokens = 3;
}
```

SparsityPhiScore.**value**
: A value of phi sparsity that is calculated as zero_tokens / total_tokens.

SparsityPhiScore.**zero_tokens**
: A numerator of phi sparsity score. A number of tokens that have zero probability in a token-topic distribution.

SparsityPhiScore.**total_tokens**
: A denominator of phi sparsity score. A total number of tokens in a token-topic distributions that are used in phi sparsity calculation.

9.5.27 ItemsProcessedScoreConfig

class messages_pb2.**ItemsProcessedScoreConfig**

Represents a configuration of an items processed score.

```
message ItemsProcessedScoreConfig {
  optional string field_name = 1 [default = "@body"];  // obsolete in BigARTM v0.5.8
  optional string stream_name = 2 [default = "@global"];
}
```

ItemsProcessedScoreConfig.**field_name**
Obsolete in BigARTM v0.5.8

ItemsProcessedScoreConfig.**stream_name**
A value that defines which stream should be used in calculation of processed items.

9.5.28 ItemsProcessedScore

class messages_pb2.**ItemsProcessedScore**

Represents a result of calculation of an items processed score.

```
message ItemsProcessedScore {
  optional int32 value = 1;
}
```

ItemsProcessedScore.**value**
A number of items that belong to the stream *ItemsProcessedScoreConfig.stream_name* and have been processed during iterations. Currently this number is aggregated throughout all iterations.

9.5.29 TopTokensScoreConfig

class messages_pb2.**TopTokensScoreConfig**

Represents a configuration of a top tokens score.

```
message TopTokensScoreConfig {
  optional int32 num_tokens = 1 [default = 10];
  optional string class_id = 2;
  repeated string topic_name = 3;
}
```

TopTokensScoreConfig.**num_tokens**
A value that defines how many top tokens should be retrieved for each topic.

TopTokensScoreConfig.**class_id**
A value that defines for which class of the model to collect top tokens. This value corresponds to *ModelConfig.class_id* field.

This parameter is optional. By default tokens will be retrieved for the default class ('@default_class').

TopTokensScoreConfig.**topic_name**
A set of values that represent the names of the topics to include in the result. The names correspond to *ModelConfig.topic_name*.

This parameter is optional. By default top tokens will be calculated for all topics in the model.

9.5.30 TopTokensScore

class messages_pb2.**TopTokensScore**

Represents a result of calculation of a top tokens score.

```
message TopTokensScore {
  optional int32 num_entries = 1;
  repeated string topic_name = 2;
  repeated int32 topic_index = 3;
  repeated string token = 4;
```

```
  repeated float weight = 5;
}
```

The data in this score is represented in a table-like format. sorted on topic_index. The following code block gives a typical usage example. The loop below is guarantied to process all top-N tokens for the first topic, then for the second topic, etc.

```
for (int i = 0; i < top_tokens_score.num_entries(); i++) {
  // Gives a index from 0 to (model_config.topics_size() - 1)
  int topic_index = top_tokens_score.topic_index(i);

  // Gives one of the topN tokens for topic 'topic_index'
  std::string token = top_tokens_score.token(i);

  // Gives the weight of the token
  float weight = top_tokens_score.weight(i);
}
```

TopTokensScore.**num_entries**
: A value indicating the overall number of entries in the score. All the remaining repeated fiels in this score will have this length.

TopTokensScore.**token**
: A repeated field of *num_entries* elements, containing tokens with high probability.

TopTokensScore.**weight**
: A repeated field of *num_entries* elements, containing the *p(t|w)* probabilities.

TopTokensScore.**topic_index**
: A repeated field of *num_entries* elements, containing integers between 0 and (*ModelConfig.topics_count* - 1).

TopTokensScore.**topic_name**
: A repeated field of *num_entries* elements, corresponding to the values of *ModelConfig.topic_name* field.

9.5.31 ThetaSnippetScoreConfig

class messages_pb2.**ThetaSnippetScoreConfig**

Represents a configuration of a theta snippet score.

```
message ThetaSnippetScoreConfig {
  optional string field_name = 1 [default = "@body"];  // obsolete in BigARTM v0.5.8
  optional string stream_name = 2 [default = "@global"];
  repeated int32 item_id = 3 [packed = true];  // obsolete in BigARTM v0.5.8
  optional int32 item_count = 4 [default = 10];
}
```

ThetaSnippetScoreConfig.**field_name**
: Obsolete in BigARTM v0.5.8

ThetaSnippetScoreConfig.**stream_name**
: A value that defines which stream should be used in calculation of a theta snippet.

ThetaSnippetScoreConfig.**item_id**
: Obsolete in BigARTM v0.5.8.

ThetaSnippetScoreConfig.**item_count**
The number of items to retrieve. ThetaSnippetScore will select last *item_count* processed items and return their theta vectors.

9.5.32 ThetaSnippetScore

class messages_pb2.**ThetaSnippetScore**

Represents a result of calculation of a theta snippet score.

```
message ThetaSnippetScore {
  repeated int32 item_id = 1;
  repeated FloatArray values = 2;
}
```

ThetaSnippetScore.**item_id**
A set of item ids for which theta snippet have been calculated. Items are identified by the item id.

ThetaSnippetScore.**values**
A set of values that define topic probabilities for each item. The length of these repeated values will match the number of item ids specified in *ThetaSnippetScore.item_id*. Each repeated field contains float array of topic probabilities in the natural order of topic ids.

9.5.33 TopicKernelScoreConfig

class messages_pb2.**TopicKernelScoreConfig**

Represents a configuration of a topic kernel score.

```
message TopicKernelScoreConfig {
  optional float eps = 1 [default = 1e-37];
  optional string class_id = 2;
  repeated string topic_name = 3;
  optional double probability_mass_threshold = 4 [default = 0.1];
}
```

- *Kernel* of a topic model is defined as the list of all tokens such that the probability p(t | w) exceeds probability mass threshold.
- *Kernel size* of a topic t is defined as the number of tokens in its kernel.
- *Topic purity* of a topic t is defined as the sum of p(w | t) across all tokens w in the kernel.
- *Topic contrast* of a topic t is defined as the sum of p(t | w) across all tokens w in the kernel defided by the size of the kernel.

TopicKernelScoreConfig.**eps**
Defines the minimum threshold on kernel size. In most cases this parameter should be kept at the default value.

TopicKernelScoreConfig.**class_id**
A value that defines the class of tokens to use for score calculation. This value corresponds to *ModelConfig.class_id* field. This value is optional. By default the score will be calculated for the default class ('@default_class').

TopicKernelScoreConfig.**topic_name**
A set of topic names that defines which topics should be used for score calculation. This value is optional, use an empty list to calculate the score for all topics.

TopicKernelScoreConfig.**probability_mass_threshold**
Defines the probability mass threshold (see the definition of *kernel* above).

9.5.34 TopicKernelScore

class messages_pb2.**TopicKernelScore**

Represents a result of calculation of a topic kernel score.

```
message TopicKernelScore {
  optional DoubleArray kernel_size = 1;
  optional DoubleArray kernel_purity = 2;
  optional DoubleArray kernel_contrast = 3;
  optional double average_kernel_size = 4;
  optional double average_kernel_purity = 5;
  optional double average_kernel_contrast = 6;
}
```

TopicKernelScore.**kernel_size**
Provides the kernel size for all requested topics. The length of this *DoubleArray* is always equal to the overall number of topics. The values of -1 correspond to non-calculated topics. The remaining values carry the kernel size of the requested topics.

TopicKernelScore.**kernel_purity**
Provides the kernel purity for all requested topics. The length of this *DoubleArray* is always equal to the overall number of topics. The values of -1 correspond to non-calculated topics. The remaining values carry the kernel size of the requested topics.

TopicKernelScore.**kernel_contrast**
Provides the kernel contrast for all requested topics. The length of this *DoubleArray* is always equal to the overall number of topics. The values of -1 correspond to non-calculated topics. The remaining values carry the kernel contrast of the requested topics.

TopicKernelScore.**average_kernel_size**
Provides the average kernel size across all the requested topics.

TopicKernelScore.**average_kernel_purity**
Provides the average kernel purity across all the requested topics.

TopicKernelScore.**average_kernel_contrast**
Provides the average kernel contrast across all the requested topics.

9.5.35 TopicModel

class messages_pb2.**TopicModel**

Represents a topic model. This message can contain data in either dense or sparse format. The key idea behind sparse format is to avoid storing zero p(w|t) elements of the Phi matrix. Please refer to the description of *TopicModel.topic_index* field for more details.

To distinguish between these two formats check whether repeated field *TopicModel.topic_index* is empty. An empty field indicate a dense format, otherwise the message contains data in a sparse format. To request topic model in a sparse format set *GetTopicModelArgs.use_sparse_format* field to True when calling *ArtmRequestTopicModel()*.

