BigARTM: Open Source Library for Regularized Multimodal Topic Modeling of Large Collections

Konstantin Vorontsov[1(✉)], Oleksandr Frei[2], Murat Apishev[3], Peter Romov[1], and Marina Dudarenko[3]

[1] Yandex, Moscow Institute of Physics and Technology, Moscow, Russia
voron@forecsys.ru, peter@romov.ru
[2] Schlumberger Information Solutions, Oslo, Norway
oleksandr.frei@gmail.com
[3] Lomonosov Moscow State University, Moscow, Russia
great-mel@yandex.ru, m.dudarenko@gmail.com

Abstract. Probabilistic topic modeling of text collections is a powerful tool for statistical text analysis. In this paper we announce the BigARTM open source project (http://bigartm.org) for regularized multimodal topic modeling of large collections. Several experiments on Wikipedia corpus show that BigARTM performs faster and gives better perplexity comparing to other popular packages, such as Vowpal Wabbit and Gensim. We also demonstrate several unique BigARTM features, such as additive combination of regularizers, topic sparsing and decorrelation, multimodal and multilanguage modeling, which are not available in the other software packages for topic modeling.

Keywords: Probabilistic topic modeling · Probabilistic latent sematic analysis · Latent dirichlet allocation · Additive regularization of topic models · Stochastic matrix factorization · EM-algorithm · BigARTM

1 Introduction

Topic modeling is a rapidly developing branch of statistical text analysis [1]. Topic model reveals a hidden thematic structure of a text collection and finds a compressed representation of each document in terms of its topics. Practical applications of topic models include many areas, such as information retrieval for long-text queries, classification, categorization, summarization and segmentation of texts. Topic models are increasingly used for non-textual and heterogeneous data including signals, images, video and networks. More ideas, models and applications are outlined in the survey [4].

From a statistical point of view, a probabilistic topic model (PTM) defines each topic by a multinomial distribution over words, and then describes each document with a multinomial distribution over topics. From an optimizational point of view, topic modeling can be considered as a special case of approximate stochastic matrix factorization. To learn a factorized representation of a text

M.Y. Khachay et al. (Eds.): AIST 2015, CCIS 542, pp. 370–381, 2015.
DOI: 10.1007/978-3-319-26123-2_36

collection is an ill-posed problem, which has an infinite set of solutions. A typical approach in this case is to apply regularization techniques, which impose problem-specific constrains and ultimately lead to a better solution.

Modern literature on topic modeling offers hundreds of models adapted to different situations. Nevertheless, most of these models are too difficult for practitioners to quickly understand, adapt and embed into applications. This leads to a common practice of tasting only the basic out-of-date models such as *Probabilistic Latent Semantic Analysis*, PLSA [6] and *Latent Dirichlet Allocation*, LDA [3]. Most practical inconveniences are rooted in Bayesian learning, which is the dominating approach in topic modeling. Bayesian inference of topic models requires a laborious mathematical work, which prevents flexible unification, modification, selection, and combination of topic models.

In this paper we announce **the BigARTM open source project** for regularized multimodal topic modeling of large collections, http://bigartm.org. The theory behind BigARTM is based on a non-Bayesian multicriteria approach — *Additive Regularization of Topic Models*, ARTM [11]. In ARTM a topic model is learned by maximizing a weighted sum of the log-likelihood and additional regularization criteria. The optimization problem is solved by a general regularized expectation-maximization (EM) algorithm, which can be applied to an arbitrary combination of regularization criteria. Many known Bayesian topic models were revisited in terms of ARTM in [12,13]. Compared to the Bayesian approach, ARTM makes it easier to design, infer and combine topic models, thus reducing the barrier for entering into topic modeling research field.

BigARTM source code is released under the New BSD License, which permits free commercial and non-commercial usage. The core of the library is written in C++ and is exposed via two equally rich APIs for C++ and Python. The library is cross-platform and can be built for Linux, Windows and OS X in both 32 and 64 bit configuration. In our experiments on Wikipedia corpus BigARTM performs better than Vowpal Wabbit LDA and Gensim libraries in terms of perplexity and runtime. Comparing to the other libraries BigARTM offers several additional features, such as regularization and multi-modal topic modeling.

