topicmodel-lib Documentation

Release 0.3.1

DSLab

Jul 24, 2020

Getting started

topicmodel-lib is a Python library for *topic modeling* - a field which provides an efficient way to discover hidden structures/semantics in massive data. Latent Dirichlet Allocation (LDA) is a popular model in this field and we focus on methods for learning LDA by online or stream scheme.

CHAPTER 1

Features

- Our library provides efficient algorithms for learning LDA from large-scale data. It includes the state-of-the-art learning methods at the present
- We also implement Cython code (a programming language that makes writing C extensions for the Python language as easy as Python itself) to increase speed of some algorithms
- We've also designed the visualization module to help users understand and explore the result that the model discovers after learning

CHAPTER 2

Getting started in 30s

Training data

Because we need to learn the model from the massive data, the loading whole of training data into memory is a bad idea. Therefore, the online/streaming learning algorithms are usually preferred in this case. Training data should be stored in a file and with a specific format. Our library supports 3 formats of training data and in here, we'll demo with ap corpus

Tutorial

First, class *DataSet* provides some functions to processing the training data:

```
>>> from tmlib.datasets import DataSet
>>> data = DataSet('ap_train_raw.txt', batch_size=100, passes=5, shuffle_every=2)
```

Learning LDA by Online VB method (Hoffman, 2010):

```
>>> from tmlib.lda import OnlineVB
>>> onlinevb = OnlineVB(data, num_topics=20)
>>> model = onlinevb.learn_model()
```

You can see the topics which is discovered by Online VB:

```
>>> model.print_top_words(5, data.vocab_file, display_result='screen')
```

For a more in-depth tutorial about topicmodel-lib, you can see documentation. In the examples folder of the repository, you will see the example code as well as training data. You can run a demo to understand how the library work

CHAPTER 3

Installation

Dependencies

To use the library, your computer must installed all of these packages first:

- Linux OS (Stable on Ubuntu)
- Python version 2 (stable on version 2.7)
- Numpy >= 1.8
- Scipy >= 0.10
- nltk (Natural Language Toolkit)
- Cython
- Pandas >= 0.20

User Installation

- Installing by pip

```
$ sudo pip install tmlib
```

- Installing by download project

  After download project, you install by running file setup.py in folder topicmodel-lib as follow:

  First, build the necessary packages:

```
python setup.py build_ext --inplace
```

  if you need permission to build:

```
sudo python setup.py build_ext --inplace
```

  After that, install library into your computer:

```
sudo python setup.py install
```

CHAPTER 4

Support

If you have an open-ended or a research question, you can join and contact via:

- Google Group
- Facebook Group

CHAPTER 5

License

The project is licensed under the MIT license.

5.1 Introduction

A very short introduction into topic models and how to solve them using topicmodel-lib. This document also introduces some basic concepts and conventions.

5.1.1 Topic models

Topic models are probabilistic models of document collections that use latent variables to encode recurring patterns of word use (Blei, 2012). Topic modeling algorithms are inference algorithms; they uncover a set of patterns that pervade a collection and represent each document according to how it exhibits them. These patterns tend to be thematically coherent, which is why the models are called "topic models." Topic models are used for both descriptive tasks, such as to build thematic navigators of large collections of documents, and for predictive tasks, such as to aid document classification. Topic models have been extended and applied in many domains

Latent Dirichlet Allocation (LDA) is the simplest topic model, LDA is a generative probabilistic model for collections of discrete data such as text corpora.