```
message TopicModel {
  enum OperationType {
    Initialize = 0;
```

```
    Increment = 1;
    Overwrite = 2;
    Remove = 3;
    Ignore = 4;
  }

  optional string name = 1 [default = "@model"];
  optional int32 topics_count = 2;
  repeated string topic_name = 3;
  repeated string token = 4;
  repeated FloatArray token_weights = 5;
  repeated string class_id = 6;

  message TopicModelInternals {
    repeated FloatArray n_wt = 1;
    repeated FloatArray r_wt = 2;
  }

  optional bytes internals = 7;  // obsolete in BigARTM v0.6.3
  repeated IntArray topic_index = 8;
  repeated OperationType operation_type = 9;
}
```

`TopicModel.`**`name`**
: A value that describes the name of the topic model (*`TopicModel.name`*).

`TopicModel.`**`topics_count`**
: A value that describes the number of topics in this message.

`TopicModel.`**`topic_name`**
: A value that describes the names of the topics included in given *TopicModel* message. This values will represent a subset of topics, defined by *`GetTopicModelArgs.topic_name`* message. In case of empty *`GetTopicModelArgs.topic_name`* this values will correspond to the entire set of topics, defined in *`ModelConfig.topic_name`* field.

`TopicModel.`**`token`**
: The set of all tokens, included in the topic model.

`TopicModel.`**`token_weights`**
: A set of token weights. The length of this repeated field will match the length of the repeated field *`TopicModel.token`*. The length of each *FloatArray* will match the *`TopicModel.topics_count`* field (in dense representation), or the length of the corresponding *IntArray* from *`TopicModel.topic_index`* field (in sparse representation).

`TopicModel.`**`class_id`**
: A set values that specify the class (modality) of the tokens. The length of this repeated field will match the length of the repeated field *`TopicModel.token`*.

`TopicModel.`**`internals`**
: Obsolete in BigARTM v0.6.3.

`TopicModel.`**`topic_index`**
: A repeated field used for sparse topic model representation. This field has the same length as *`TopicModel.token`*, *`TopicModel.class_id`* and *`TopicModel.token_weights`*. Each element in *topic_index* is an instance of *IntArray* message, containing a list of values between 0 and the length of *`TopicModel.topic_name`* field. This values correspond to the indices in *`TopicModel.topic_name`* array, and tell which topics has non-zero `p(w|t)` probabilities for a given token. The actual `p(w|t)` values can be found in *`TopicModel.token_weights`* field. The length of each *IntArray* message

in *TopicModel.topic_index* field equals to the length of the corresponding *FloatArray* message in *TopicModel.token_weights* field.

> **Warning:** Be careful with *TopicModel.topic_index* when this message represents a subset of topics, defined by *GetTopicModelArgs.topic_name*. In this case indices correspond to the selected subset of topics, which might not correspond to topic indices in the original *ModelConfig* message.

TopicModel.**operation_type**
A set of values that define operation to perform on each token when topic model is used as an argument of *ArtmOverwriteTopicModel()*.

Initialize	Indicates that a new token should be added to the topic model. Initial n_wt counter will be initialized with random value from [0, 1] range. *TopicModel.token_weights* is ignored. This operation is ignored if token already exists.
Increment	Indicates that n_wt counter of the token should be increased by values, specified in *TopicModel.token_weights* field. A new token will be created if it does not exist yet.
Overwrite	Indicates that n_wt counter of the token should be set to the value, specified in *TopicModel.token_weights* field. A new token will be created if it does not exist yet.
Remove	Indicates that the token should be removed from the topic model. *TopicModel.token_weights* is ignored.
Ignore	Indicates no operation for the token. The effect is the same as if the token is not present in this message.

9.5.36 ThetaMatrix

class messages_pb2.**ThetaMatrix**

Represents a theta matrix. This message can contain data in either dense or sparse format. The key idea behind sparse format is to avoid storing zero p(t|d) elements of the Theta matrix. Sparse representation of Theta matrix is equivalent to sparse representation of Phi matrix. Please, refer to *TopicModel* for detailed description of the sparse format.

```
message ThetaMatrix {
  optional string model_name = 1 [default = "@model"];
  repeated int32 item_id = 2;
  repeated FloatArray item_weights = 3;
  repeated string topic_name = 4;
  optional int32 topics_count = 5;
  repeated string item_title = 6;
  repeated IntArray topic_index = 7;
}
```

ThetaMatrix.**model_name**
A value that describes the name of the topic model. This name will match the name of the corresponding model config.

ThetaMatrix.**item_id**
A set of item IDs corresponding to *Item.id* values.

ThetaMatrix.**item_weights**
A set of item ID weights. The length of this repeated field will match the length of the repeated field *ThetaMatrix.item_id*. The length of each *FloatArray* will match the *ThetaMatrix.topics_count* field (in dense representation), or the length of the corresponding *IntArray* from *ThetaMatrix.topic_index* field (in sparse representation).

ThetaMatrix.**topic_name**
: A value that describes the names of the topics included in given *ThetaMatrix* message. This values will represent a subset of topics, defined by *GetThetaMatrixArgs.topic_name* message. In case of empty *GetTopicModelArgs.topic_name* this values will correspond to the entire set of topics, defined in *ModelConfig.topic_name* field.

ThetaMatrix.**topics_count**
: A value that describes the number of topics in this message.

ThetaMatrix.**item_title**
: A set of item titles, corresponding to *Item.title* values. Beware that this field might be empty (e.g. of zero length) if all items did not have title specified in *Item.title*.

ThetaMatrix.**topic_index**
: A repeated field used for sparse theta matrix representation. This field has the same length as *ThetaMatrix.item_id*, *ThetaMatrix.item_weights* and *ThetaMatrix.item_title*. Each element in *topic_index* is an instance of *IntArray* message, containing a list of values between 0 and the length of *TopicModel.topic_name* field. This values correspond to the indices in *ThetaMatrix.topic_name* array, and tell which topics has non-zero `p(t|d)` probabilities for a given item. The actual `p(t|d)` values can be found in *ThetaMatrix.item_weights* field. The length of each *IntArray* message in *ThetaMatrix.topic_index* field equals to the length of the corresponding *FloatArray* message in *ThetaMatrix.item_weights* field.

  Warning: Be careful with *ThetaMatrix.topic_index* when this message represents a subset of topics, defined by *GetThetaMatrixArgs.topic_name*. In this case indices correspond to the selected subset of topics, which might not correspond to topic indices in the original *ModelConfig* message.

9.5.37 CollectionParserConfig

class messages_pb2.**CollectionParserConfig**

Represents a configuration of a collection parser.

```
message CollectionParserConfig {
  enum Format {
    BagOfWordsUci = 0;
    MatrixMarket = 1;
  }

  optional Format format = 1 [default = BagOfWordsUci];
  optional string docword_file_path = 2;
  optional string vocab_file_path = 3;
  optional string target_folder = 4;
  optional string dictionary_file_name = 5;
  optional int32 num_items_per_batch = 6 [default = 1000];
  optional string cooccurrence_file_name = 7;
  repeated string cooccurrence_token = 8;
  optional bool use_unity_based_indices = 9 [default = true];
}
```

CollectionParserConfig.**format**
: A value that defines the format of a collection to be parsed.

BagOfWordsUci	A bag-of-words collection, stored in UCI format. UCI format must have two files - *vocab.*.txt* and *docword.*.txt*, defined by *docword_file_path* and *vocab_file_path*. The format of the docword.*.txt file is 3 header lines, followed by NNZ triples: D W NNZ docID wordID count docID wordID count ... docID wordID count The file must be sorted on docID. Values of wordID must be unity-based (not zero-based). The format of the vocab.*.txt file is line containing wordID=n. Note that words must not have spaces or tabs. In vocab.*.txt file it is also possible to specify *Batch.class_id* for tokens, as it is shown in this example: token1 @default_class token2 custom_class token3 @default_class token4 Use space or tab to separate token from its class. Token that are not followed by class label automatically get ''@default_class'' as a lable (see ''token4" in the example).
MatrixMarket	See the description at http://math.nist.gov/MatrixMarket/formats.html In this mode parameter *docword_file_path* must refer to a file in Matrix Market format. Parameter *vocab_file_path* is also required and must refer to a dictionary file exported in gensim format (dictionary.save_as_text()).

CollectionParserConfig.**docword_file_path**

A value that defines the disk location of a `docword.*.txt` file (the bag of words file in sparse format).

`CollectionParserConfig.`**`vocab_file_path`**
A value that defines the disk location of a `vocab.*.txt` file (the file with the vocabulary of the collection).

`CollectionParserConfig.`**`target_folder`**
A value that defines the disk location where to stores all the results after parsing the colleciton. Usually the resulting location will contain a set of *batches*, and a *DictionaryConfig* that contains all unique tokens occured in the collection. Such location can be further passed MasterComponent via *MasterComponentConfig.disk_path*.

`CollectionParserConfig.`**`dictionary_file_name`**
A file name where to save the *DictionaryConfig* message that contains all unique tokens occured in the collection. The file will be created in *target_folder*.

This parameter is optional. The dictionary will be still collected even when this parameter is not provided, but the resulting dictionary will be only returned as the result of *ArtmRequestParseCollection*, but it will not be stored to disk.

In the resulting dictionary each entry will have the following fields:

- *DictionaryEntry.key_token* - the textual representation of the token,
- *DictionaryEntry.class_id* - the label of the default class ("@DefaultClass"),
- *DictionaryEntry.token_count* - the overall number of occurrences of the token in the collection,
- *DictionaryEntry.items_count* - the number of documents in the collection, containing the token.
- *DictionaryEntry.value* - the ratio between *token_count* and *total_token_count*.

Use *ArtmRequestLoadDictionary* method to load the resulting dictionary.

`CollectionParserConfig.`**`num_items_per_batch`**
A value indicating the desired number of items per batch.

`CollectionParserConfig.`**`cooccurrence_file_name`**
A file name where to save the *DictionaryConfig* message that contains information about co-occurrence of all pairs of tokens in the collection. The file will be created in *target_folder*.

This parameter is optional. No cooccurrence information will be collected if the filename is not provided.

In the resulting dictionary each entry will correspond to two tokens ('<first>' and '<second>'), and carry the information about co-occurrence of this tokens in the collection.

- *DictionaryEntry.key_token* - a string of the form '<first>~<second>', produced by concatenation of two tokens together via the tilde symbol ('~'). <*first*> tokens is guarantied lexicographic less than the <*second*> token.
- *DictionaryEntry.class_id* - the label of the default class ("@DefaultClass").
- *DictionaryEntry.items_count* - the number of documents in the collection, containing both tokens ('<first>' and '<second>')

Use *ArtmRequestLoadDictionary* method to load the resulting dictionary.

`CollectionParserConfig.`**`cooccurrence_token`**
A list of tokens to collect cooccurrence information. A cooccurrence of the pair <*first*>~<*second*> will be collected only when both tokens are present in *CollectionParserConfig.cooccurrence_token*.

`CollectionParserConfig.`**`use_unity_based_indices`**
A flag indicating whether to interpret indices in docword file as unity-based or as zero-based. By default *'use_unity_based_indices = True'*, as required by UCI bag-of-words format.

9.5.38 SynchronizeModelArgs

class messages_pb2.**SynchronizeModelArgs**

Represents an argument of synchronize model operation.