The rest of the paper is organized as follows. In Sect. 2 we introduce a multimodal topic model for documents with metadata. In Sect. 3 we generalize the fast online algorithm [5] to multimodal ARTM. In Sect. 4 we describe parallel architecture and implementation details of the BigARTM library. In Sect. 5 we report results of our experiments on large datasets. In Sect. 6 we discuss advantages, limitations and open problems of BigARTM.

2 Multimodal Regularized Topic Model

Let D denote a finite set (collection) of texts and W^1 denote a finite set (vocabulary) of all terms from these texts. Each term can represent a single word or a key phrase. A document can contain not only words, but also terms of other modalities. Each modality is defined by a finite set (vocabulary) of terms W^m, $m = 1, \ldots, M$. Examples of not-word modalities are: authors, class or category

labels, date-time stamps, references to/from other documents, entities mentioned in texts, objects found in the images associated with the documents, users that read or downloaded documents, advertising banners, etc.

Assume that each term occurrence in each document refers to some latent topic from a finite set of topics T. Text collection is considered to be a sample of triples (w_i, d_i, t_i), $i = 1, \ldots, n$, drawn independently from a discrete distribution $p(w, d, t)$ over the finite space $W \times D \times T$, where $W = W^1 \sqcup \cdots \sqcup W^m$ is a disjoint union of the vocabularies across all modalities. Terms w_i and documents d_i are observable variables, while topics t_i are latent variables.

Following the idea of Correspondence LDA [2] and Dependency LDA [9] we introduce a topic model for each modality:

$$p(w \,|\, d) = \sum_{t \in T} p(w \,|\, t)\, p(t \,|\, d) = \sum_{t \in T} \phi_{wt} \theta_{td}, \quad d \in D,\ w \in W^m,\ m = 1, \ldots, M.$$

The parameters $\theta_{td} = p(t \,|\, d)$ and $\phi_{wt} = p(w \,|\, t)$ form matrices $\Theta = \big(\theta_{td}\big)_{T \times D}$ of *topic probabilities for the documents*, and $\Phi^m = \big(\phi_{wt}\big)_{W^m \times T}$ of *term probabilities for the topics*. The matrices Φ^m, if stacked vertically, form a $W \times T$-matrix Φ. Matrices Φ^m and Θ are *stochastic*, that is, their vector-columns represent discrete distributions. Usually $|T|$ is much smaller than $|D|$ and $|W|$.

To learn parameters Φ^m, Θ from the multimodal text collection we maximize the log-likelihood for each m-th modality:

$$\mathscr{L}_m(\Phi^m, \Theta) = \sum_{d \in D} \sum_{w \in W^m} n_{dw} \ln p(w \,|\, d) \to \max_{\Phi^m, \Theta},$$

where n_{dw} is the number of occurrences of the term $w \in W^m$ in the document d. Note that topic distributions of documents Θ are common for all modalities. Following the ARTM approach, we add a regularization penalty term $R(\Phi, \Theta)$ and solve a constrained multicriteria optimization problem via scalarization:

$$\sum_{m=1}^{M} \tau_m \mathscr{L}_m(\Phi^m, \Theta) + R(\Phi, \Theta) \to \max_{\Phi, \Theta}; \tag{1}$$

$$\sum_{w \in W^m} \phi_{wt} = 1,\ \phi_{wt} \geqslant 0; \qquad \sum_{t \in T} \theta_{td} = 1,\ \theta_{td} \geqslant 0. \tag{2}$$

The local maximum (Φ, Θ) of the problem (1), (2) satisfies the following system of equations with auxiliary variables $p_{tdw} = p(t \,|\, d, w)$:

$$p_{tdw} = \operatorname*{norm}_{t \in T}\big(\phi_{wt} \theta_{td}\big); \tag{3}$$

$$\phi_{wt} = \operatorname*{norm}_{w \in W^m}\left(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}}\right); \quad n_{wt} = \sum_{d \in D} n_{dw} p_{tdw}; \tag{4}$$

$$\theta_{td} = \underset{t \in T}{\text{norm}}\left(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}\right); \quad n_{td} = \sum_{w \in d} \tau_{m(w)} n_{dw} p_{tdw}; \tag{5}$$

where operator $\underset{t \in T}{\text{norm}}\, x_t = \frac{\max\{x_t, 0\}}{\sum_{s \in T} \max\{x_s, 0\}}$ transforms a vector $(x_t)_{t \in T}$ to a discrete distribution; $m(w)$ is the modality of the term w, so that $w \in W^{m(w)}$.

