Topic Modeling Library and Framework User Guide

Avanesov Valeriy
Kozlov Ilya

August 20, 2014

1 Input

The aim of this project is to provide flexible and simple library and framework for topic modeling. The main properties of this project:

- Ability to apply user's defined regularizer, sparsifier and stopping criteria.
- Ability to use multilingual topic modeling.

The paper is organized as follow: section 2 contain a quick introduction on topic modeling, 3 describe how to start project, without description of details of implementation. These two sections should allow you to use model LDA or PLSA in a simple use-causes. The fallowing section describes a details of implementation and it necessary for deeper understanding of project structure and implementation of your own regularizer, sparsifier and other stuff. Section 4 describe structure of project, section 5 describe regularizers and explain how to implement you own regularizer, section 6 describe what sparsifier is and how to implement you own sparsifier. Section 7 describe how to define that we do enough iteration of EM-algorithm. Section 8 describe how to use model in case of a multilingual texts.

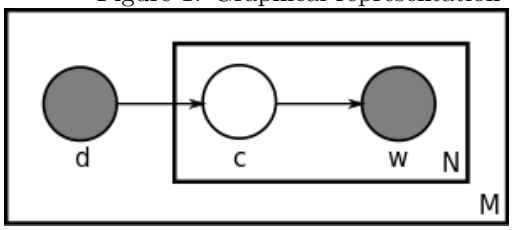
2 Introduction to the topic modeling

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

Figure 1: Graphical representation of PLSA generative process



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 [**?**]. 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 $\rightarrow$ 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.

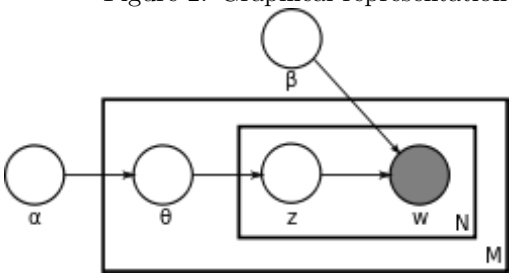
2.1.3 Sparse PLSA

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.

2.1.4 LDA

LDA is an extension of PLSA, LDA was described in [**?**] and now it is one of the most popular (may be the most popular) topic model. The idea of this model is that distribution of documents by topics and distribution of words by topic has some symmetric dirichlet[1] prior distribution and we maximize posterior distribution instead of finding a maximum likelihood solution.

Figure 2: Graphical representation of LDA generative process



α and β is hyperparameters of model.

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

[1] `http://en.wikipedia.org/wiki/Dirichlet_distribution`

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

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:

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

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

2.5.1 Probability interpretation

Does regularizer have probabilistic interpretation? Yes, it has. Imagine that Φ and Θ have some prior distribution, for example dirichlet distribution. In this case:

$$P(D, \Theta, \Phi) = P(D|\Theta, \Phi) \times P(\Theta, \Phi) \tag{14}$$

Logarithm equation 14:

$$\ln P(D, \Theta, \Phi) = \ln(P(D|\Theta, \Phi) \times P(\Theta, \Phi) = \ln(P(D|\Theta, \Phi)) + \ln(P(\Theta, \Phi)) \tag{15}$$

Denote $L(\Phi, \Theta) = \ln(P(D|\Theta, \Phi))$ and $R(\Phi, \Theta) = \ln(P(\Theta, \Phi))$ and obtain 12

Example Consider the example with symmetric dirichlet distribution[2] with parameter α. Assume we estimate n_{td} for each topic in E-step and want to perform M- step. We would find new distribution of document by topic as a maximum posterior probability solution:

$$\ln(P(\vec{\theta}|\vec{n}_{dt})) \to \max \tag{16}$$

$$\ln(P(\vec{\theta}|\vec{n}_{dt})) \propto \ln(P(\vec{n}_{dt}|\vec{\theta}) \times P(\vec{\theta})) \Leftrightarrow \tag{17}$$

$$\ln(P(\vec{\theta}|\vec{n}_{dt})) = \ln(\prod_{t\in T} \theta_t^{n_{td}} \times \prod_{t\in T} \theta_t^{\alpha-1}) + const \Leftrightarrow \tag{18}$$

$$\ln(P(\vec{\theta}|\vec{n}_{dt})) = (\sum_{t\in T} (n_{td} + \alpha - 1)\ln(\theta_t) + const \Leftrightarrow \tag{19}$$