```
message SynchronizeModelArgs {
  optional string model_name = 1;
  optional float decay_weight = 2 [default = 0.0];
  optional bool invoke_regularizers = 3 [default = true];
  optional float apply_weight = 4 [default = 1.0];
}
```

SynchronizeModelArgs.**model_name**
The name of the model to be synchronized. This value is optional. When not set, all models will be synchronized with the same decay weight.

SynchronizeModelArgs.**decay_weight**
The decay weight and *apply_weight* define how to combine existing topic model with all increments, calculated since the last *ArtmSynchronizeModel()*. This is best described by the following formula:

`n_wt_new = n_wt_old * decay_weight + n_wt_inc * apply_weight`,

where `n_wt_old` describe current topic model, `n_wt_inc` describe increment calculated since last *ArtmSynchronizeModel()*, `n_wt_new` define the resulting topic model.

Expected values of both parameters are between 0.0 and 1.0. Here are some examples:

- Combination of *decay_weight=0.0* and *apply_weight=1.0* states that the previous Phi matrix of the topic model will be disregarded completely, and the new Phi matrix will be formed based on new increments gathered since last model synchronize.
- Combination of *decay_weight=1.0* and *apply_weight=1.0* states that new increments will be appended to the current Phi matrix without any decay.
- Combination of *decay_weight=1.0* and *apply_weight=0.0* states that new increments will be disregarded, and current Phi matrix will stay unchanged.
- To reproduce Online variational Bayes for LDA algorighm by Matthew D. Hoffman set *decay_weight = 1 - rho* and *apply_weight = rho*, where parameter rho is defined as *rho = exp(tau + t, -kappa)*. See Online Learning for Latent Dirichlet Allocation for further details.

SynchronizeModelArgs.**apply_weight**
See *decay_weight* for the description.

SynchronizeModelArgs.**invoke_regularizers**
A flag indicating whether to invoke all phi-regularizers.

9.5.39 InitializeModelArgs

class messages_pb2.**InitializeModelArgs**

Represents an argument of *ArtmInitializeModel()* operation. Please refer to example14_initialize_topic_model.py for further information.

```
message InitializeModelArgs {
  enum SourceType {
    Dictionary = 0;
    Batches = 1;
  }
```

```
  message Filter {
    optional string class_id = 1;
    optional float min_percentage = 2;
    optional float max_percentage = 3;
    optional int32 min_items = 4;
    optional int32 max_items = 5;
    optional int32 min_total_count = 6;
    optional int32 min_one_item_count = 7;
  }

  optional string model_name = 1;
  optional string dictionary_name = 2;
  optional SourceType source_type = 3 [default = Dictionary];

  optional string disk_path = 4;
  repeated Filter filter = 5;
}
```

`InitializeModelArgs.`**`model_name`**
: The name of the model to be initialized.

`InitializeModelArgs.`**`dictionary_name`**
: The name of the dictionary containing all tokens that should be initialized.

9.5.40 GetTopicModelArgs

Represents an argument of *`ArtmRequestTopicModel()`* operation.

```
message GetTopicModelArgs {
  enum RequestType {
    Pwt = 0;
    Nwt = 1;
  }

  optional string model_name = 1;
  repeated string topic_name = 2;
  repeated string token = 3;
  repeated string class_id = 4;
  optional bool use_sparse_format = 5;
  optional float eps = 6 [default = 1e-37];
  optional RequestType request_type = 7 [default = Pwt];
}
```

`GetTopicModelArgs.`**`model_name`**
: The name of the model to be retrieved.

`GetTopicModelArgs.`**`topic_name`**
: The list of topic names to be retrieved. This value is optional. When not provided, all topics will be retrieved.

`GetTopicModelArgs.`**`token`**
: The list of tokens to be retrieved. The length of this field must match the length of *`class_id`* field. This field is optional. When not provided, all tokens will be retrieved.

`GetTopicModelArgs.`**`class_id`**
: The list of classes corresponding to all tokens. The length of this field must match the length of *`token`* field. This field is only required together with *`token`*, otherwise it is ignored.

`GetTopicModelArgs.`**`use_sparse_format`**
: An optional flag that defines whether to use sparse format for the resulting `TopicModel` message. See

`TopicModel` message for additional information about the sparse format. Note that setting *use_sparse_format* = *true* results in empty *`TopicModel.internals`* field.

`GetTopicModelArgs.`**`eps`**
A small value that defines zero threshold for `p(w|t)` probabilities. This field is only used in sparse format. `p(w|t)` below the threshold will be excluded from the resulting Phi matrix.

`GetTopicModelArgs.`**`request_type`**
An optional value that defines what kind of data to retrieve in this operation.

Pwt	Indicates that the resulting *TopicModel* message should contain `p(w|t)` probabilities. This values are normalized to form a probability distribution (`sum_w p(w|t) = 1` for all topics `t`).
Nwt	Indicates that the resulting *TopicModel* message should contain internal `n_wt` counters of the topic model. This values represent an internal state of the topic model.

Default setting is to retrieve `p(w|t)` probabilities. This probabilities are sufficient to infer `p(t|d)` distributions using this topic model.

`n_wt` counters allow you to restore the precise state of the topic model. By passing this values in *`ArtmOverwriteTopicModel()`* operation you are guarantied to get the model in the same state as you retrieved it. As the result you may continue topic model inference from the point you have stopped it last time.

`p(w|t)` values can be also restored via c:func:*ArtmOverwriteTopicModel* operation. The resulting model will give the same `p(t|d)` distributions, however you should consider this model as *read-only*, and do not call *`ArtmSynchronizeModel()`* on it.

9.5.41 GetThetaMatrixArgs

Represents an argument of *`ArtmRequestThetaMatrix()`* operation.

```
message GetThetaMatrixArgs {
  optional string model_name = 1;
  optional Batch batch = 2;
  repeated string topic_name = 3;
  repeated int32 topic_index = 4;
  optional bool clean_cache = 5 [default = false];
  optional bool use_sparse_format = 6 [default = false];
  optional float eps = 7 [default = 1e-37];
}
```

`GetThetaMatrixArgs.`**`model_name`**
The name of the model to retrieved theta matrix for.

`GetThetaMatrixArgs.`**`batch`**
The *Batch* to classify with the model.

`GetThetaMatrixArgs.`**`topic_name`**
The list of topic names, describing which topics to include in the Theta matrix. The values of this field should correspond to values in *`ModelConfig.topic_name`*. This field is optional, by default all topics will be included.

`GetThetaMatrixArgs.`**`topic_index`**
The list of topic indices, describing which topics to include in the Theta matrix. The values of this field should be an integers between 0 and (*`ModelConfig.topics_count`* - 1). This field is optional, by default all topics will be included.

Note that this field acts similar to *`GetThetaMatrixArgs.topic_name`*. It is not allowed to specify both *topic_index* and *topic_name* at the same time. The recommendation is to use *topic_name*.

`GetThetaMatrixArgs.`**`clean_cache`**
: An optional flag that defines whether to clear the theta matrix cache after this operation. Setting this value to *True* will clear the cache for a topic model, defined by *GetThetaMatrixArgs.model_name*. This value is only applicable when *MasterComponentConfig.cache_theta* is set to *True*.

`GetThetaMatrixArgs.`**`use_sparse_format`**
: An optional flag that defines whether to use sparse format for the resulting `ThetaMatrix` message. See `ThetaMatrix` message for additional information about the sparse format.

`GetThetaMatrixArgs.`**`eps`**
: A small value that defines zero threshold for `p(t|d)` probabilities. This field is only used in sparse format. `p(t|d)` below the threshold will be excluded from the resulting Theta matrix.

9.5.42 GetScoreValueArgs

Represents an argument of get score operation.