The system of Eqs. (3)–(5) follows from Karush–Kuhn–Tucker conditions. It can be solved by various numerical methods. Particularly, the simple-iteration method is equivalent to the EM algorithm, which is typically used in practice. For single modality ($M = 1$) it gives the regularized EM algorithm proposed in [11]. With no regularization ($R = 0$) it corresponds to PLSA [6].

Many Bayesian topic models can be considered as special cases of ARTM with different regularizers R, as shown in [12,13]. For example, LDA [3] corresponds to the entropy smoothing regularizer.

Due to the unified framework of additive regularization BigARTM can build topic models for various applications simply by choosing a suitable combination of regularizers from a build-in user extendable library.

3 Online Topic Modeling

Following the idea of Online LDA [5] we split the collection D into batches D_b, $b = 1, \ldots, B$, and organize EM iterations so that each document vector θ_d is iterated until convergence at a constant matrix Φ, see Algorithms 1 and 2. Matrix Φ is updated rarely, after all documents from the batch are processed. For a large collection matrix Φ often stabilizes after small initial part of the collection. Therefore a single pass through the collection might be sufficient to learn a topic model.

Algorithm 1 does not specify how often to synchronize Φ matrix at steps 5–8. It can be done after every batch or less frequently (for instance if $\frac{\partial R}{\partial \phi_{wt}}$ takes long time to evaluate). This flexibility is especially important for concurrent implementation of the algorithm, where multiple batches are processed in parallel. In this case synchronization can be triggered when a fixed number of documents had been processed since the last synchronization.

The online reorganization of the EM iterations is not necessarily associated with Bayesian inference used in [5]. Different topic models, from PLSA to multimodal and regularized models, can be learned by the above online EM algorithm.

4 BigARTM Architecture

The main goal for BigARTM architecture is to ensure a constant memory usage regardless of the collection size. For this reason each D_b batch is stored on disk in a separate file, and only a limited number of batches is loaded into the main memory at any given time. The entire Θ matrix is never stored in the memory. As a result, the memory usage stays constant regardless of the size of the collection.

Algorithm 1. Online EM-algorithm for multimodal ARTM

Input: collection $\{D_b\colon b = 1, \ldots, B\}$, discounting factor $\rho \in (0, 1]$;
Output: matrix Φ;

initialize ϕ_{wt} for all $w \in W$ and $t \in T$;
$n_{wt} := 0$, $\tilde{n}_{wt} := 0$ for all $w \in W$ and $t \in T$;
for all *batches* D_b, $b = 1, \ldots, B$
  $(\tilde{n}_{wt}) := (\tilde{n}_{wt}) + \mathsf{ProcessBatch}(D_b, \phi_{wt})$;
  if *(synchronize)* **then**
    $n_{wt} := \rho n_{wt} + \tilde{n}_{dw}$ for all $w \in W$ and $t \in T$;
    $\phi_{wt} := \underset{w \in W^m}{\operatorname{norm}}\left(n_{wt} + \phi_{wt}\frac{\partial R}{\partial \phi_{wt}}\right)$ for all $w \in W^m$, $m = 1, \ldots, M$ and $t \in T$;
    $\tilde{n}_{wt} := 0$ for all $w \in W$ and $t \in T$;

Algorithm 2. $\mathsf{ProcessBatch}(D_b, \phi_{wt})$

Input: batch D_b, matrix ϕ_{wt};
Output: matrix $(\tilde{n}_{wt})$;

$\tilde{n}_{wt} := 0$ for all $w \in W$ and $t \in T$;
for all $d \in D_b$
  initialize $\theta_{td} := \frac{1}{|T|}$ for all $t \in T$;
  repeat
    $p_{tdw} := \underset{t \in T}{\operatorname{norm}}\left(\phi_{wt}\theta_{td}\right)$ for all $t \in T$;
    $n_{td} := \sum_{w \in d} \tau_{m(w)} n_{dw} p_{tdw}$ for all $t \in T$;
    $\theta_{td} := \underset{t \in T}{\operatorname{norm}}\left(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}\right)$ for all $t \in T$;
  until θ_d *converges*;
  increment $\tilde{n}_{wt}$ by $n_{dw} p_{tdw}$ for all $w \in d$ and $t \in T$;

Concurrency. An general rule of concurrency design is to express parallelism at the highest possible level. For this reason BigARTM implements a concurrent processing of the batches and keeps a single-threaded code for the ProcessBatch (D_b, ϕ_{wt}) routine.