Large-scale learning

Modern data analysis requires computation with massive data. These problems illustrate some of the challenges to modern data analysis. Our data are complex and high-dimensional; we have assumptions to make from science, intuition, or other data analyses that involve structures we believe exist in the data but that we cannot directly observe; and finally, our data sets are large, possibly even arriving in a never-ending stream. We deploy this library to computing with graphical models that are appropriate for massive datasets, data that might not fit in memory or even be stored locally. This is an efficient tool for learning LDA at large scales

Learning methods for LDA

To learn LDA at large-scale, a good and efficient approach is stochastic learning (online/streaming methods). The learning process includes 2 main steps:

- Inference for individual document: infer to find out the **hidden local variables**: topic proportion θ or topic indices **z**. To deal with this, we can estimate directly or estimate their distribution P(θ | γ), P(**z** | ϕ) (γ, phi called "variational parameters").
- Update **global variable** in a stochastic way to find out directly topics β or we can estimate topics by finding out its distribution P(β | λ) (estimating variational parameter λ). Global variable here maybe β or λ depend on each stochastic methods.

Indeed, this phase is as same as training step in machine learning.

5.2 Corpus

A corpus is a collection of digital documents. This collection is the input to topicmodel-lib from which it will infer the structure of the documents, their topics, topic proportions, etc. The latent structure inferred from the corpus can later be used to assign topics to new documents which were not present in the training corpus. For this reason, we also refer to this collection as the training corpus. No human intervention (such as tagging the documents by hand) is required - the topic classification is unsupervised.

5.2.1 Data Format

Because we need to learn the model from the massive data, the loading whole of training data into memory is a bad idea. Corpus used for training should be stored in a file and with a specific format. Our library supports 3 formats of data:

Raw Text

```
raw_corpus = ["Human machine interface for lab abc computer applications",
              "A survey of user opinion of computer system response time",
              "The EPS user interface management system",
              "System and human system engineering testing of EPS",
              "Relation of user perceived response time to error measurement",
              "The generation of random binary unordered trees",
              "The intersection graph of paths in trees",
              "Graph minors IV Widths of trees and well quasi ordering",
              "Graph minors A survey"]
```

The raw corpus must be stored in a file. Each document is placed in 2 pair tag <DOC></DOC> and <TEXT></TEXT> as follow

```
<DOC>
# maybe is title or others or nothing
<TEXT>
# Plain text
</TEXT>
</DOC>
```

You can see raw AP corpus for example

Term-frequency (tf)

Term-frequency format (**tf**) is derived from Blei, 2003. This is a corpus which we achieve after preprocessing raw corpus. We also extract to a vocabulary set for that corpus (unique terms in whole corpus)

Under LDA, the words of each document are assumed exchangeable. Thus, each document is succinctly represented as a sparse vector of word counts. The data is a file where each line is of the form:

[N] [term_1]:[count] [term_2]:[count] ... [term_N]:[count]

where [N] is the number of unique terms in the document, and the [count] associated with each term is how many times that term appeared in the document. Note that [term_i] is an integer which indexes the term (index of that term in file vocabulary); it is not a string.

For example, with corpus as raw_corpus above and file vocabulary is:

```
0. "human"
1. "machine"
2. "interface"
3. "lab"
4. "abc"
5. "computer"
6. "applications"
7. "survey"
8. "user"
9. "opinion"
10. "system"
11. "response"
12. "time"
13. "eps"
14. "management"
15. "engineering"
16. "testing"
17. "relation"
18. "perceived"
19. "error"
20. "measurement"
21. "generation"
22. "random"
23. "binary"
24. "unordered"
25. "trees"
26. "intersection"
27. "graph"
28. "paths"
29. "minors"
30. "widths"
31. "quasi"
32. "ordering"
```

The **tf** format of corpus will be:

```
7 0:1 1:1 2:1 3:1 4:1 5:1 6:1
7 7:1 8:1 9:1 5:1 10:1 11:1 12:1
5 13:1 8:1 2:1 14:1 10:1
5 10:2 0:1 15:1 16:1 13:1
7 17:1 8:1 18:1 11:1 12:1 19:1 20:1
5 21:1 22:1 23:1 24:1 25:1
4 26:1 27:1 28:1 25:1
```

(continues on next page)

(continued from previous page)

```
6 27:1 29:1 30:1 25:1 31:1 32:1
3 27:1 29:1 7:1
```

Term-sequence (sq)

Each document is represented by a sequence of token as follow

[token_1] [token_2] [token_3]....