```
message GetScoreValueArgs {
  optional string model_name = 1;
  optional string score_name = 2;
  optional Batch batch = 3;
}
```

`GetScoreValueArgs.`**`model_name`**
: The name of the model to retrieved score for.

`GetScoreValueArgs.`**`score_name`**
: The name of the score to retrieved.

`GetScoreValueArgs.`**`batch`**
: The *Batch* to calculate the score. This option is only applicable to cumulative scores. When not provided the score will be reported for all batches processed since last *ArtmInvokeIteration()*.

9.5.43 AddBatchArgs

Represents an argument of *ArtmAddBatch()* operation.

```
message AddBatchArgs {
  optional Batch batch = 1;
  optional int32 timeout_milliseconds = 2 [default = -1];
  optional bool reset_scores = 3 [default = false];
  optional string batch_file_name = 4;
}
```

`AddBatchArgs.`**`batch`**
: The *Batch* to add.

`AddBatchArgs.`**`timeout_milliseconds`**
: Timeout in milliseconds for this operation.

`AddBatchArgs.`**`reset_scores`**
: An optional flag that defines whether to reset all scores before this operation.

`AddBatchArgs.`**`batch_file_name`**
: An optional value that defines disk location of the batch to add. You must choose between parameters *batch_file_name* or *batch* (either of them has to be specified, but not both at the same time).

9.5.44 InvokeIterationArgs

Represents an argument of *ArtmInvokeIteration()* operation.

```
message InvokeIterationArgs {
  optional int32 iterations_count = 1 [default = 1];
  optional bool reset_scores = 2 [default = true];
  optional string disk_path = 3;
}
```

`InvokeIterationArgs.`**`iterations_count`**
: An integer value describing how many iterations to invoke.

`InvokeIterationArgs.`**`reset_scores`**
: An optional flag that defines whether to reset all scores before this operation.

`InvokeIterationArgs.`**`disk_path`**
: A value that defines the disk location with batches to process on this iteration.

9.5.45 WaitIdleArgs

Represents an argument of *ArtmWaitIdle()* operation.

```
message WaitIdleArgs {
  optional int32 timeout_milliseconds = 1 [default = -1];
}
```

`WaitIdleArgs.`**`timeout_milliseconds`**
: Timeout in milliseconds for this operation.

9.5.46 ExportModelArgs

Represents an argument of *ArtmExportModel()* operation.

```
message ExportModelArgs {
  optional string file_name = 1;
  optional string model_name = 2;
}
```

`ExportModelArgs.`**`file_name`**
: A target file name where to store topic model.

`ExportModelArgs.`**`model_name`**
: A value that describes the name of the topic model. This name will match the name of the corresponding model config.

9.5.47 ImportModelArgs

Represents an argument of *ArtmImportModel()* operation.