To split collection into batches and process them concurrently is a common approach, introduced in AD-LDA algorithm [8], and then further developed in PLDA [15] and PLDA+ [7] algorithms. These algorithms require all concurrent workers to become idle before an update of the Φ matrix. Such synchronization step adds a large overhead in the online algorithm where Φ matrix is updated multiple times on each iteration. An alternative architecture without the synchronization step is described in [10], however it mostly targets a distributed cluster environment. In our work we develop an efficient single-node architecture where all workers benefit from the shared memory space.

To run multiple ProcessBatch in parallel the inputs and outputs of this routine are stored in two separate in-memory queues, locked for push and pop operations with spin locks (Fig. 1). This approach does not add any noticeable

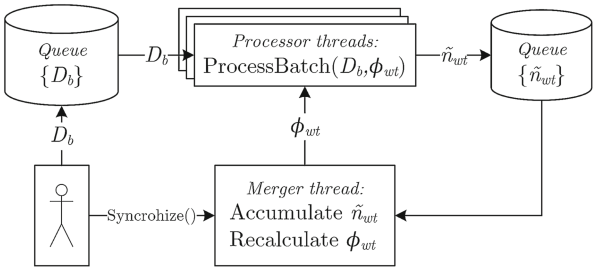

Fig. 1. Diagram of key BigARTM components

synchronization overhead because both queues only store smart pointers to the actual data objects, so push and pop operations does not involve copying or relocating big objects in the memory.

Smart pointers are also essential for lifecycle of the Φ matrix. This matrix is *read* by all processors threads, and can be *written* at any time by the merger thread. To update Φ without pausing all processor threads we keep two copies — an *active* Φ and a *background* Φ matrices. The active matrix is read-only, and is used by the processor threads. The background matrix is being built in a background by the merger thread at steps 6 and 7 of Algorithm 1, and once it is ready merger thread marks it as active. Before processing a new batch the processor thread gets the current active matrix from the merger thread. This object is passed via shared smart pointer to ensure that processor thread can keep ownership of its Φ matrix until the batch is fully processed. As a result, all processor threads keep running concurrently with the update of Φ matrix.

Note that all processor threads share the same Φ matrix, which means that memory usage stays at constant level regardless of how many cores are used for computation. Using memory for two copies of the Φ matrix in our opinion gives a reasonable usage balance between memory and CPU resources. An alternative solution with only one Φ matrix is also possible, but it would require a heavy usage of atomic CPU instructions. Such operations are very efficient, but still come at a considerable synchronization cost[1], and using them for all reads and writes of the Φ matrix would cause a significant performance degradation for merger and processor threads. Besides, an arbitrary overlap between reads and writes of the Φ matrix eliminates any possibility of producing a deterministic result. The design with two copies of the Φ matrix gives much more control over this and in certain cases allows BigARTM to behave in a fully deterministic way.

The design with two Φ matrices only supports a single merger thread, and we believe it should handle all $\tilde{n}_{wt}$ updates coming from many threads. This is a reasonable assumption because merging at step 6 takes only about $O(|W| \cdot |T|)$ operations to execute, while ProcessBatch takes $O(n|T|I)$ operations, where n is the number of non-zero entries in the batch, I is the average number of inner iterations in ProcessBatch routine. The ratio $n/|W|$ is typically from 100 to 1000

[1] http://stackoverflow.com/questions/2538070/atomic-operation-cost.

(based on datasets in UCI Bag-Of-Words repository), and I is $10 \ldots 20$, so the ratio safely exceeds the expected number of cores (up to 32 physical CPU cores in modern workstations, and even 60 cores of the Intel Xeon Phi co-processors).