And boundary:

$$\sum_{t\in T} \theta_t = 1 \tag{20}$$

We may write down a Lagrangian from 19 and 20

$$L(\vec{\theta}, \lambda) = \sum_{t\in T} (n_{td} + \alpha - 1)\ln(\theta_t) - \lambda(\sum_{t\in T} \theta_t - 1) \tag{21}$$

$$\frac{\partial L}{\partial \theta_t} = \frac{n_{td} + \alpha - 1}{\theta_t} - \lambda = 0 \tag{22}$$

$$\theta_t \propto n_{td} + \alpha - 1 \tag{23}$$

Where n_{td} was estimated in E-step and $\theta_{td}\frac{\partial R(\Phi,\Theta)}{\partial \theta_{td}} = \theta_{td}\frac{\partial(\alpha-1)\ln(\theta_{td})}{\partial \theta_{td}} = \alpha - 1$ is regularizer.
See example of regularizer implementation in ru.ispras.modis.tm.regularizer.SymmetricDirichlet

[2] `http://en.wikipedia.org/wiki/Dirichlet_distribution`

3 Quick start

In order to use topic modeling one have to perform the following step:

1. Read documents
2. Split each document into a sequence of words
3. Replace words by it serial number
4. Build a topic model.
5. Train the topic model.
6. Save results or use it in application.

In this section would be shown how to perform each step.

3.1 Read the data

First of all we have to read the data from disc. It worth mention that text normalization is non-goal of our project, thus text should be already preprocessed. It step is depend on the organization of input data, so it is no use to describe this step in general way. Instead of it we describe this step for our example file, one may easy modify this step for his use case. It worth to mention that our project do not performs any text normalization. In this introduction we would use file arxiv.part from directory examples. This is a thousand of scientific articles from Arxiv[3]. Texts was preprocessed, words was separated by space. One line corresponds to one document.
First of all one have to read data, and split each line by space:

```
val lines = Source.fromFile(new File("examples/arxiv.part")).getLines()
val wordsSequence = lines.map(line => line.split(" "))
```

Than one have to construct a TextualDocument from each sequence of words:

```
val textualDocuments = wordsSequence.map(words =>
new TextualDocument(Map(Category -> words)))
```

Each TextualDocument is a map from attribute to the correspond sequence of words. For example attribute may denote language of text in the case of multilingual topic modeling. If there is only one attribute (as in our case) one may use Category.

3.2 Replace words by it serial number

In order to save the memory in our application we have to replace words by it serial number and group the same words together.

$$Seq(duck,\ \ duck,\ \ duck,\ \ goose) \rightarrow Seq((0,3),(1,1)))$$

[3] `http://arxiv.org/`

For this purpose we may use object Numerator. It takes into input sequence of textual documents, replace words by it serial number, group the same words and return sequence of documents. It also return Alphabet (map form word to serial number and vice versa) `val (documents, alphabet) = Numerator(textualDocuments)`.

3.2.1 How to use alphabet

Alphabet is hold map from words to it index and vice versa. Thus it allow to

1. get words by it index and attribute:
   `alphabet(Category, 1) // goose`
2. get index of word by attribute and word:
   `alphabet.getIndex(Category, duck) // 0`
3. get the number of words, corresponding to the given attribute
   `alphabet.numberOfWords(Category) // 100500`

3.3 Build model

One may build model with class Builder (see package ru.ispras.modis.builder). Our project provide three types of builders:

1. PLSABuilder – builds a standard PLSA (as it described in original work [**?**])
2. LDABuilder – builds an LDA (PLSA with Dirichlet regularizer, for details section 2.5)
3. RobustPLSABuilder – build a robust PLSA. Robust PLSA take into account that some words are too rare and can't be explained by any topic (we call this kind of words "noise"). Some other words may be too common (for example a stop-word, that we forget to remove in the preprocessing step). This kind of words may be refer to any topic (we call that kind of word "background").