```
message ImportModelArgs {
  optional string file_name = 1;
  optional string model_name = 2;
}
```

`ImportModelArgs.`**`file_name`**
: A target file name from where to load topic model.

`ImportModelArgs.`**`model_name`**
A value that describes the name of the topic model. This name will match the name of the corresponding model config.

9.6 Python Interface

This document explains all classes in python interface of BigARTM library.

9.6.1 Library

class `artm.library.`**`Library`**(*artm_shared_library* = *""*)
Creates an ArtmLibrary object, wrapping the BigARTM shared library.

The *artm_shared_library* is an optional argument, which provides full file name of artm shared library (a disk path plus `artm.dll` on Windows or `artm.so` on Linux). When *artm_shared_library* is not specified the shared library will be searched in folders listed in `PATH` system variable. You may also configure `ARTM_SHARED_LIBRARY` system variable to provide full file name of artm shared library.

`CreateMasterComponent`(*config* = *None*)
Creates and returns an instance of *MasterComponent* class. *config* defines an optional *MasterComponentConfig* parameter that may carry the configuration of the master component.

`SaveBatch`(*batch*, *disk_path*)
Saves a given *Batch* into a disk location specified by *disk_path*.

`ParseCollection`(*collection_parser_config*)
Parses a text collection as defined by *collection_parser_config* (*CollectionParserConfig*). Returns an instance of *DictionaryConfig* which carry all unique words in the collection and their frequencies.

For more information refer to *ArtmRequestParseCollection()* and *ArtmRequestLoadDictionary()*.

`LoadDictionary`(*full_filename*)
Loads a *DictionaryConfig* from the file, defined by *full_filename* argument.

For more information refer to *ArtmRequestLoadDictionary()*.

`LoadBatch`(*full_filename*)
Loads a *Batch* from the file, defined by *full_filename* argument.

For more information refer to *ArtmRequestLoadBatch()*.

`ParseCollectionOrLoadDictionary`(*docword_file_path*, *vocab_file_path*, *target_folder*)
A simple helper method that runs *ParseCollection()* when *target_folder* is empty, otherwise tried to use *LoadDictionary()* to load the dictionary from *target_folder*.

The *docword_file_path* and *vocab_file_path* arguments should provide the disk location of docword and vocab files of the collection to be parsed.

9.6.2 MasterComponent

class `artm.library.`**`MasterComponent`**(*config* = *None*, *lib* = *None*, *disk_path* = *None*)
Creates a master component.

config is an optional instance of *MasterComponentConfig*, providing an initial configuration of the master component.

lib is an optional argument pointing to `Library`. When not specified, a default library will be used. Check the constructor of `Library` for more details.

disk_path is an optional value providing the disk folder with batches to process by this master component. Changing *disk_path* is not supported (you must recreate a new instance MasterComponent to do so). Use `InvokeIteration()` will process all batches, located under *disk_path*. Alternatively use `AddBatch()` to add a specific batch into processor queue.

Dispose()
Disposes the master component and releases all unmanaged resources.

config()
Returns current *MasterComponentConfig* of the master component.

CreateModel(config=None, topics_count=None, inner_iterations_count=None, class_ids=None, topic_names=None, use_sparse_format=None, request_type=None)
Creates and returns an instance of `Model` class based on a given *ModelConfig*. Note that the model has to be further tuned by several iterative scans over the text collection. Use `InvokeIteration()` to perform such scans.

All parameters will override values, specifed in *config*.

RemoveModel(*model*)
Removes an instance of `Model` from the master component. After this operation the *model* object became invalid and must not be used.

CreateRegularizer(*name*, *type*, *config*)
Creates and returns an instance of `Regularizer` component. *name* can be any unique identifier, that you can further use to identify regularizer (for example, in `ModelConfig.regularizer_name`). *type* can be any regularizer type (for example, the `RegularizerConfig_Type_DirichletTheta`). *config* can be any regularizer config (for example, a *SmoothSparseThetaConfig*).

CreateSmoothSparseThetaRegularizer(*name = None*, *config = None*)
Creates an instance of SmoothSparseThetaRegularizer. *config* is an optional argument of *SmoothSparseThetaConfig* type.

CreateSmoothSparsePhiRegularizer(*name = None*, *config = None*, *topic_names=None*, *class_ids=None*)
Creates an instance of SmoothSparsePhiRegularizer. *config* is an optional argument of *SmoothSparsePhiConfig* type.

CreateDecorrelatorPhiRegularizer(*name = None*, *config = None*, *topic_names=None*, *class_ids=None*)
Creates an instance of DecorrelatorPhiRegularizer. *config* is an optional argument of *DecorrelatorPhiConfig* type.

RemoveRegularizer(*regularizer*)
Removes an instance of `Regularizer` from the master component. After this operation the *regularizer* object became invalid and must not be used.

CreateScore(*name*, *type*, *config*)
Creates a score calculator inside the master component. *name* can be any unique identifier, that you can further use to identify the score (for example, in `ModelConfig.score_name`). *type* can be any score type (for example, the `ScoreConfig_Type_Perplexity`). *config* can be any score config (for example, a *PerplexityScoreConfig*).

CreatePerplexityScore(*self*, *name = None*, *config = None*, *stream_name = None*, *class_ids=None*)
Creates an instance of PerplexityScore. *config* is an optional argument of *PerplexityScoreConfig* type.

CreateSparsityThetaScore(*self*, *name = None*, *config = None*, *topic_names=None*)
Creates an instance of SparsityThetaScore. *config* is an optional argument of *SparsityThetaScoreConfig* type.

CreateSparsityPhiScore(*self*, *name = None*, *config = None*, *topic_names=None*, *class_id=None*)
Creates an instance of SparsityPhiScore. *config* is an optional argument of *SparsityPhiScoreConfig* type.

CreateItemsProcessedScore(*self*, *name = None*, *config = None*)
Creates an instance of ItemsProcessedScore. *config* is an optional argument of *ItemsProcessedScoreConfig* type.

CreateTopTokensScore(*self*, *name = None*, *config = None*, *num_tokens = None*, *class_id = None*, *topic_names=None*)
Creates an instance of TopTokensScore. *config* is an optional argument of *TopTokensScoreConfig* type.

CreateThetaSnippetScore(*self*, *name = None*, *config = None*)
Creates an instance of ThetaSnippetScore. *config* is an optional argument of *ThetaSnippetScoreConfig* type.

CreateTopicKernelScore(*self*, *name = None*, *config = None*, *topic_names=None*, *class_id=None*)
Creates an instance of TopicKernelScore. *config* is an optional argument of *TopicKernelScoreConfig* type.

RemoveScore(*name*)
Removes a score calculator with the specific name from the master component.

CreateDictionary(*config*)
Creates and returns an instance of `Dictionary` class component with a specific *DictionaryConfig*.

RemoveDictionary(*dictionary*)
Removes an instance of `Dictionary` from the master component. After this operation the *dictionary* object became invalid and must not be used.

Reconfigure(*config = None*)
Updates the configuration of the master component with new *MasterComponentConfig* value, provided by *config* parameter. Remember that some changes of the configuration are not allowed (for example, the `MasterComponentConfig.disk_path` must not change). Such configuration parameters must be provided in the constructor of `MasterComponent`.

AddBatch(*self*, *batch = None*, *batch_filename = None*, *timeout = None*, *reset_scores = False*, *args=None*)
Adds an instance of *Batch* class to the processor queue. Master component creates a copy of the *batch*, so any further changes of the *batch* object will not be picked up. *batch_filename* is an alternative to file with binary-serialized batch (you must use either *batch* or *batch_filename* option, but not both at the same time).

This operation awaits until there is enough space in processor queue. It returns *True* if await succeeded within the timeout, otherwise returns *False*. The provided timeout is in milliseconds. By default it allows an infinite time for `AddBatch()` operation.

args is an optional argument of *AddBatchArgs* type.

InvokeIteration(*iterations_count = 1*, *disk_path = None*, *args=None*)
Invokes several iterations over the collection. The recommended value for *iterations_count* is 1. *disk_path* defines the disk location with batches to process on this iteration. For more iterations use for loop around `InvokeIteration()` method. This operation is asynchronous. Use `WaitIdle()` to await until all iterations succeeded.

args is an optional argument of *InvokeIterationArgs* type.

WaitIdle(*timeout = None*, *args=None*)
Awaits for ongoing iterations. Returns *True* if iterations had been finished within the timeout, otherwise returns *False*. The provided timeout is in milliseconds. Use *timeout = -1* to allow infinite time for `WaitIdle()` operation. Remember to call `Model.Synchronize()` operation to synchronize each model that you are currently processing.

args is an optional argument of *WaitIdleArgs* type.

CreateStream(*stream*)
Creates a data stream base on the *stream* (*Stream*).

RemoveStream(*stream_name*)
Removes a stream with the specific name from the master component.

GetTopicModel(*model = None*, *args = None*)
Retrieves and returns an instance of *TopicModel* class, carrying all the data of the topic model (including the Phi matrix). Parameter *model* should be an instance of `Model` class. For more settings use *args* parameter (see *GetTopicModelArgs* for all available options).

GetRegularizerState(*regularizer_name*)
Retrieves and returns the internal state of a regularizer with the specific name.

GetThetaMatrix(*model = None*, *batch = None*, *clean_cache = None*, *args = None*)
Retrieves an instance of *ThetaMatrix* class. The content depends on *batch* parameter. When *batch* is provided, the resulting *ThetaMatrix* will contain theta values estimated for all documents in the batch. When *batch* is not provided, the resulting *ThetaMatrix* will contain theta values gathered during the last iteration.

Parameter *model* should be an instance of `Model` class. For more settings use *args* parameter (see *GetThetaMatrixArgs* for all available options).

When used without *batch*, this operation require `MasterComponentConfig.cache_theta` to be set to *True* before starting the last iteration. In this case the entire ThetaMatrix must fit into CPU memory, and for this reason `MasterComponentConfig.cache_theta` is turned off by default.

9.6.3 Model

class `artm.library.`**Model**
This constructor must not be used explicitly. The only correct way of creating a Model is through `MasterComponent.CreateModel()` method.

name()
Returns the string name of the model.

Reconfigure(*config = None*)
Updates the configuration of the topic model with new *ModelConfig* value, provided by *config* parameter. When *config* is not specified the configuration is updated with `config()` value. Remember that some changes of the configuration are applied immediately after this call. For example, changes to `ModelConfig.topics_count` or `ModelConfig.topic_name` will be applied only during the next `Synchronize` call.

Note that changes `ModelConfig.topics_count` or `ModelConfig.topic_name` are only supported on an idle master component (e.g. in between iterations). Changing these values during an ongoing iteration may cause unexpected results.

topics_count()
Returns the number of topics in the model.

config()
Returns current *ModelConfig* of the topic model.

Synchronize(*decay_weight = 0.0*, *apply_weight = 1.0*, *invoke_regularizers = True*, *args=None*)
This operation updates the Phi matrix of the topic model with all model increments, collected since the last call to `Synchronize()` method. The Phi matrix is calculated according to *decay_weight* and *apply_weight* (refer to `SynchronizeModelArgs.decay_weight` for more details). Depending on *invoke_regularizers* parameter this operation may also invoke all regularizers.

Remember to call `Synchronize()` operation every time after call `MasterComponent.WaitIdle()`.

For more settings use *args* parameter (see *SynchronizeModelArgs* for all available options).

Initialize(*dictionary = None*, *args=None*)
Generates a random initial approximation for the Phi matrix of the topic model.

dictionary must be an instance of `Dictionary` class.

For more settings use *args* parameter (see *InitializeModelArgs* for all available options).

Export(*filename*)
Exports topic model into a file.

Import(*filename*)
Imports topic model from a file.

Overwrite(*topic_model*, *commit = True*)
Updates the model with new Phi matrix, defined by *topic_model* (*TopicModel*). This operation can be used to provide an explicit initial approximation of the topic model, or to adjust the model in between iterations.

Depending on the *commit* flag the change can be applied immediately (*commit = true*) or queued (*commit = false*). The default setting is to use *commit = true*. You may want to use *commit = false* if your model is too big to be updated in a single protobuf message. In this case you should split your model into parts, each part containing subset of all tokens, and then submit each part in separate Overwrite operation with *commit = false*. After that remember to call `MasterComponent.WaitIdle()` and `Model.Synchronize()` to propagate your change.

Enable()
Sets `ModelConfig.enabled` to *True* for the current topic model. This means that the model will be updated on `MasterComponent.InvokeIteration()`.

EnableScore(*score*)
By default model does calculate any scores even if they are created with `MasterComponent.CreateScore()`. Method *EnableScore* tells to the model that *score* should be applied to the model. Parameter *tau* defines the regularization coefficient of the regularizer. *score* must be an instance of `Score` class.

EnableRegularizer(*regularizer*, *tau*)
By default model does not use any regularizers even if they are created with `MasterComponent.CreateRegularizer()`. Method *EnableRegularizer* tells to the model that *regularizer* should be applied to the model. Parameter *tau* defines the regularization coefficient of the regularizer. *regularizer* must be an instance of `Regularizer` class.

Disable()
Sets `ModelConfig.enabled` to *False*‘ for the current topic model. This means that the model will not be updated on `MasterComponent.InvokeIteration()`, but the the scores for the model still will be collected.

9.6.4 Regularizer

class `artm.library.`**Regularizer**
This constructor must not be used explicitly. The only correct way of creating a Regularizer is through

MasterComponent.CreateRegularizer() method (or similar methods in *MasterComponent* class, dedicated to a particular type of the regularizer).

name()
Returns the string name of the regularizer.

Reconfigure(*type*, *config*)
Updates the configuration of the regularizer with new regularizer configuration, provided by *config* parameter. The *config* object can be, for example, of *SmoothSparseThetaConfig* type (or similar). The type must match the current type of the regularizer.

9.6.5 Score

class artm.library.**Score**
This constructor must not be used explicitly. The only correct way of creating a Score is through *MasterComponent.CreateScore()* method (or similar methods in *MasterComponent* class, dedicated to a particular type of the score).

name()
Returns the string name of the score.

GetValue(*model = None*, *batch = None*)
Retrieves the score for a specific model. For cumulative scores such as Perplexity of ThetaSparsity score it is possible to use *batch* argument.

9.6.6 Dictionary

class artm.library.**Dictionary**(*master_component*, *config*)
This constructor must not be used explicitly. The only correct way of creating a Dictionary is through *MasterComponent.CreateDictionary()* method.

name()
Returns the string name of the dictionary.

Reconfigure(*config*)
Updates the configuration of the dictionary with new *DictionaryConfig* value, provided by *config* parameter.

9.6.7 Visualizers

class artm.library.**Visualizers**
This class provides a set of static method to visualize some scores.

PrintTopTokensScore(*top_tokens_score*)

Prints the TopTokensScore.

PrintThetaSnippetScore(*theta_snippet_score*)

Prints the ThetaSnippetScore.

9.6.8 Exceptions

exception artm.library.**InternalError**
An exception class corresponding to *ARTM_INTERNAL_ERROR* error code.

exception artm.library.**ArgumentOutOfRangeException**
An exception class corresponding to *ARTM_ARGUMENT_OUT_OF_RANGE* error code.

exception artm.library.**InvalidMasterIdException**
An exception class corresponding to *ARTM_INVALID_MASTER_ID* error code.

exception artm.library.**CorruptedMessageException**
An exception class corresponding to *ARTM_CORRUPTED_MESSAGE* error code.

exception artm.library.**InvalidOperationException**
An exception class corresponding to *ARTM_INVALID_OPERATION* error code.

exception artm.library.**DiskReadException**
An exception class corresponding to *ARTM_DISK_READ_ERROR* error code.

exception artm.library.**DiskWriteException**
An exception class corresponding to *ARTM_DISK_WRITE_ERROR* error code.

9.6.9 Constants

artm.library.**Stream_Type_Global**

artm.library.**Stream_Type_ItemIdModulus**

artm.library.**RegularizerConfig_Type_DirichletTheta**

artm.library.**RegularizerConfig_Type_DirichletPhi**

artm.library.**RegularizerConfig_Type_SmoothSparseTheta**

artm.library.**RegularizerConfig_Type_SmoothSparsePhi**

artm.library.**RegularizerConfig_Type_DecorrelatorPhi**

artm.library.**ScoreConfig_Type_Perplexity**

artm.library.**ScoreData_Type_Perplexity**

artm.library.**ScoreConfig_Type_SparsityTheta**

artm.library.**ScoreData_Type_SparsityTheta**

artm.library.**ScoreConfig_Type_SparsityPhi**

artm.library.**ScoreData_Type_SparsityPhi**

artm.library.**ScoreConfig_Type_ItemsProcessed**

artm.library.**ScoreData_Type_ItemsProcessed**

artm.library.**ScoreConfig_Type_TopTokens**

artm.library.**ScoreData_Type_TopTokens**

artm.library.**ScoreConfig_Type_ThetaSnippet**

artm.library.**ScoreData_Type_ThetaSnippet**

artm.library.**ScoreConfig_Type_TopicKernel**

artm.library.**ScoreData_Type_TopicKernel**

artm.library.**PerplexityScoreConfig_Type_UnigramDocumentModel**

