Topic Modeling Library and Framework Specification

Valeriy Avanesov, Ilya Kozlov

February 2014

1 Terminology

- Textual document – Textual document is a map attribute $\to$ sequence of words.
- Document – Document is a map from attribute to sequence of integer numbers.
- Topic – multinomial distribution on words.

2 Introduction

2.1 Generative model

In topic modeling every document viewed as mixture of topics. Each topic is a multinomial distribution on words, so generation model may be defined as follow:

- For every position in document d i.i.d choose topic t from distribution of topics by document
- Choose word w from topic t

The aim of topic modeling is to recover topics and distribution of document by topics.

2.1.1 Polylingual topic model

Suppose one has a collection of documents on different languages. We have a prior knowledge that some of the documents (or all of them) are written on the same topic but in different languages (for example one document may be a translation of another). Wikipedia may be considered as a source of this type of data. It leads us to a polylingual topic modeling [1]. In this model we assume that every topic is a set of multinomial distributions, one per language. Also we assume that every document may hold more than one set of words, so we represent document as map attribute $\to$ set of words.

2.1.2 Robust PLSA

In robust PLSA we assume that some of words too specific for document and may not be explained by topic distribution. Conversely, some of words too common and may be explained by any topic. Robust PLSA take it into account. In robust PLSA word may be generated from topic, noise or background. Noise is a multinomial distribution on rare words. Every document has its own noise. Background is a multinomial distribution on common words. Background is one for all documents. To generate words w in document d we:

- with probability γ generate word from noise
- with probability ε generate word from background
- with probability $1 - \gamma - \varepsilon$ generate word from topic distribution as in 2.1

γ and ε are hyperparameter of model.

2.1.3 Sparse PLSA

One document often corresponds to only a few topics, not to every. Analogously, word may corresponds only to a few topics, not to every topic. Sparse PLSA takes it into account, and replace some of topics weights in document by zero and replace some weights in distribution of words by topic by zero. Sparsification of topic modeling has some features, which allow to sparse distribution of words by topic and distribution of document by topics without decreasing of model quality.

2.2 Topic modeling as optimization problem

According to generative model one can estimate probability to observe collection D as:

$$p(D) = \prod_{d \in D} \prod_{w \in d} \sum_{t} p(t|d) p(w|t) \tag{1}$$

Denote $\varphi_{wt} = p(w|t)$ and $\theta_{td} = p(t|d)$. One may obtain φ_{wt} and θ_{td} as solution of optimization problem

$$L = \sum_{d \in D} \sum_{w \in d} \log \sum_{t} \varphi_{wt} \theta_{td} \to \max \tag{2}$$

with boundary

$$\forall t \quad \sum_{w} \varphi_{wt} = 1, \quad \forall d \quad \sum_{w} \theta_{td} = 1 \tag{3}$$

and

$$\forall t, w \quad \varphi_{wt} \geq 0, \quad \forall d, t \quad \theta_{wt} \geq 0 \tag{4}$$

2.3 Topic modeling as matrix decomposition

2.3.1 Kullback-Leibler divergence

Kullback-Leibler divergence is a non-negative measure of difference between two different probability distribution:

$$KL(p_i||q_i) = \sum_{i=1}^{n} p_i \ln\left(\frac{p_i}{q_i}\right) \tag{5}$$

Consider an empirical distribution $\hat{p}_i$ and some parametric distribution $q_i = q_i(\alpha)$ which is used to explain $\hat{p}_i$. Easy to see that in this case minimization of KL–divergence is equivalent to estimation of α by maximum-likelihood:

$$KL(p_i||q_i(\alpha)) = \sum_{i=1}^{n} p_i \ln\left(\frac{p_i}{q_i(\alpha)}\right) \to \min_{\alpha} \Leftrightarrow \sum_{i=1}^{n} p_i \ln(q_i(\alpha)) \to \max_{\alpha} \tag{6}$$

Thus one can easily see that (2) equivalent to weighted Kullback-Leibler divergence minimization:

$$\sum_{d\in D} n_d KL_w\left(\frac{n_{dw}}{n_d}||\sum_{t\in T}\varphi_{wt}\theta_{td}\right) \to \min_{\Phi,\Theta} \tag{7}$$

where n_{wd}– number of words w in document d, n_d – number of words in document d.

2.3.2 Matrix decomposition

Denote empirical distribution of words by document as $\hat{p}(w,d) = \frac{n_{wd}}{n_d}$. According to this notation one can consider the problem (2) as matrix decomposition:

$$F \approx_{KL} \Phi\Theta \tag{8}$$

where matrix $F = (\hat{p}(w,d))_{W\times D}$ is empirical distribution of words by document, matrix $\Phi = (\varphi_{wt})_{W\times D}$ is distribution of words by topics and matrix $\Theta = (\theta_{td})_{T\times D}$ is distribution of documents by topics. Thus, our optimization problem may be rewritten in Kullback–Leibler notation as

$$KL(F, \Phi\Theta) \to \min \tag{9}$$

Thus PLSA may be observed as stochastic matrix decomposition.

