Topic Modeling Library and Framework User Guide

Avanesov Valeriy
Kozlov Ilya

June 4, 2015

Contents

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-defined regularizer, sparsifier and stopping criteria.
- Ability to use multilingual topic modeling.

The paper is organized as follow: section 3 contains a quick introduction to topic modeling, 4 describes how to start a project without description of details of implementation. These two sections should allow you to use LDA and PLSA model in simple use-cases. The following section describes details of implementation and is necessary for deeper understanding of project structure and implementation of your own regularizer, sparsifier and other stuff. Section 5 describes a structure of the project, section 6 describes regularizers and explains how to implement your own regularizer, section 7 describes what sparsifier is and how to implement you own sparsifier. Section 8 for the description of how to define that convergence of EM-algorithm is achieved. Section 9 describes (actually, it doesn't, though we are working on it) how to use model in multilingual setting.

2 How to build project

In this section we will describe how to build a project. To build this project you need to have the following program:

1. git. It is necessary to obtain source from github. In Ubuntu one can obtain git by typo

   ```
   sudo apt-get install git
   ```

2. The project is written on Scala, thus you need to have Scala. One can obtain the latest version from `http://www.scala-lang.org/`.

3. Java is also would be useful. One can obtain the latest version from `https://www.oracle.com/java/`

4. To build the project you need to have sbt. One can obtain the latest version from `http://www.scala-sbt.org/`

Now lets build tm. First of all we have to obtain the source:

```
git clone https://github.com/ispras/tm.git
```

Go to the directory with tm:

```
cd tm
```

And now we can build project:

```
sbt assembly
```

The .jar file will occur in target/scala-2.11/ directory. Lets run an example task

```
java -classpath target/scala-2.11/tm-assembly-1.0.jar ru.ispras.modis
    .tm.scripts.QuickStart
```

It will start topic modeling on the 1000 scientific articles. You can add .jar file in you project.

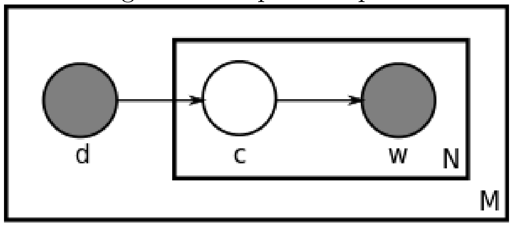
3 Introduction to topic modeling

Generative model

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

- 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 every document by topics.

Polylingual topic model

Suppose one has a collection of documents in 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 a map from attributes $\rightarrow$ sequences of words.

Robust PLSA

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

- with probability $\propto \gamma$ generate word from noise

- with probability $\propto \varepsilon$ generate word from background
- with probability $\propto 1 - \gamma - \varepsilon$ generate word from topic distribution as in 3

γ and ε are hyperparameters of the model.

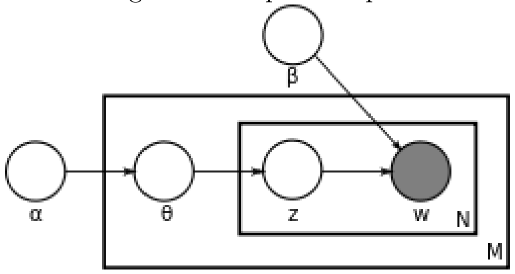
Sparse PLSA

A single 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 replaces some of topics weights in document by zero and replaces 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.

LDA

LDA is an extension of PLSA, LDA was described in [2] 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 topics has Dirichlet [1] (usually, symmetric) prior distribution and we maximize marginal distribution $P(data, parameters)$ instead of conditional one - $P(data|parameters)$.

Figure 2: Graphical representation of LDA generative process



α and β are hyperparameters of the model.

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 can obtain φ_{wt} and θ_{td} as a 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 boundaries

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

and

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

Topic modeling as matrix decomposition

Kullback-Leibler divergence

Kullback-Leibler divergence is a non-negative semi-metric between a pair of probability distributions:

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

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.

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.

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 $p(t|d)$

Complexity

Obviously, time complexity for every iteration is $O(T \times n_{uniq} \times D)$.
Space complexity is given by $O(W \times T + D \times T)$.
Here n_{uniq} is the average number of distinct tokens in a document, D stands for number of documents and W – for the size of vocabulary.

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), select number of topics. 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.

Probability interpretation

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

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

Take the logarithm of the 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`

4 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.