artm.library.**PerplexityScoreConfig_Type_UnigramCollectionModel**

artm.library.**CollectionParserConfig_Format_BagOfWordsUci**

9.7 Plain C interface of BigARTM

This document explains all public methods of the low level BigARTM interface.

9.7.1 Introduction

The goal of low level BigARTM interface is to expose all functionality of the library in a set of simple functions written in good old plain C language. This makes it easier to consume BigARTM from various programming environments. For example, the Python Interface of BigARTM uses ctypes module to call the low level BigARTM interface. Most programming environments also have similar functionality: PInvoke in C#, loadlibrary in Matlab, etc.

Note that most methods in this API accept a serialized binary representation of some Google Protocol Buffer message. Please, refer to Messages for more details about each particular message.

All methods in this API return an integer value. Negative return values represent an error code. See *error codes* for the list of all error codes. To get corresponding error message as string use `ArtmGetLastErrorMessage()`. Non-negative return values represent a success, and for some API methods might also incorporate some useful information. For example, `ArtmCreateMasterComponent()` returns the ID of newly created master component, and `ArtmRequestTopicModel()` returns the length of the buffer that should be allocated before calling `ArtmCopyRequestResult()`.

9.7.2 ArtmCreateMasterComponent

int **ArtmCreateMasterComponent**(int *length*, const char* *master_component_config*)

Creates a master component.

Parameters

- **master_component_config** (*const_char**) – Serialized *MasterComponentConfig* message, describing the configuration of the master component.
- **length** (*int*) – The length in bytes of the *master_component_config* message.

Returns In case of success, a non-negative ID of the master component, otherwise one of the *error codes*.

The ID, returned by this operation, is required by most methods in this API. Several master components may coexist in the same process. In such case any two master components with different IDs can not share any common data, and thus they are completely independent from each other.

9.7.3 ArtmReconfigureMasterComponent

int **ArtmReconfigureMasterComponent**(int *master_id*, int *length*, const char* *master_component_config*)

Changes the configuration of the master component.

Parameters

- **master_id** (*int*) – The ID of a master component returned by `ArtmCreateMasterComponent()` method.
- **master_component_config** (*const_char**) – Serialized *MasterComponentConfig* message, describing the new configuration of the master component.
- **length** (*int*) – The length in bytes of the *master_component_config* message.

Returns A zero value if operation succeeded, otherwise one of the *error codes*.

9.7.4 ArtmDisposeMasterComponent

int **ArtmDisposeMasterComponent**(int *master_id*)

Disposes master component together with all its models, regularizers and dictionaries.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.

Returns This operation always returns *ARTM_SUCCESS*.

This operation releases memory and other unmanaged resources, used by the master component.

After this operation the *master_id* value becames invalid and must not be used in other operations.

9.7.5 ArtmCreateModel

int **ArtmCreateModel**(int *master_id*, int *length*, const char* *model_config*)

Defines a new topic model.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **model_config** (*const_char**) – Serialized *ModelConfig* message, describing the configuration of the topic model.
- **length** (*int*) – The length in bytes of the *model_config* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

Note that this method only defines the configuration a topic model, but does not tune it. Use *ArtmInvokeIteration()* method to process the collection of textual documents, and then *ArtmRequestTopicModel()* to retrieve the resulting topic model.

It is important to notice that *model_config* must have a unique value in the *ModelConfig.name* field, that can be further used to identify the model (for example in *ArtmRequestTopicModel()* call).

9.7.6 ArtmReconfigureModel

int **ArtmReconfigureModel**(int *master_id*, int *length*, const char* *model_config*)

Updates the configuration of topic model.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **model_config** (*const_char**) – Serialized *ModelConfig* message, describing the new configuration of the topic model.
- **length** (*int*) – The length in bytes of the *model_config* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.7 ArtmDisposeModel

int **ArtmDisposeModel**(int *master_id*, const char* *model_name*)

Explicitly delete a specific topic model. All regularizers within specific master component are also deleted automatically by *ArtmDisposeMasterComponent()*.

After *ArtmDisposeModel()* the *model_name* became invalid and shell not be used in *ArtmRequestScore()*, *ArtmRequestTopicModel()*, *ArtmRequestThetaMatrix()* or any other method (or protobuf message) that require *model_name*.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **model_name** (*const_char*) – A string identified of the model that should be deleted.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.8 ArtmCreateRegularizer

int **ArtmCreateRegularizer**(int *master_id*, int *length*, const char* *regularizer_config*)

Creates a new regularizer.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **regularizer_config** (*const_char*) – Serialized *RegularizerConfig* message, describing the configuration of a new regularizer.
- **length** (*int*) – The length in bytes of the *regularizer_config* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

This operation only creates the regularizer so that it can be used by topic models. To actually apply the regularizer you should include its name in *ModelConfig.regularizer_name* list of a model config.

9.7.9 ArtmReconfigureRegularizer

int **ArtmReconfigureRegularizer**(int *master_id*, int *length*, const char* *regularizer_config*)

Updates the configuration of the regularizer.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **regularizer_config** (*const_char*) – Serialized *RegularizerConfig* message, describing the configuration of a new regularizer.
- **length** (*int*) – The length in bytes of the *regularizer_config* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.10 ArtmDisposeRegularizer

int **ArtmDisposeRegularizer**(int *master_id*, const char* *regularizer_name*)

Explicitly delete a specific regularizer. All regularizers within specific master component are also deleted automatically by *ArtmDisposeMasterComponent()*.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **regularizer_name** (*const_char**) – A string identified of the regularizer that should be deleted.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.11 ArtmCreateDictionary

int **ArtmCreateDictionary**(int *master_id*, int *length*, const char* *dictionary_config*)

Creates a new dictionary.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **dictionary_config** (*const_char**) – Serialized *DictionaryConfig* message, describing the configuration of a new dictionary.
- **length** (*int*) – The length in bytes of the *dictionary_config* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.12 ArtmReconfigureDictionary

int **ArtmReconfigureDictionary**(int *master_id*, int *length*, const char* *dictionary_config*)

Updates the dictionary.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **dictionary_config** (*const_char**) – Serialized *DictionaryConfig* message, describing the new configuration of the dictionary.
- **length** (*int*) – The length in bytes of the *dictionary_config* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.13 ArtmDisposeDictionary

int **ArtmDisposeDictionary**(int *master_id*, const char* *dictionary_name*)

Explicitly delete a specific dictionary. All dictionaries within specific master component are also deleted automatically by *ArtmDisposeMasterComponent()*.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **dictionary_name** (*const_char*)* – A string identified of the dictionary that should be deleted.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.14 ArtmAddBatch

int **ArtmAddBatch**(int *master_id*, int *length*, const char* *add_batch_args*)
Adds batch for processing.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **add_batch_args** (*const_char*)* – Serialized *AddBatchArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *add_batch_args* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.15 ArtmInvokeIteration

int **ArtmInvokeIteration**(int *master_id*, int *length*, const char* *invoke_iteration_args*)
Invokes several iterations over the collection.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **char* invoke_iteration_args** (*const*) – Serialized *InvokeIterationArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *invoke_iteration_args* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.16 ArtmSynchronizeModel

int **ArtmSynchronizeModel**(int *master_id*, int *length*, const char* *sync_model_args*)
Synchronizes topic model.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **sync_model_args** (*const_char*)* – Serialized *SynchronizeModelArgs* message, describing the arguments of this operation.

- **length** (*int*) – The length in bytes of the *sync_model_args* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

This operation updates the Phi matrix of the topic model with all model increments, collected since last call to ArtmSynchronizeModel. In addition, this operation invokes all Phi-regularizers for the requested topic model.

9.7.17 ArtmInitializeModel

int **ArtmInitializeModel** (int *master_id*, int *length*, const char* *init_model_args*)

Initializes the phi matrix of a topic model with some random initial approximation.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **init_model_args** (*const_char**) – Serialized *InitializeModelArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *init_model_args* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.18 ArtmExportModel

int **ArtmExportModel** (int *master_id*, int *length*, const char* *export_model_args*)

Exports phi matrix into a file.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **export_model_args** (*const_char**) – Serialized *ExportModelArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *export_model_args* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.19 ArtmImportModel

int **ArtmImportModel** (int *master_id*, int *length*, const char* *import_model_args*)

Import phi matrix from a file.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **import_model_args** (*const_char**) – Serialized *ImportModelArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *import_model_args* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.20 ArtmWaitIdle

int **ArtmWaitIdle**(int *master_id*, int *length*, const char* *wait_idle_args*)

Awaits for ongoing iterations.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **wait_idle_args** (*const_char**) – Serialized *WaitIdleArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *wait_idle_args* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.21 ArtmOverwriteTopicModel

int **ArtmOverwriteTopicModel**(int *master_id*, int *length*, const char* *topic_model*)

This operation schedules an update of an entire topic model or of it subpart.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **topic_model** (*const_char**) – Serialized *TopicModel* message, describing the new topic model.
- **length** (*int*) – The length in bytes of the *topic_model* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

Note that this operation only schedules the update of a topic model. To make sure the update is completed you must call *ArtmWaitIdle()* and *ArtmSynchronizeModel()*. Remember that by default *ArtmSynchronizeModel()* will calculate all regularizers enabled in the configuration of the topic model. The may result in a different topic model than the one you passed as *topic_model* parameter. To avoid this behavior set *SynchronizeModelArgs.invoke_regularizers* to `false`.

9.7.22 ArtmRequestThetaMatrix

int **ArtmRequestThetaMatrix**(int *master_id*, int *length*, const char* *get_theta_args*)

Requests theta matrix. Use `ArtmCopyRequestResult()` to copy the resulting message.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by *ArtmCreateMasterComponent()* method.
- **get_theta_args** (*const_char**) – Serialized *GetThetaMatrixArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *get_theta_args* message.

Returns In case of success, returns the length in bytes of a buffer that should be allocated on callers site and then passed to `ArtmCopyRequestResult()` method. This will populate the buffer with *ThetaMatrix* message, carrying the requested information. In case of a failure, returns one of the *error codes*.

9.7.23 ArtmRequestTopicModel

int **ArtmRequestTopicModel**(int *master_id*, int *length*, const char* *get_model_args*)
Requests topic model.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by `ArtmCreateMasterComponent()` method.
- **get_model_args** (*const_char**) – Serialized *GetTopicModelArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *get_model_args* message.

Returns In case of success, returns the length in bytes of a buffer that should be allocated on callers site and then passed to `ArtmCopyRequestResult()` method. This will populate the buffer with *TopicModel* message, carrying the requested information. In case of a failure, returns one of the *error codes*.

9.7.24 ArtmRequestRegularizerState

int **ArtmRequestRegularizerState**(int *master_id*, const char* *regularizer_name*)
Requests state of a specific regularizer.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by `ArtmCreateMasterComponent()` method.
- **regularizer_name** (*const_char**) – A string identified of the regularizer.

Returns In case of success, returns the length in bytes of a buffer that should be allocated on callers site and then passed to `ArtmCopyRequestResult()` method. This will populate the buffer with *RegularizerInternalState* message, carrying the requested information. In case of a failure, returns one of the *error codes*.

9.7.25 ArtmRequestScore

int **ArtmRequestScore**(int *master_id*, int *length*, const char* *get_score_args*)
Request the result of score calculation.

Parameters

- **master_id** (*int*) – The ID of a master component, returned by `ArtmCreateMasterComponent()` method.
- **const_char*** – get_score_args: Serialized *GetScoreValueArgs* message, describing the arguments of this operation.
- **length** (*int*) – The length in bytes of the *get_score_args* message.

Returns In case of success, returns the length in bytes of a buffer that should be allocated on callers site and then passed to `ArtmCopyRequestResult()` method. This will populate the buffer with *ScoreData* message, carrying the requested information. In case of a failure, returns one of the *error codes*.

9.7.26 ArtmRequestParseCollection

int **ArtmRequestParseCollection**(int *length*, const char* *collection_parser_config*)

Parses a text collection into a set of batches and stores them on disk. Returns a *DictionaryConfig* message that lists all tokens, occured in the collection.

Check the description of *CollectionParserConfig* message for more details about this operation.

Parameters

- **const_char*** – collection_parser_config: Serialized *CollectionParserConfig* message, describing the configuration the collection parser.
- **length** (*int*) – The length in bytes of the *collection_parser_config* message.

Returns In case of success, returns the length in bytes of a buffer that should be allocated on callers site and then passed to `ArtmCopyRequestResult()` method. The buffer will contain *DictionaryConfig* message, that lists all unique tokens from the collection being parsed. In case of a failure, returns one of the *error codes*.

Warning: The following error most likelly indicate that you are trying to parse a very large file in 32 bit version of BigARTM.
`InternalError : failed mapping view: The parameter is incorrect`
Try to use 64 bit BigARTM to workaround this issue.

9.7.27 ArtmRequestLoadDictionary

int **ArtmRequestLoadDictionary**(const char* *filename*)

Loads a *DictionaryConfig* message from disk.

Parameters

- **const_char*** – filename: A full file name of a file that contains a serialized DictionaryConfig message.

Returns In case of success, returns the length in bytes of a buffer that should be allocated on callers site and then passed to `ArtmCopyRequestResult()` method. The buffer will contain the resulting *DictionaryConfig* message. In case of a failure, returns one of the *error codes*.

This method can be used to load `CollectionParserConfig.dictionary_file_name` or `CollectionParserConfig.cooccurrence_file_name` dictionaries, saved by *ArtmRequestParseCollection* method.

9.7.28 ArtmRequestLoadBatch

int **ArtmRequestLoadBatch**(const char* *filename*)

Loads a *Batch* message from disk.

Parameters

- **const_char*** – filename: A full file name of a file that contains a serialized Batch message.

Returns In case of success, returns the length in bytes of a buffer that should be allocated on callers site and then passed to *ArtmCopyRequestResult()* method. The buffer will contain the resulting *Batch* message. In case of a failure, returns one of the *error codes*.

This method can be used to load batches saved by *ArtmRequestParseCollection* method or *ArtmSaveBatch* method.

9.7.29 ArtmCopyRequestResult

int **ArtmCopyRequestResult**(int *length*, char* *address*)
Copies the result of the last request.

Parameters

- **const_char*** – address: Target memory location to copy the data.
- **length** (*int*) – The length in bytes of the *address* buffer.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.30 ArtmSaveBatch

int **ArtmSaveBatch**(const char* *disk_path*, int *length*, const char* *batch*)
Saves a *Batch* message to disk.

Parameters

- **const_char*** – disk_path: A floder where to save the batch.
- **batch** (*const_char**) – Serialized *Batch* message to save.
- **length** (*int*) – The length in bytes of the *batch* message.

Returns Returns *ARTM_SUCCESS* value if operation succeeded, otherwise returns one of the *error codes*.

9.7.31 ArtmGetLastErrorMessage

const char* **ArtmGetLastErrorMessage**()
Retrieves the textual error message, occured during the last failing request.

9.7.32 Error codes