2.4 Expectation-Maximization algorithm

Unfortunately (2) has no analytical solution. Thus we use Expectation - Maximization (EM) algorithm. This algorithm consists of two steps:

1. Estimation of number n_{dwt} of words w, produced by topic t in document d. (E - step)

2. Optimization of distribution of documents by topics and optimization of distribution of topics by words relying on the n_{dwt} values obtained during E - step . (M - step)

One can estimate n_{dwt} as follows:

$$n_{dwt} = n_{wd} p(w|t) p(t|d) \tag{10}$$

where n_{wd} – number of words w in document d. Thus, probability $p(w|t)$ may be estimated as

$$p(w|t) = \frac{n_{wt}}{n_t} = \frac{\sum_d n_{dwt}}{\sum_w \sum_d n_{dwt}} \tag{11}$$

Similarly for $p(t|d)$

2.5 Regularizers

Regularizers may improve human-understandability of the topics, transform PLSA to LDA, provide an ability for semi-supervised learning (employ a prior knowledge about document topic or topics structure). Instead of optimization (2) we optimize

$$L(\Phi, \Theta) + R(\Phi, \Theta) \to \max_{\Phi, \Theta} \tag{12}$$

Where $R(\Phi, \Theta)$ is a twice differentiable function, named regularizer. Solution of this problem leads to a modification of M-step:

$$\varphi_{wt} \propto \left(\hat{n}_{wt} + \varphi_{wt} \frac{\partial R(\Phi, \Theta)}{\partial \varphi_{wt}} \right)_+, \quad \theta_{td} \propto \left(\hat{n}_{dt} + \theta_{td} \frac{\partial R(\Phi, \Theta)}{\partial \varphi_{td}} \right)_+ \tag{13}$$

Where $\hat{n}$ has been estimated by E-step.

3 Implementation

We take a sequence of documents into input and return distributions of documents by topics and words by topics. We can divide our problem into the following parts:

1. Replace every word by its ordered number. For this purpose we use a class Numerator. Then we store a map from words to its ordered number in class Alphabet.

2. Choose initial approximation for matrices Φ and Θ. For this purpose we use class InitialApproxiomationGeneration.

3. Make E-step for every attribute in every documents. For this purpose we use class PLSABrick. A single brick processes a single attribute. Brick may be robust or non robust or whatever user implements it.

4. Then we have to perform M-step for matrix Φ (for every attribute) and Θ To perform M-step for matrix Φ we use PLSABrick and we in order to perform M-step for matrix Θ we use class PLSA.

5. After every iteration we check the stop criterion. We use class StoppingCriteria for this purpose.

6. In each step we use class Sparcifier in order to make our model sparse.

3.1 Sparcifier

We implement standard sparsifier, which replace weight by zero if it less then *threshold*, sparsification starts from iteration *startIteration* and it does not replace more than $maxNumberOfZeroised$ numbers. *threshold*, *startIteration* and $maxNumberOfZeroised$ a parameters.

3.2 Regularizer

Every regularizer take into input matrices Φ and Θ We implement the following regularizers: ZeroRegularizer (do nothing), AnticorrelationRegularizer (make topics more different), DirechletRegularizer (transforme PLSA into LDA).
Regularizer take into input distribution of words by topics (matrix Φ) and distribution of documents by topics (matrix Θ). Regularizer may calculate derivative $\frac{\partial R}{\partial \varphi_{wt}}$ and $\frac{\partial R}{\partial \theta_{wt}}$

4 "Quick" start

4.1 Creating a documents

Text normalization is not goal of this project. So we suppose that we have sequence of sequence of words (word is a string). First of all we have to create sequence of textual documents. To create TextualDocument type
`new TextualDocument(Map("lang" -> Seq[String]))`
Then use Numerator
`Numerator(Seq[TextualDocument])`$(Doc_0, Doc_1, Doc_2, \dots Doc_n)$
This method replace words by its serial number and group words. It also return Alphabet to convert serial number to word.

4.2 Creation of PLSA

PLSA consist of 4 main parts:

1. PLSABrick one per attribute.

2. StopingCriteria

3. InitiaApproximationGenerator

4. ThetaSparsifier

Construction of PLSABrick is describes in corresponding subsection. One of commonly used stopping criteria is a MaxNumberOfIterationsStopingCriteria (performs fixed number of iteration and terminate)
`val stoppingCriteria = new MaxNumberOfIterationsStopingCriteria(numberOfIterations)`

If you don't wont use sparsifier you may use IdenticalSparsifier (it do nothing)
`val sparsifier = new IdenticalSparsifier()`
Initial approximation generator generate Initial approximation for matrices Φ and Θ. One of a commonly used method is to draw every position in matrix from uniform distribution and make normalization. For this purposes one may use RandomInitialApproximationGenerator
`val initialApproximationGenerator = new RandomInitialApproximationGenerator()`

Now we may construct PLSA:
`val plsa = new PLSA(brick, sparsifier, initialApproximationGenerator, stoppingCriteria)`

4.2.1 Creation of PLSABrick

PLSABrick consist of three main parts:

1. attributeType: type of attribute to process
2. phiSparsifier
3. regularizer

Thus, before construct the PLSABrick we have to construct phiSparsifier and regularizer. If you don't wont use sparsifier you may use IdenticalSparsifier (it do nothing)
`val sparsifier = new IdenticalSparsifier()`
If you wont to use PLSA without regularizer you may use ZeroRegularizer (it do nothing). To convert PLSA to LDA you may use DirichletRegularizer.
`val regularizer = new ZeroRegularizer()` for pure PLSA
`val regularizer = new DirichletRegularizer(`α`, `β`)` for LDA. α and β is parameters in dirichlet distribution

Now we may construct PLSABrick:
`val brick = new PLSABrick("lang", sparsifier, regularizer)`

4.3 Using of PLSA

Now one may use PLSA for collection of documents:
`val trainedModel = plsa.trainOnDocuments(documents)`

Training model hold distribution of documents by topic (matrix Θ), distribution of words by topics (matrix Φ). Training model can also calculate distribution of new document by topics with fixed distribution of words by topic (matrix Φ).

References

[1] David Mimno, Hanna M. Wallach, Jason Naradowsky, David A. Smith, and Andrew McCallum. Polylingual topic models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 880–889, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.