[token_i] also is index of that token in vocabulary file, not a string. (maybe exist that [token_i] = [token_j]) The **sq** format of the corpus above will be:

```
0 1 2 3 4 5 6
7 8 9 5 10 11 12
13 8 2 14 10
10 0 10 15 16 13
17 8 18 11 12 19 20
21 22 23 24 25
26 27 28 25
27 29 30 25 31 32
27 29 7
```

5.3 Guide to learn model

In this phase, the main task is to find out the global variable (topics) - in this project, we call it named *model* for simple. We designed the state-of-the-art methods (online/streaming learning): Online VB, Online CVB0, Online CGS, Online OPE, Online FW, Streaming VB, Streaming OPE, Streaming FW, ML-OPE, ML-CGS, ML-FW

All of this methods are used in the same way. So, in this guide, we'll demo with a specific method such as Online VB. This method is proposed by Hoffman-2010, using stochastic variational inference

5.3.1 Data Preparation

Make sure that your training data must be stored in a text file and abide by the *Data Format*: **tf**, **sq** or **raw text**

We also support the preprocessing module to work with the raw text format, you can convert to the tf or sq format. But if you don't want to use it, it's OK because we integrated that work in class `DataSet`. Therefore, the first thing you need to do is create an object `DataSet`

```
from tmlib.datasets import DataSet
# data_path is the path of file contains your training data
data = DataSet(data_path, batch_size=5000, passes=5, shuffle_every=2)
```

The statement above is used when *data_path* is the raw text format. If your training file is the tf or sq format. You need to add an argument is the vocabulary file of the corpus as follow:

```
# vocab_file is the path of file vocabulary of corpus
data = DataSet(data_path, batch_size=5000, passes=5, shuffle_every=2, vocab_
↪file=vocab_file)
```

The parameters **batch_size**, **passes**, **shuffle_every** you can see in documentation here

5.3.2 Learning

First, we need to create an object `OnlineVB`:

```
from tmlib.lda import OnlineVB
onl_vb = OnlineVB(data=data, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.
↪9)
```

`data` is the object which created above. Parameter **num_topics** number of requested latent topics to be extracted from the training corpus. **alpha**, **eta** are hyperparameters of LDA model that affect sparsity of the topic proportions (θ) and topic-word (β) distributions. **tau0**, **kappa** are learning parameters which are used in the update global variable step (same meaning as learning rate in the gradient descent optimization)

Start learning by call function **learn_model**:

```
model = onl_vb.learn_model()
```

The returned result is an object LdaModel

You can also save the model (β or λ) or some statistics such as: learning time, sparsity of document in the learning process

```
model = onl_vb.learn_model(save_statistic=True, save_model_every=2, compute_sparsity_
↪every=2, save_top_words_every=2, num_top_words=10, model_folder='models')
```

The result is saved in folder *models*. More detail about this parameters, read here

One more thing, the topic proportions (θ) of each document in the corpus can be saved in a file `.h5`. This work is necessary for visualization module but it'll make the learning time slower. So, be careful when using it!

```
# for example: path_of_h5_file = 'models/database.h5'
model = onl_vb.learn_model(save_topic_proportion=path_of_h5_file)
```

5.3.3 Saving model, display topics

After the learning phase as above, you can save the topic distribution (*model* - β or λ)

```
# path_to_save is the path of file to save model
model.save_model(path_to_save, file_type='binary')
```

File *path_to_save* is the `.npy` file if type of file is binary or is the `.txt` file if **file_type** is `'txt'`

You also can display the topics discovered

```
# display topics, print the top 10 words of each topic to screen
model.print_top_words(10, data.vocab_file, display_result='screen')
```

If you want to save in a file:

```
# path_file is to which data is saved
model.print_top_words(10, data.vocab_file, display_result=path_file)
```

5.4 Inference for new documents

After learning phase, you have the *model* - topic distributions (β or λ). You want to infer for some documents to find out what topics these documents are related to. We need to estimate topic-proportions θ

First, create an object `DataSet` to load new documents from a file.