Data Layout. BigARTM uses dense single-precision matrices to represent Φ and Θ. Together with the Φ matrix we store a global dictionary of all terms $w \in W$. This dictionary is implemented as std::unordered_map that maps a string representation of $w \in W$ into its integer index in the Φ matrix. This dictionary can be extended automatically as more and more batches came through the system. To achieve this each batch D_b contains a local dictionary W_b, listing all terms that occur in the batch. The n_{dw} elements of the batch are stored as a sparse CSR matrix (Compressed Sparse Raw format), where each row correspond to a document $d \in D_b$, and terms w run over a local batch dictionary W_b.

For performance reasons Φ matrix is stored in column-major order, and Θ in row-major order. This layout ensures that $\sum_t \Phi_{wt}\theta_{td}$ sum runs on contiguous memory blocks. In both matrices all values smaller than 10^{-16} are always replaced with zero to avoid performance issues with denormalized numbers[2].

Programming Interface. All functionality of BigARTM is expressed in a set of extern C methods. To input and output complex data structures the API uses Google Protocol Buffers[3]. This approach makes it easy to integrate BigARTM into any research or production environment, as almost every modern language has an implementation of Google Protocol Buffers and a way of calling extern C code (ctypes module for Python, loadlibrary for Matlab, PInvoke for C#, etc.).

On top of the extern C API BigARTM already has convenient wrappers in C++ and Python. We are also planning to implement a Java wrapper in the near future. In addition to the APIs the library also has a simple CLI interface.

BigARTM has built-in libraries of regularizers and quality measures that can be extended in current implementation only through project recompilation.

Basic Tools. A careful selection of the programming tools is important for any software project. This is especially true for BigARTM as its code is written in C++, a language that by itself offers less functionality comparing to Python, .NET Framework or Java. To mitigate this we use various parts of the Boost C++ Libraries, Google Protocol Buffers for data serialization, ZeroMQ library for network communication, and several other libraries.

BigARTM uses CMake as a cross-platform build system, and it successfully builds on Windows, Linux and OS X in 32 and 64 bit configurations. Building the library require a recent C++ compiler with C++11 support (GNU GCC 4.6.3, clang 3.4 or Visual Studio 2012 or newer), and Boost Libraries 1.46.1 or newer. All the other third-parties are included in BigARTM repository.

We also use free online services to store source code (https://github.com), to host online documentation (https://readthedocs.org) and to run automated continuous integration builds (http://travis-ci.org).

[2] http://en.wikipedia.org/wiki/Denormal_number#Performance_issues.
[3] http://code.google.com/p/protobuf/.

5 Experiments

In this section we evaluate the runtime performance and the algorithmic quality of BigARTM against two popular software packages — Gensim [14] and Vowpal Wabbit[4]. We also demonstrate some of the unique BigARTM features, such as combining regularizers and multi-language topic modeling via multimodality, which are not available in the other software packages.

All three libraries (VW.LDA, Gensim and BigARTM) work out-of-core, e. g. they are designed to process data that is too large to fit into a computer's main memory at one time. This allowed us to benchmark on a fairly large collection — 3.7 million articles from the English Wikipedia[5]. The conversion to bag-of-words was done with gensim.make_wikicorpus script[6], which excludes all non-article pages (such as category, file, template, user pages, etc.), and also pages that contain less than 50 words. The dictionary is formed by all words that occur in at least 20 documents, but no more than in 10 % documents in the collection. The resulting dictionary was caped at $|W| = 100\,000$ most frequent words.

Both Gensim and VW.LDA represents the resulting topic model as Dirichlet distribution over Φ and Θ matrices: $\boldsymbol{\theta}_d \sim \mathrm{Dir}(\boldsymbol{\gamma}_d)$ and $\boldsymbol{\phi}_t \sim \mathrm{Dir}(\boldsymbol{\lambda}_t)$. On contrary, BigARTM outputs a non-probabilistic matrices Φ and Θ. To compare the perplexity we take the mean or the mode of the posterior distributions:

$$\phi_{wt}^{\text{mean}} = \underset{w \in W}{\text{norm}}\, \lambda_{wt}, \qquad \theta_{td}^{\text{mean}} = \underset{t \in T}{\text{norm}}\, \gamma_{td};$$

$$\phi_{wt}^{\text{mode}} = \underset{w \in W}{\text{norm}}(\lambda_{wt} - 1), \qquad \theta_{td}^{\text{mode}} = \underset{t \in T}{\text{norm}}(\gamma_{td} - 1).$$