```
#define ARTM_SUCCESS 0
#define ARTM_STILL_WORKING -1
#define ARTM_INTERNAL_ERROR -2
#define ARTM_ARGUMENT_OUT_OF_RANGE -3
#define ARTM_INVALID_MASTER_ID -4
#define ARTM_CORRUPTED_MESSAGE -5
#define ARTM_INVALID_OPERATION -6
#define ARTM_DISK_READ_ERROR -7
#define ARTM_DISK_WRITE_ERROR -8
```

ARTM_SUCCESS
The API call succeeded.

ARTM_STILL_WORKING
This error code is applicable only to *ArtmWaitIdle()*. It indicates that library is still processing the collection. Try to retrieve results later.

ARTM_INTERNAL_ERROR
The API call failed due to internal error in BigARTM library. Please, collect steps to reproduce this issue and report it with BigARTM issue tracker.

ARTM_ARGUMENT_OUT_OF_RANGE
The API call failed because one or more values of an argument are outside the allowable range of values as defined by the invoked method.

ARTM_INVALID_MASTER_ID
An API call that require *master_id* parameter failed because MasterComponent with given ID does not exist.

ARTM_CORRUPTED_MESSAGE
Unable to deserialize protocol buffer message.

ARTM_INVALID_OPERATION
The API call is invalid in current state or due to provided parameters.

ARTM_DISK_READ_ERROR
The required files coult not be read from disk.

ARTM_DISK_WRITE_ERROR
The required files could not be writtent to disk.

9.8 C++ interface

BigARTM C++ interface is currently not documented. The main entry point is `MasterModel` class from `src/artm/cpp_interface.cc`. Please referto `src/bigartm//srcmain.cc` for usage examples, and ask questions at bigartm-users or open a new issue.