If data format in that file is *Raw Text*, you need the vocabulary file used in learning phase

```
from tmlib.datasets import DataSet

data = DataSet()
# vocab_file is the vocabulary file used in learning phase
new_corpus = data.load_new_documents(file_new_docs, vocab_file=vocab_file)
```

or if data format is the **tf** or **sq** format. The statement simply is:

```
new_corpus = data.load_new_documents(file_new_docs)
```

After that, you have to load the model which is saved in the learning phase into object `OnlineVB`

```
# create object LdaModel
learnt_model = LdaModel()
# read topic distribution from file
lda_model.load_model(path_file_to_read)

# load lda_model into OnlineVB
from tmlib.lda import OnlineVB
online_vb = OnlineVB(lda_model=learnt_model)
```

Call `infer_new_docs` function to rum inference

```
gamma = online_vb.infer_new_docs(new_corpus)
# you can estimate topic proportion theta from variational parameter gamma
theta = online_vb.estimate_topic_proportion(gamma)
```

5.5 Example

You can see in example folder. We prepared the AP corpus including both raw corpus and term-frequency corpus. In here, we'll show code with both of type corpus and use method Online OPE (known is fast than Online VB)

- Raw AP corpus

```
from tmlib.lda import OnlineOPE
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning
onl_ope = OnlineOPE(data=data, num_topics=20, alpha=0.2)
model = onl_vb.learn_model()
# save model (beta or lambda)
model.save_model('topic_distribution.npy')
# display top 10 words of each topic
```

(continues on next page)

(continued from previous page)

```
model.print_top_words(10, data.vocab_file, display_result='screen')

# inference for new documents
vocab_file = data.vocab_file
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_file=vocab_
↪file)
topic_proportions = onl_ope.infer_new_docs(new_corpus)
```

Topics:

```
topic 0: water, environmental, people, trust, earth, pollution, taxes, claims,␣
↪air, boat,
topic 1: year, people, years, mrs, police, time, day, family, state, women,
topic 2: percent, year, company, department, million, plant, health, state,␣
↪report, study,
topic 3: police, government, people, military, iraq, army, killed, officials,␣
↪israel, war,
topic 4: cent, cents, weather, lower, temperatures, bushel, snow, inches, coast,␣
↪central,
topic 5: billion, deficit, housing, japan, japanese, trade, fair, cuba, imports,␣
↪exports,
topic 6: cbs, abc, williams, miles, area, earthquake, quake, homeless, pope, john,
topic 7: bush, president, dukakis, states, reagan, congress, campaign, house,␣
↪united, south,
topic 8: campaign, state, northern, mexico, republican, police, county, alabama,␣
↪cuomo, governor,
topic 9: trade, takeshita, deconcini, oil, pension, committee, keating, fernandez,
↪ lawsuit, illness,
topic 10: court, case, attorney, trial, judge, federal, charges, law, justice,␣
↪jury,
topic 11: party, government, political, workers, national, labor, opposition,␣
↪people, elections, country,
topic 12: million, tax, sales, income, year, cash, estate, assets, money, billion,
topic 13: school, movie, film, board, parents, ban, mca, theater, roberts, fees,
topic 14: students, computer, farmers, teachers, smoking, schools, student,␣
↪stolen, kasparov, faculty,
topic 15: dollar, yen, late, gold, london, bid, ounce, bank, thursday, dealers,
topic 16: percent, market, year, stock, million, prices, billion, rose, exchange,␣
↪index,
topic 17: soviet, gorbachev, united, union, president, officials, west, year,␣
↪germany, east,
topic 18: bill, senate, house, kennedy, sen, rep, measure, humphrey, director,␣
↪thompson,
topic 19: meese, museum, disney, school, city, board, art, smith, buildings,␣
↪memorial,
```