First of all one may look at tm/src/main/scala/ru/ispras/modis/tm/script-s/QuickStart.scala

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

Read the data

First of all we have to read the data from disc. It worth mentioning that text normalization is non-goal of our project, thus text should be already preprocessed. This step depends on the organization of input data, so it is no use to describe this step in general way. Instead of this we describe this step for our example file, one may easy modify this step for his own use case. In this introduction we would use file arxiv.part from directory examples. This is a thousand of scientific papers from Arxiv[3]. Texts was preprocessed, words were separated by space. Every line corresponds to a single document.
First of all one has 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(DefaultAttributeType -> 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 can use `DefaultAttributeType`. Furthermore, in such a case one can use `SingleAttributeNumerator` instead on `Numerator` – it takes `Seq[String]` as inputes and implicitly uses `DefaultAttributeType`.

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.

$$Seq(duck,\quad duck,\quad duck,\quad goose) \to Seq((0,3),(1,1)))$$

[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)`.

How to use alphabet

Alphabet holds a map from words to their indices 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`

Build model

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

1. PLSABuilder – builds a standard PLSA (as it is described in original paper [3])
2. LDABuilder – builds an LDA (PLSA with Dirichlet regularizer, for details section 3)
3. RobustPLSABuilder – builds a robust PLSA. Robust PLSA takes 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). Words of this kind are referred to any topic (we call that kind of word "background").

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

1. Set number of topics in model:
   `val numberOfTopics = 25`
2. Set the number of iterations for EM-algorithm to perform:
   `val numberOfIteration = 100`
3. Create an instance of class java.util.Random:
   `val random = new Random()`

4. And build the model:
   `val builder = new PLSABuilder(numberOfTopics, alphabet, documents, random, numberOfIteration)`

Training a model

Now we build a model and can perform stochastic matrix decomposition(train the model)
`val trainedModel = plsa.train(documents)`
trainedModel holds the matrices Φ and Θ (see 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`

How to interpret the results

If you are not familiar with topic modeling read 3. In our library we store the distribution of words by topic in matrix Φ, distribution of documents by topics in matrix Θ. In order to obtain the probability to generate word w from topic t one may use method
`trainedModel.phi(Category).probability(t, w)`
In order to obtain the probability to generate topic t in document d
`trainedModel.theta.probability(d, t)`

How to deal 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 holds values, estimated by E-step (see 3), stochastic matrix contain a probability. Method of classes Phi and Theta are explained 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	

5 Building bricks of our model

In the previous section we have described how to build a model, but we still have not described how our implementation works. Let's fix this shortcoming. Our model supports a multilingual documents, thus document may contain few texts related to different attributes, every text from the same document inherits the same distribution of topics. Thus we have one matrix Φ per attribute and one matrix Θ for all attributes. In order to train our model we have to

1. Generate some initial approximation for matrix Θ and every matrix Φ
2. Perform E-step and estimate the number of words w in document d, produced by topic t for every attribute.
3. Apply reqularizer to every matrix. (see 3)
4. Perform M-step for every matrix.
5. Sparsify matrices Φ and matrix Θ. (see 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 does the main part of work. Every brick processes a single attribute. It performs E-step, apples reqularizer to matrix Φ, performs M-step to the matrix Φ and sparsifies matrix Φ. PLSA includes one brick per attribute. Also it stores regularizer and sparsifier for matrix Θ and stopping criteria.

6 Regularizer

The theory is given in section 3. Here we describe the details of implementation of regularizer in our project and describe some standard regularizers, which are implemented in our library.

Implementation

One can 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.

Implemented reqularizer

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 reqularizer: 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 3)

7 Sparsifier

One document often corresponds to only a few topics, not to everyone. Analogously, word 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 purpose we use sparsifier. Any sparsifier should inherit from trait Sparsifier and implement a method apply. This method takes into input MatrixForSparsifier and numberOfIteration. Sparsifier may obtain probabilities from matrix and decide which cells should be replaced by zero, based on value, number of iteration and its state. To replace i-th row and j-th column by zero one should call

```
matrix.setZero(i, j)
```

Implemented sparsifier

We implement

- ZeroSparsifier, it does nothing. If you don't want to use sparsifier you should use this one.
- ThresholdSparsifier - this class replaces value with zero if it is less than threshold if it did not replaced in this way more than `maxNumberOfZeroised` during this method call and the current number of iteration exceeds `startIteration`.

8 Stopping Criteria

There is no conventional way to define how many iteration should be performed. 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 other stopping criterion, thus we allow user to implement his own criterion. Any criterion should inherit criterion `trait StoppingCriteria`. Stopping criterion decides to stop based on perplexity after this iteration, perplexity after previous iteration, number of iterations and internal state.

9 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.