Learning Topic Models with Arbitrary Loss

Murat Apishev
great-mel@yandex.ru

Konstantin Vorontsov
k.v.vorontsov@phystech.edu

Lomonosov Moscow State University
Moscow Institute of Physics and Technology

April, 2020

Topic Modeling

Topic Modeling — an application of machine learning to statistical text analysis.

Topic — a specific terminology of the subject area, the set of terms (unigrams or *n*-grams) frequently appearing together in documents.

Topic model uncovers latent semantic structure of a text collection:

- topic t is a probability distribution $p(w|t)$ over terms w;
- document d is a probability distribution $p(t|d)$ over topics t.

Applications — information retrieval for long-text queries, classification, categorization, summarization of texts.

Topic Modeling Task

Given: W — set (vocabulary) of terms (unigrams or $n-$grams),
D — set (collection) of text documents $d \subset W$,
n_{dw} — how many times term w appears in document d.

Find: model $p(w|d) = \sum\limits_{t \in T} \phi_{wt}\theta_{td}$ with parameters $\underset{W\times T}{\Phi}$ и $\underset{T\times D}{\Theta}$:
$\phi_{wt} = p(w|t)$ — term probabilities w in each topic t,
$\theta_{td} = p(t|d)$ — topic probabilities t in each document d.

Criteria log-likelihood maximization:

$$\sum_{d\in D}\sum_{w\in d} n_{dw} \ln \sum_{t\in T} \phi_{wt}\theta_{td} \;\to\; \max_{\phi,\theta};$$

$$\phi_{wt} \geqslant 0; \quad \textstyle\sum_w \phi_{wt} = 1; \qquad \theta_{td} \geqslant 0; \quad \textstyle\sum_t \theta_{td} = 1.$$

PLSA and EM-algorithm

Log-likelihood maximization:

$$\sum_{d\in D}\sum_{w\in W} n_{dw} \ln \sum_{t} \phi_{wt}\theta_{td} \;\to\; \max_{\Phi,\Theta};$$

EM-algorithm: the simple iteration method for the set of equations

$$\begin{array}{r} \text{E-шаг:} \\ \\ \text{M-шаг:} \\ \\ \end{array} \begin{cases} p_{tdw} = \operatorname*{norm}_{t\in T}\bigl(\phi_{wt}\theta_{td}\bigr); \\ \phi_{wt} = \operatorname*{norm}_{w\in W}\bigl(n_{wt}\bigr), \quad n_{wt} = \sum\limits_{d\in D} n_{dw}\, p_{tdw}; \\ \theta_{td} = \operatorname*{norm}_{t\in T}\bigl(n_{td}\bigr), \quad n_{td} = \sum\limits_{w\in d} n_{dw}\, p_{tdw}; \end{cases}$$

where $\operatorname*{norm}_{i\in I} x_i = \frac{\max\{x_i,0\}}{\sum\limits_{j\in I}\max\{x_j,0\}}$.

EM-algorithm for ARTM

Log-likelihood maximization with additive regularization criterion R:

$$\sum_{d\in D}\sum_{w\in W} n_{dw}\ln\sum_t \phi_{wt}\theta_{td} + R(\Phi,\Theta) \;\to\; \max_{\Phi,\Theta};$$

EM-algorithm: the simple iteration method for the set of equations

$$\begin{array}{rl}
\text{E-step:} & \begin{cases} p_{tdw} = \operatorname*{norm}_{t\in T}\bigl(\phi_{wt}\theta_{td}\bigr); \\[2ex]
\phi_{wt} = \operatorname*{norm}_{w\in W}\Bigl(n_{wt} + \phi_{wt}\frac{\partial R}{\partial \phi_{wt}}\Bigr), \quad n_{wt} = \sum_{d\in D} n_{dw}p_{tdw}; \\[2ex]
\theta_{td} = \operatorname*{norm}_{t\in T}\Bigl(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}\Bigr), \quad n_{td} = \sum_{w\in d} n_{dw}p_{tdw}. \end{cases} \\
\text{M-step:} &
\end{array}$$

Phi sparsification and decorrelation

Two examples of regularizers:

- LDA-style smoothing/sparsifying Φ with given positive/negative values β_{wt}:

$$R(\Phi) = \sum_{t \in T} \sum_{w \in W} \beta_{wt} \ln \phi_{wt};$$

- Topic decorrelation, that makes topics as diverse as possible:

$$R(\Phi) = -\frac{\tau}{2} \sum_{t \in T} \sum_{s \in T \setminus t} \sum_{w \in W} \phi_{wt} \phi_{ws}.$$