In this example we would use a standard PLSA, other builders works analogously. To build plsa one should

1. Set number of topic in model:
   `val numberOfTopics = 25`
2. Set the number of iteration in EM-algorithm:
   `val numberOfIteration = 100`
3. Create instance of class java.util.Random:
   `val random = new Random()`
4. And build the model:
   ```
   val builder = new PLSABuilder(numberOfTopics, alphabet, documents,
   random, numberOfIteration)
   ```

3.4 Training a model

Now we build a model and may perform stochastic matrix decomposition(train the model)
`val trainedModel = plsa.train(documents)`
trainedModel hold the matrices Φ and Θ (see 2.3) Φ is distribution of words by topic thus the number in the intersects of $i-th$ row and $j-th$ column show the probability to generate word j from topic i. Each attribute set according to the matrix Φ. To obtain matrix Φ, corresponds to attribute Category:
`val phi = trainedModel.phi(Category)`
Θ is a distribution of documents by topic thus the number in the intersects of $i-th$ row and $j-th$ column show the weight of topic j in document i. To obtain matrix Θ:
`val theta = trainedModel.theta`

3.4.1 How to interpret results

If you don't familiar with topic modeling read at least 2.1. In our library we story distribution of words by topic in matrix Φ, distribution of documents by topics in matrix Θ. To see the probability to generate word w from topic t one may use method
`trainedModel.phi(Category).probability(t, w)`
To see the probability to generate topic t in document d
`trainedModel.theta.probability(d, t)`

3.4.2 How to work with matrices

Matrices Φ and Θ hold different information, but support the similar methods. Both classes hold matrix of expectation and stochastic matrix. Expectation matrix hold values, estimated by E-step (see 2.4), stochastic matrix contain a probability. One describe method of classes Phi and Theta in table 1

We also have an utility to save matrix in file:
`TopicHelper.saveMatrix("path/to/file", matrix)` where matrix may be Φ or Θ. One also may write top n words from each topic to estimate topic coherence with `TopicHelper.printAllTopics(n, phi, alphabet)`

Table 1: Method of class Phi and class Theta

method	Phi	Theta
addToExpectation(row, $column$, $value$)	row - topic index column - word index $column$ - topic index	row - document index value - n_{dwt} value - n_{dwt}
probability($rowIndex$, $columnIndex$)	probability to generate word $columnIndex$ from topic $rowIndex$	weight of topic $columnIndex$ in document $rowIndex$
expectation($rowIndex$, $columnIndex$)	return expected value for words, $columnIndex$ generated from topic $rowIndex$: n_{wt}	return expected value for words, generated from topic $columnIndex$ in document $rowIndex$: n_{dt}
numberOfRows	number of topics	number of documents
numberOfColumns	number of words	number of topics
dump()	replace negative value in expectation matrix by zero, replace value in stochastic matrix by corresponding values from expectation matrix, normalize stochastic matrix and replace values in expectation matrix by zero	

4 Building bricks of our model

In the previous section we describe how to build a model, but we steel do not describe how model works. Let's fix this shortcoming. Our model support a multilingual documents, thus document may contain a few text with different attributes, all texts in one document inherit the same distribution of topics. Thus we have had one matrix Φ per attribute and one matrix Θ for all attribute. In order to train our model we have to

1. Generate some initial approximation for matrix Θ and every matrices Φ
2. Perform E-step and estimate the number of words w in document d, produced by topic t for every attribute.
3. Apply requlatizer to every matrix. (see 2.5)
4. Perform M-step for every matrix.
5. Sparsify matrices Φ and matrix Θ. (see 2.1.3)
6. Check the stopping criteria and return training model if it time to stop or return to the step 2 otherwise.