```
class MasterModel {
 public:
  explicit MasterModel(const MasterModelConfig& config);
  ~MasterModel();

  int id() const { return id_; }
  MasterComponentInfo info() const;  // misc. diagnostics information

  const MasterModelConfig& config() const { return config_; }
  MasterModelConfig* mutable_config() { return &config_; }
  void Reconfigure();  // apply MasterModel::config()

  // Operations to work with dictionary through disk
  void GatherDictionary(const GatherDictionaryArgs& args);
  void FilterDictionary(const FilterDictionaryArgs& args);
  void ImportDictionary(const ImportDictionaryArgs& args);
  void ExportDictionary(const ExportDictionaryArgs& args);
  void DisposeDictionary(const std::string& dictionary_name);

  // Operations to work with dictinoary through memory
  void CreateDictionary(const DictionaryData& args);
  DictionaryData GetDictionary(const GetDictionaryArgs& args);
```

```
// Operatinos to work with batches through memory
void ImportBatches(const ImportBatchesArgs& args);
void DisposeBatch(const std::string& batch_name);

// Operations to work with model
void InitializeModel(const InitializeModelArgs& args);
void ImportModel(const ImportModelArgs& args);
void ExportModel(const ExportModelArgs& args);
void FitOnlineModel(const FitOnlineMasterModelArgs& args);
void FitOfflineModel(const FitOfflineMasterModelArgs& args);

// Apply model to batches
ThetaMatrix Transform(const TransformMasterModelArgs& args);
ThetaMatrix Transform(const TransformMasterModelArgs& args, Matrix* matrix);

// Retrieve operations
TopicModel GetTopicModel(const GetTopicModelArgs& args);
TopicModel GetTopicModel(const GetTopicModelArgs& args, Matrix* matrix);
ThetaMatrix GetThetaMatrix(const GetThetaMatrixArgs& args);
ThetaMatrix GetThetaMatrix(const GetThetaMatrixArgs& args, Matrix* matrix);

// Retrieve scores
ScoreData GetScore(const GetScoreValueArgs& args);
template <typename T>
T GetScoreAs(const GetScoreValueArgs& args);
```

Warning: What follows below in this page is really outdated.

In addition to this page consider to look at Plain C interface of BigARTM, Python Interface or Messages. These documentation files are also to certain degree relevant for C++ interface, because C++ interface is quite similar to Python interface and share the same Protobuf messages.

9.8.1 MasterComponent

class `MasterComponent`

`MasterComponent`(**const** MasterComponentConfig &*config*)
Creates a master component with configuration defined by *MasterComponentConfig* message.

void **`Reconfigure`**(**const** MasterComponentConfig &*config*)
Updates the configuration of the master component.

const MasterComponentConfig &**`config`**() **const**
Returns current configuration of the master component.

MasterComponentConfig ***`mutable_config`**()
Returns mutable configuration of the master component. Remember to call *`Reconfigure()`* to propagate your changes to master component.

void **`InvokeIteration`**(int *iterations_count* = 1)
Invokes certain number of iterations.

bool **`AddBatch`**(**const** Batch &*batch*, bool *reset_scores*)
Adds batch to the processing queue.

bool **WaitIdle**(int *timeout* = -1)
Waits for iterations to be completed. Returns true if BigARTM completed before the specific timeout, otherwise false.

std::shared_ptr<TopicModel> **GetTopicModel**(**const** std::string &*model_name*)
Retrieves Phi matrix of a specific topic model. The resulting message *TopicModel* will contain information about token weights distribution across topics.

std::shared_ptr<TopicModel> **GetTopicModel**(**const** GetTopicModelArgs &*args*)
Retrieves Phi matrix based on extended parameters, specified in *GetTopicModelArgs* message. The resulting message *TopicModel* will contain information about token weights distribution across topics.

std::shared_ptr<ThetaMatrix> **GetThetaMatrix**(**const** std::string &*model_name*)
Retrieves Theta matrix of a specific topic model. The resulting message *ThetaMatrix* will contain information about items distribution across topics. Remember to set `MasterComponentConfig.cache_theta` prior to the last iteration in order to gather Theta matrix.

std::shared_ptr<ThetaMatrix> **GetThetaMatrix**(**const** GetThetaMatrixArgs &*args*)
Retrieves Theta matrix based on extended parameters, specified in *GetThetaMatrixArgs* message. The resulting message *ThetaMatrix* will contain information about items distribution across topics.

std::shared_ptr<T> **GetScoreAs**<T>(**const** *Model* &*model*, **const** std::string &*score_name*)
Retrieves given score for a specific model. Template argument must match the specific *ScoreData* type of the score (for example, *PerplexityScore*).

9.8.2 Model

class Model

Model(**const** *MasterComponent* &*master_component*, **const** ModelConfig &*config*)
Creates a topic model defined by *ModelConfig* inside given `MasterComponent`.

void **Reconfigure**(**const** ModelConfig &*config*)
Updates the configuration of the model.

const std::string &**name**() **const**
Returns the name of the model.

const ModelConfig &**config**() **const**
Returns current configuration of the model.

ModelConfig ***mutable_config**()
Returns mutable configuration of the model. Remember to call `Reconfigure()` to propagate your changes to the model.

void **Overwrite**(**const** TopicModel &*topic_model*, bool *commit* = true)
Updates the model with new Phi matrix, defined by *topic_model*. This operation can be used to provide an explicit initial approximation of the topic model, or to adjust the model in between iterations.

Depending on the *commit* flag the change can be applied immediately (*commit* = *true*) or queued (*commit* = *false*). The default setting is to use *commit* = *true*. You may want to use *commit* = *false* if your model is too big to be updated in a single protobuf message. In this case you should split your model into parts, each part containing subset of all tokens, and then submit each part in separate Overwrite operation with *commit* = *false*. After that remember to call `MasterComponent::WaitIdle()` and `Synchronize()` to propagate your change.

void **Initialize**(**const** *Dictionary* &*dictionary*)
Initialize topic model based on the `Dictionary`. Each token from the dictionary will be included in the model with randomly generated weight.

void **Export**(**const** string &*file_name*)
Exports topic model into a file.

void **Import**(**const** string &*file_name*)
Imports topic model from a file.

void **Synchronize**(double *decay_weight*, double *apply_weight*, bool *invoke_regularizers*)
Synchronize the model.

This operation updates the Phi matrix of the topic model with all model increments, collected since the last call to `Synchronize()` method. The weights in the Phi matrix are set according to *decay_weight* and *apply_weight* values (refer to `SynchronizeModelArgs.decay_weight` for more details). Depending on *invoke_regularizers* parameter this operation may also invoke all regularizers.

Remember to call `Model::Synchronize()` operation every time after calling `MasterComponent::WaitIdle()`.

void **Synchronize**(**const** SynchronizeModelArgs &*args*)
Synchronize the model based on extended arguments *SynchronizeModelArgs*.

9.8.3 Regularizer

class Regularizer

Regularizer(**const** *MasterComponent* &*master_component*, **const** RegularizerConfig &*config*)
Creates a regularizer defined by *RegularizerConfig* inside given `MasterComponent`.

void **Reconfigure**(**const** RegularizerConfig &*config*)
Updates the configuration of the regularizer.

const RegularizerConfig &**config**() **const**
Returns current configuration of the regularizer.

RegularizerConfig ***mutable_config**()
Returns mutable configuration of the regularizer. Remember to call `Reconfigure()` to propagate your changes to the regularizer.

9.8.4 Dictionary

class Dictionary

Dictionary(**const** *MasterComponent* &*master_component*, **const** DictionaryConfig &*config*)
Creates a dictionary defined by *DictionaryConfig* inside given `MasterComponent`.

void **Reconfigure**(**const** DictionaryConfig &*config*)
Updates the configuration of the dictionary.

const std::string **name**() **const**
Returns the name of the dictionary.

const DictionaryConfig &**config**() **const**
Returns current configuration of the dictionary.

9.8.5 Utility methods

void **SaveBatch**(**const** Batch &*batch*, **const** std::string &*disk_path*)
Saves *Batch* into a specific folder. The name of the resulting file will be autogenerated, and the extention set to *.batch*

std::shared_ptr<DictionaryConfig> **LoadDictionary**(**const** std::string &*filename*)
Loads the *DictionaryConfig* message from a specific file on disk. *filename* must represent full disk path to the dictionary file.

std::shared_ptr<Batch> **LoadBatch**(**const** std::string &*filename*)
Loads the *Batch* message from a specific file on disk. *filename* must represent full disk path to the batch file, including *.batch* extention.

std::shared_ptr<DictionaryConfig> **ParseCollection**(**const** CollectionParserConfig &*config*)
Parses a text collection as defined by *CollectionParserConfig* message. Returns an instance of *DictionaryConfig* which carry all unique words in the collection and their frequencies.

9.9 Windows distribution

This chapter describes content of BigARTM distribution package for Windows, available at https://github.com/bigartm/bigartm/releases.

`bin/`	Precompiled binaries of BigARTM for Windows. This folder must be added to `PATH` system variable.
`bin/artm.dll`	Core functionality of the BigARTM library.
`bin/cpp_client.exe`	Command line utility allows to perform simple experiments with BigARTM. Remember that not all BigARTM features are available through cpp_client, but it can serve as a good starting point to learn basic functionality. For further details refer to `/ref/cpp_client`.
`protobuf/`	A minimalistic version of Google Protocol Buffers (https://code.google.com/p/protobuf/) library, required to run BigARTM from Python. To setup this package follow the instructions in `protobuf/python/README` file.
`python/artm/`	Python programming interface to BigARTM library. This folder must be added to `PYTHONPATH` system variable.
`library.py`	Implements all classes of BigARTM python interface.
`messages_pb2.py`	Contains all protobuf messages that can be transfered in and out BigARTM core library. Most common features are exposed with their own API methods, so normally you do not use python protobuf messages to operate BigARTM.
`python/examples/`	Python examples of how to use BigARTM: Files `docword.kos.txt` and `vocab.kos.txt` represent a simple collection of text files in Bag-Of-Words format. The files are taken from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Bag+of+Words).
`src/`	

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

a

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

Z