The perplexity measure is defined as

$$\mathscr{P}(D, p) = \exp\left(-\frac{1}{n} \sum_{d \in D} \sum_{w \in d} n_{dw} \ln p(w \mid d)\right). \tag{6}$$

Comparison to Existing Software Packages. The *Vowpal Wabbit (VW)* is a library of online algorithms that cover a wide range of machine learning problems. For topic modeling VW has the VW.LDA algorithm, based on the Online Variational Bayes LDA [5]. VW.LDA is neither multi-core nor distributed, but an effective single-threaded implementation in C++ made it one of the fastest tools for topic modeling.

The *Gensim* library specifically targets the area of topic modeling and matrix factorization. It has two LDA implementations — LdaModel and LdaMulticore, both based on the same algorithm as VW.LDA (Online Variational Bayes LDA [5]). Gensim is entirely written in Python. Its high performance is achieved through the usage of NumPy library, built over low-level BLAS libraries (such

[4] https://github.com/JohnLangford/vowpal_wabbit/.
[5] http://dumps.wikimedia.org/enwiki/20141208/.
[6] https://github.com/piskvorky/gensim/tree/develop/gensim/scripts/.

Table 1. The comparison of BigARTM with VW.LDA and Gensim. *Train time* is the time for model training, *inference* is the time for calculation of θ_d of 100 000 held-out documents, *perplexity* is calculated according to (6) on held-out documents.

Library	Procs	Train Time	Inference Time	Perplexity Mode	Perplexity Mean
BigARTM	1	35 min	72 s	4000	
LdaModel	1	369 min	395 s	4213	4161
VW.LDA	1	73 min	120 s	4061	4108
BigARTM	4	9 min	20 s	4061	
LdaMulticore	4	60 min	222 s	4055	4111
BigARTM	8	4.5 min	14 s	4304	
LdaMulticore	8	57 min	224 s	4379	4455

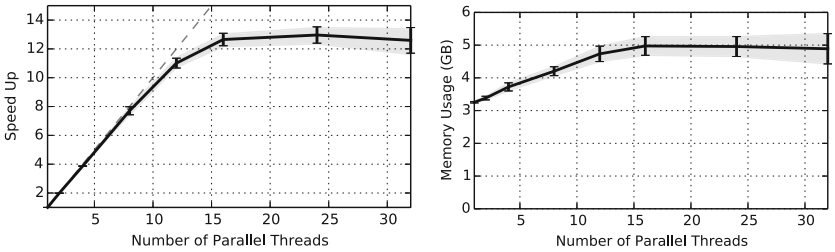

Fig. 2. Running BigARTM in parallel: speed up (left) and memory usage (right)

as Intel MKL, ATLAS, or OpenBLAS). In LdaModel all batches are processed sequentially, and the concurrency happens entirely within NumPy. In LdaMulticore the workflow is similar to BigARTM — several batches are processed concurrently, and there is a single aggregation thread that asynchronously merges the results.

Each run in our experiment performs one pass over the Wikipedia corpus and produces a model with $|T| = 100$ topics. The runtime is reported for an Intel-based CPU with 16 physical cores with hyper-threading. The collection was split into batches with 10000 documents each (`chunksize` in Gensim, `minibatch` in VW.LDA). The update rule in online algorithm used $\rho = (b + \tau_0)^{-0.5}$, where b is the number of batches processed so far, and τ_0 is an a constant offset parameter introduced in [5], in our experiment $\tau_0 = 64$. Updates were performed after each batch in non-parallel runs, and after P batches when running in P threads. LDA priors were fixed as $\alpha = 0.1,\ \beta = 0.1$, so that $\boldsymbol{\theta}_d \sim \mathrm{Dir}(\alpha),\ \boldsymbol{\phi}_t \sim \mathrm{Dir}(\beta)$.

Table 1 compares the performance of VW.LDA, Gensim LdaModel, Gensim LdaMulticore, and BigARTM. Figure 2 shows BigARTM speedup and memory consumption depending on the number of CPU threads for Amazon AWS c3.8xlarge with 32 virtual cores, Gensim 0.10.3 under Python 2.7.