The main part of our project is PLSABricks, it do the main part of work. One brick process one attribute. It performs E-step, apples requlatizer to matrix Φ, performs M-step to the matrix Φ and sparsifies matrix Φ. PLSA include one brick per attribute, regularizer and sparsifier for matrix Θ, stopping criteria.

5 Regularizer

5.1 Regularizer

The description of theory are described in 2.5. Hear we would describe the detail of implementation of regularizer in our project and describe some standard regularizers, which are implemented in our library.

5.1.1 Implementation

One may find regularizers in ru.ispras.modis.regularizer. Any regularizer should inherit from class Regularizer. In order to implement his own regularizer user have to implement methods regularizePhi, regularizeTheta and apply.

Implementation of regularizePhi
As one may see in 13,

$$\varphi_{wt} \propto \left(\hat{n}_{wt} + \varphi_{wt} \frac{\partial R(\Phi, \Theta)}{\partial \varphi_{wt}} \right)_+ \tag{24}$$

Thus, in order to implement method regularizePhi one have to:

- calculate $\varphi_{wt} \frac{\partial R(\Phi,\Theta)}{\partial \varphi_{wt}}$ using matrix Φ and matrix Θ for each word w and each topic t.
  In order to take i-th row and j-th column in matrix Φ one may use

  $$phi.probability(i, j)$$

  where i– topic index, j– word index.
  Analogously for matrix Θ:

  $$theta.probability(i, j)$$

  , where i-topic index, j - document index.

- Add these values to matrix of expectation, in order to add $\varphi_{wt} \frac{\partial R(\Phi,\Theta)}{\partial \varphi_{wt}}$ to expectation matrix n_{wt} one should use method

  $$phi.addToExpectation(t, w, \varphi_{wt} \frac{\partial R(\Phi, \Theta)}{\partial \varphi_{wt}})$$

Implementation of regularizeTheta
This method is analogous to the previous paragraph. One have to calculate

$$\theta_{td} \frac{\partial R(\Phi, \Theta)}{\partial \varphi_{td}}$$

and add it to expectation of n_{dt}.

Implementation of apply

apply is used to calculate 12 instead of log likelihood (and corresponding perplexity). If you wont calculate log likelihood or you are lazy to implement this method return 0f.

5.1.2 Implemented requlatizer

Now in our project are implemented following regularizers:

- ZeroRegularizer, it regularizer do nothing, if you don't wont to use regularizer use this one
- RegularizerSum allow to apply a sequence of regularizers sequentially. For example if you have a few requlatizer: r_1, r_2, r_3 and you wont to apply them sequentially. For this aim
  `import ru.ispras.modis.regularizer.Regularizer.toRegularizerSum`
  `val regularizerSum =` r_1 `+` r_2 `+` r_3
- SymmetricDirichlet, it regularizer add a dirichlet prior to the distribution of document by topic and words by topic, it is used to convert PLSA into LDA (see 2.1.4)

6 Sparsifier

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. Thus some weights in matrices Φ and Θ may be replaced by zero without drop of quality of our model. For this aim we use sparsifier. Any sparsifier should inherit from trait Sparsifier and implement a method apply. This method take into input MatrixForSparsifier, numberOfIteration. Sparsifier may obtain probabilities from matrix and decide which cells should be replaced by zero, based on value, number of iteration and internal state. To replace i-th row and j-th column by zero one should call
`matrix.setZero(i, j)`

6.1 Implemented sparsifier

We implement

- ZeroSparsifier, it do nothing. If you don't want to use sparsifier you should use this one.
- ThresholdSparsifier - it class replace value by zero if it less than threshold, but not more than maxNumberOfZeroised by an iteration and only after startIteration.

7 Stopping Criteria

There is no conventional way to define how many iteration should be executed. One of the common way to do that is to execute a fixed number of iterations, where the number of iteration is a some heuristic. There are another stopping criterion, thus we allow user to implement his own criterion. Any criterion should inherit trait StoppingCriteria. Stopping criterion decide to stop based on perplexity after this iteration, perplexity after previous iteration, number of iterations and internal state.

8 Multilingual topic modeling

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.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[3] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 50–57, New York, NY, USA, 1999. ACM.