EM-algorithm with arbitrary loss

Replace the logarithm in the standard log-likelihood loss $\ln p(w|d)$ with a smooth function ℓ:

$$\sum_{d\in D}\sum_{w\in d} n_{dw}\,\ell\Bigl(\sum_{t\in T}\phi_{wt}\theta_{td}\Bigr) + R(\Phi,\Theta) \;\to\; \max_{\Phi,\Theta}; \tag{1}$$

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

EM-algorithm with arbitrary loss

Theorem

The local maximum (Φ, Θ) of the optimization problem (1), (2) with differentiable loss ℓ and differentiable regularizer R satisfies the system of M-step equations

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

$$\theta_{td} = \operatorname*{norm}_{t\in T}\left(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}\right); \quad n_{td} = \sum_{w\in d} n_{dw} p_{tdw};$$

and the E-step equation

$$p_{tdw} = \phi_{wt}\theta_{td}\ell'\left(\sum_{t\in T}\phi_{wt}\theta_{td}\right).$$

Special case: fast E-steps

The simplest function $\ell(p) = p$ gives a new optimization problem:

$$\sum_{d \in D} n_d \langle \hat{p}(w|d), p(w|d) \rangle + R(\Phi, \Theta) \;\to\; \max_{\Phi,\Theta};$$

In this case E-step equation is computed without normalization

$$p_{tdw} = \phi_{wt} \theta_{td}.$$

As such modification allows a significant speed-up, we call it a *fast E-step*.

BigARTM library

Features:

- Fast parallel and online processing of Big Data;
- Multimodal and regularized topic modeling;
- Built-in library of regularizers and quality measures.

Community:

- Open-source https://github.com/bigartm;
- Documentation http://bigartm.org.

License and programming environment:

- Freely available for commercial usage (BSD 3-Clause license);
- Cross-platform — Windows, Linux, Mac OS X (32 bit, 64 bit);
- Programming APIs: command line, C++, Python.

Offline and online EM-algorithms

Offline algorithm:

- performs several passes through collection;
- while processing E-step for each document collects n_{wt} counters;
- at the end of each pass proceeds M-step: uses final n_{wt} values to construct new Φ.

Online algorithm:

- performs one pass through collection;
- collects n_{wt} counters for a batch of documents;
- performs M-step after processing of a given number of batches;
- processes next generation of batches with new version of Φ.

Asynchronous online algorithm:

- performs M-step for results of old batches generation processing in parallel with E-step for new generation;
- achieves better likelihood than offline or synchronous online do in a given time interval.

BigARTM optimization for sparse models

BigARTM generally stores four types of big matrices:

- document-topic matrix Θ;
- topic-term matrix Φ;
- topic-term counters matrix n_{wt};
- topic-term regularization amendments matrix r_{wt}.

Usually we don't store Θ matrix, as it has $O(|D|)$ size.

Other three matrices are stored in the same structure called **Φ-like** matrix.

BigARTM optimization for sparse models

Current state:

- BigARTM uses dense real-valued matrices to store Φ-like matrices;
- for each term w the corresponding values are located in memory as a single continuous block;
- the memory can be accessed in a locally sequential way while processing loops over a set of topics;
- **BUT:** in case of sparse model calculations for most of the elements will be wasted as they are zero.

BigARTM optimization for sparse models

Proposed hybrid format:

- each line (an array S of length m) can be stored in one of two forms, either sparse or dense;
- form depends on the number k of non-zero elements in S;
- in **dense** form S is stored as before (continuous memory block);
- in **sparse** form it is stored in three arrays:
  - V — real-valued array with all k non-zero elements of S in the original order;
  - I — integer array, stores for each element of V its index in the original array;
  - M — bitmap with length m ($M[i] == 1$ if $S[i] > 0$).

BigARTM optimization for sparse models

Proposed hybrid format:

- in **sparse** form it is stored in three arrays:
  - V — real-valued array with all k non-zero elements of S in the original order;
  - I — integer array, stores for each element of V its index in the original array;
  - M — bitmap with length m ($M[i]$ is 1 if $S[i] > 0$).

We need:

- V to store non-zero elements;
- M to organize effective ($O(1)$) random access to zero elements of S;
- I to proceed a loop over non-zero elements of S and to allow $O(\log(k))$ access to them.

Dataset and quality measures

Dataset: subset of English Wikipedia:

- 200K documents for offline algorithm;
- 1M documents for online algorithms;
- 100K terms in dictionary in both cases.