Table 2. Comparison of LDA and ARTM models. Quality measures: $\mathcal{P}_{10\,k}$, $\mathcal{P}_{100\,k}$ — hold-out perplexity on 10 K and 100 K documents sets, $\mathcal{S}_\Phi$, $\mathcal{S}_\Theta$ — sparsity of Φ and Θ matrices (in %), $\mathcal{K}_s$, $\mathcal{K}_p$, $\mathcal{K}_c$ — average topic kernel size, purity and contrast respectively.

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

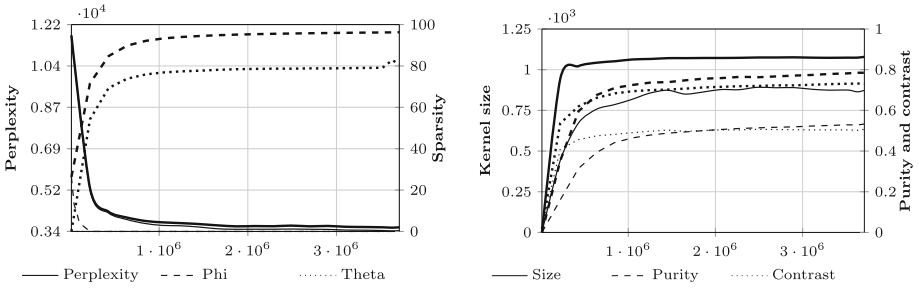

Fig. 3. Comparison of LDA (thin) and ARTM (bold) models. The number of processed documents is shown along the X axis.

Experiments with Combination of Regularizers. BigARTM has a built-in library of regularizers, which can be used in any combination. In the following experiment we combine three regularizers: sparsing of ϕ_t distributions, sparsing of θ_d distributions, and pairwise decorrelation of ϕ_t distributions. This combination helps to improve several quality measures without significant loss of perplexity, according to experiments on the offline implementation of ARTM [13]. The goal of our experiment is to show that this remains true for the online implementation in BigARTM. We use the following built-in quality measures: the hold-out perplexity, the sparsity of Φ and Θ matrices, and the characteristics of topic lexical kernels (size, purity, and contrast) averaged across all topics.

Table 2 compares the results of additive combination of regularizers (ARTM) and the usual LDA model. Figure 3 presents quality measures as functions of the number of processed documents. The left chart shows perplexity and sparsity of Φ, Θ matrices, and the right chart shows average lexical kernel measures.

Experiments on Multi-language Wikipedia. To show how BigARTM works with multimodal datasets we prepared a text corpus containing all English and Russian Wikipedia articles with mutual interlanguage links. We represent each linked pair of articles as a single multi-language document with two modalities, one modality for each language. That is how our multi-language collection acts as a multimodal document collection.

The dump of Russian articles[7] had been processed following the same technique as we previously used in experiments on English Wikipedia. Russian words

[7] http://dumps.wikimedia.org/ruwiki/20141203/.

Table 3. Top 10 words with $p(w \mid t)$ probabilities (in %) from two-language topic model, based on Russian and English Wikipedia articles with mutual interlanguage links.

Topic 68				**Topic 79**			
research	4.56	институт	6.03	goals	4.48	матч	6.02
technology	3.14	университет	3.35	league	3.99	игрок	5.56
engineering	2.63	программа	3.17	club	3.76	сборная	4.51
institute	2.37	учебный	2.75	season	3.49	фк	3.25
science	1.97	технический	2.70	scored	2.72	против	3.20
program	1.60	технология	2.30	cup	2.57	клуб	3.14
education	1.44	научный	1.76	goal	2.48	футболист	2.67
campus	1.43	исследование	1.67	apps	1.74	гол	2.65
management	1.38	наука	1.64	debut	1.69	забивать	2.53
programs	1.36	образование	1.47	match	1.67	команда	2.14
Topic 88				**Topic 251**			
opera	7.36	опера	7.82	windows	8.00	windows	6.05
conductor	1.69	оперный	3.13	microsoft	4.03	microsoft	3.76
orchestra	1.14	дирижер	2.82	server	2.93	версия	1.86
wagner	0.97	певец	1.65	software	1.38	приложение	1.86
soprano	0.78	певица	1.51	user	1.03	сервер	1.63
performance	0.78	театр	1.14	security	0.92	server	1.54
mozart	0.74	партия	1.05	mitchell	0.82	программный	1.08
sang	0.70	сопрано	0.97	oracle	0.82	пользователь	1.04
singing	0.69	вагнер	0.90	enterprise	0.78	обеспечение	1.02
operas	0.68	оркестр	0.82	users	0.78	система	0.96

were lemmatized with Yandex MyStem 3.0[8]. To further reduce the dictionary we only keep words that appear in no less than 20 documents, but no more than in 10 % of documents in the collection. The resulting collection contains 216175 pairs of Russian–English articles, with combined dictionary of 196749 words (43 % Russian, 57 % English words).

We build multi-language model with 400 topics. They cover a wide range of themes such as science, architecture, history, culture, technologies, army, different countries. All 400 topics were reviewed by an independent assessor, and he successfully interpreted all except four topics.

Table 3 shows top 10 words for four randomly selected topics. Top words in these topics are clearly consistent between Russian and English languages. The Russian part of last topic contains some English words such as “Windows” or “Server” because it is common to use them in Russian texts without translation.

6 Conclusions

BigARTM in an open source project for parallel online topic modeling of large text collections. It provides a high flexibility for various applications due to

[8] https://tech.yandex.ru/mystem/.

multimodality and additive combinations of regularizers. BigARTM architecture has a rich potential. Current components can be reused in a distributed solution that runs on cluster. Further improvement of single-node can be achieved by offloading batch processing into GPU.

Acknowledgements. The work was supported by the Russian Foundation for Basic Research grants 14-07-00847, 14-07-00908, 14-07-31176 and by Skolkovo Institute of Science and Technology (project 081-R).

References

1. Blei, D.M.: Probabilistic topic models. Commun. ACM **55**(4), 77–84 (2012)
2. Blei, D.M., Jordan, M.I.: Modeling annotated data. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, pp. 127–134. ACM, New York (2003)
3. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
4. Daud, A., Li, J., Zhou, L., Muhammad, F.: Knowledge discovery through directed probabilistic topic models: a survey. Front. Comput. Sci. China **4**(2), 280–301 (2010)
5. Hoffman, M.D., Blei, D.M., Bach, F.R.: Online learning for latent dirichlet allocation. In: NIPS, pp. 856–864. Curran Associates Inc. (2010)
6. Hofmann, T.: Probabilistic latent semantic indexing. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 50–57. ACM, New York (1999)
7. Liu, Z., Zhang, Y., Chang, E.Y., Sun, M.: PLDA+: parallel latent Dirichlet allocation with data placement and pipeline processing. ACM Trans. Intell. Syst. Technol. **2**(3), 26:1–26:18 (2011)
8. Newman, D., Asuncion, A., Smyth, P., Welling, M.: Distributed algorithms for topic models. J. Mach. Learn. Res. **10**, 1801–1828 (2009)
9. Rubin, T.N., Chambers, A., Smyth, P., Steyvers, M.: Statistical topic models for multi-label document classification. Mach. Learn. **88**(1–2), 157–208 (2012)
10. Smola, A., Narayanamurthy, S.: An architecture for parallel topic models. Proc. VLDB Endow. **3**(1–2), 703–710 (2010)
11. Vorontsov, K.V.: Additive regularization for topic models of text collections. Dokl. Math. **89**(3), 301–304 (2014)
12. Vorontsov, K.V., Potapenko, A.A.: Additive regularization of topic models. Mach. Learn. **101**(1–3), 303–323 (2015)
13. Vorontsov, K., Potapenko, A.: Tutorial on probabilistic topic modeling: additive regularization for stochastic matrix factorization. In: Ignatov, D.I., Khachay, M.Y., Panchenko, A., Konstantinova, N., Yavorsky, R.E. (eds.) AIST 2014. CCIS, vol. 436, pp. 29–46. Springer, Heidelberg (2014)
14. Řehůřek, R., Sojka, P.: Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, ELRA, Valletta, pp. 45–50, May 2010
15. Wang, Y., Bai, H., Stanton, M., Chen, W.-Y., Chang, E.Y.: PLDA: parallel latent dirichlet allocation for large-scale applications. In: Goldberg, A.V., Zhou, Y. (eds.) AAIM 2009. LNCS, vol. 5564, pp. 301–314. Springer, Heidelberg (2009)