Quality measures:

- *Perplexity* is an inverse of the likelihood of data, (the smaller — the better):

$$\mathcal{P}(\Phi, \Theta) = \exp\left(-\frac{1}{n}\sum_{d \in D}\sum_{w \in d} n_{dw} \ln p(w|d)\right);$$

- *Coherence of a topic* t is the average PPMI over term pairs:

$$\mathcal{C}_t(\Phi) = \frac{2}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} \mathrm{PPMI}(w_i, w_j).$$

  Average coherence over topics is a good interpretability measure (the greater — the better).

Experiment 1

Check the benefits of BigARTM optimization for sparse models

Compare three models:

Features / model type	1	2	3
Enabled optimization	No	No	Yes
Uniform sparsification	No	Yes	Yes

Quality measures:

- training time;
- peak memory consumption.

Experimental parameters:

- model type;
- number of topics (100/500/1000/2000);
- number of document passes during one collection pass (1/5/10/15).

Experiment 1: results

- **Offline algorithm**
  - No improvements from optimization for 100 topics or 1 document pass;
  - In other cases model 3 is faster than 1 (up to 30% acceleration);
  - Model 1 has lowest memory consumption.
- **Online algorithm**
  - No improvements from optimization for 100 topics or 1 document pass;
  - In other cases model 3 is faster than 1 (up to 30% acceleration);
  - Models 1 and 3 have the same memory consumption.
- **Asynchronous online algorithm**
  - In all cases model 3 is faster than 1 (up to 30% acceleration);
  - In all cases model 3 consumes less memory than 1 (up to 23% economy).

Experiment 2

Check the benefits of combining normal and fast E-steps while training one model.

Training strategies to compare:

- **FULL:** all iterations are normal;
- **NONE:** all iterations are fast;
- **MIXED:** fast and normal iterations alternate;
- **HALF:** the first half of the iterations is fast, the second is the usual one;
- **LAST:** 80% of the first iterations are fast, the rest are the usual ones;
- **SPARSE:** FULL model with uniform sparsification;
- **DECOR:** FULL model with topic decorrelation.

Iteration for offline algorithm is a collection pass, for online — document pass.

Experiment 2

Quality measures:

- training time;
- perplexity;
- average coherence.

Experimental parameters:

- training strategy;
- number of topics (100/500/1000/2000);
- model updates frequency for online algorithms in number of processed batches (32/24/16/8).

Additional measuring for offline:
perplexity and coherence values on minimal time across all strategies.

Experiment 2: offline algorithm results

- DECOR strategy
  - is the best choice in case of model with 100 topics;
  - fails for larger models due to computation complexity.
- NONE strategy is the fastest one, and also improves coherence, but it spoils perplexity very much;
- HALF strategy is an optimal choice both in case of
  - fixed number of iterations;
  - score values on minimal time;

  anyway it saves or even decreases final perplexity value and achieves up to 10% coherence growth compared to base FULL case.

Experiment 2: online algorithm results

- DECOR strategy
  - is the best solution for a model with 100 topics;
  - fails for larger models.
- NONE/HALF/MIXED/LAST strategies with the same frequency of model updates
  - reduce train time by 25-50%;
  - spoil both perplexity and coherence significantly.
- HALF with more frequent updates allows
  - a big (up to 23%) improvement of the perplexity;
  - to achieve same time and coherence values or their minor decay.

Experiment 2: asynchronous online algorithm results

- DECOR strategy
  - is the best solution for a model with 100 and 500 topics;
  - too slow for models with 1000 and 2000 topics.
- NONE/HALF/MIXED/LAST strategies with the same frequency of model updates
  - reduce train time significantly (up to 2 times);
  - spoil both perplexity and coherence significantly.
- HALF strategy with more frequent model updates is a best strategy for models with 1000 and 2000 topics, as it
  - allows to get more than twice the gain in perplexity;
  - gives coherence losses within 10%;
  - in all cases remains faster than the basic algorithm by 10-30%.
- SPARSE strategy in some cases
  - allows to obtain comparable results with HALF strategy;
  - works even faster than HALF due to optimization for sparse models.

Conclusions

- We generalized the EM-algorithm for any differentiable loss function in an optimized functional when training topic models;
- We found experimentally the superior strategy for combining normal and fast E-steps;
- We proposed efficient optimization for sparse models;
- We discovered some new properties of the topic decorrelation.
- The future work includes:
  - study of regularizers and mixing E-step strategies combinations;
  - studying loss functions, other than linear and logarithmic ones.