- tf format

```
from tmlib.lda import OnlineOPE
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train.txt', batch_size=100, passes=5, shuffle_
↪every=2, vocab_file='data/vocab.txt')
# learning
onl_ope = OnlineOPE(data=data, num_topics=20, alpha=0.2)
model = onl_vb.learn_model()
```

(continues on next page)

(continued from previous page)

```
# save model (beta or lambda)
model.save_model('topic_distribution.npy')
# display top 10 words of each topic
model.print_top_words(10, data.vocab_file, display_result='screen')

# inference for new documents
new_corpus = data.load_new_documents('data/ap_infer.txt')
topic_proportions = onl_ope.infer_new_docs(new_corpus)
```

Topics:

```
topic 0: two, new, people, i, years, first, officials, time, fire, day,
topic 1: israel, minister, prime, vietnam, thatcher, party, opec, ministers,␣
↪demjanjuk, labor,
topic 2: million, percent, futures, year, market, bank, analysts, new, cbs, nbc,
topic 3: dukakis, jackson, democratic, presidential, campaign, candidates,␣
↪candidate, vote, voters, delegates,
topic 4: million, company, new, billion, inc, corp, board, year, court, federal,
topic 5: bush, united, states, president, trade, billion, house, congress, new,␣
↪budget,
topic 6: stock, market, dollar, trading, exchange, yen, prices, late, index, rose,
topic 7: korean, korea, city, village, police, north, st, traffic, koreas, citys,
topic 8: police, people, killed, two, government, army, military, officials,␣
↪three, city,
topic 9: south, africa, african, black, elections, party, national, war, mandela,␣
↪blacks,
topic 10: states, united, nicaragua, noriega, drug, contras, court, coup,␣
↪humphrey, manila,
topic 11: reagan, china, nuclear, study, b, prisoners, fitzwater, researchers,␣
↪games, animals,
topic 12: i, new, people, years, percent, year, last, state, time, two,
topic 13: trial, case, prison, charges, convicted, jury, attorney, guilty,␣
↪sentence, prosecutors,
topic 14: rain, northern, texas, inches, california, central, damage, santa,␣
↪hospital, valley,
topic 15: soviet, government, gorbachev, union, party, president, political, two,␣
↪news, people,
topic 16: service, offer, court, companies, firm, ruling, information, appeals,␣
↪operations, services,
topic 17: water, care, homeless, environmental, pollution, fair, species, air,␣
↪disaster, farm,
topic 18: percent, year, cents, oil, prices, west, german, rate, sales, price,
topic 19: air, plane, flight, two, iraq, soviet, force, kuwait, airport, iraqi,
```

5.6 LDA Model

5.6.1 class LdaModel

```
tmlib.lda.LdaModel(num_terms=None, num_topics=None, random_type=0)
```

This class provides some function to help you manipulate model (λ or β): you can save model, load model or display words of topics...

Parameters

- **num_terms**: int, default: None

  number of words in vocabulary file

- **num_topics**: int, default: None

  number of topics

- **random_type**: int, default: 0

  Initialize randomly array of λ (or β) (size num_topics x num_terms). If random_type = 0, model is initialized with uniform distribution. Otherwise, initialized with gamma distribution

Attributes

- **num_terms**: int
- **num_topics**: int
- **model**: array 2 dimentions (num_topics x num_terms)

  λ or β. Depend on the learning method: λ in case Online VB, Online CVB0, Online CGS, Streaming VB and β for Online OPE, Online FW, Streaming OPE, Streaming FW, ML-CGS, ML-OPE, ML-FW

- **presence_score**: a measure allow us to rank the topics. If a topic has the high value of presence_score, it means that corpus is related to that topic very much

Methods

- __init__(*num_terms=None, num_topics=None, random_type=0*)
- **normalize** ()

  Used for estimating β from λ. This function is usually used for regularized methods ML-CGS, ML-FW, ML-OPE

- **print_top_words** (self, num_words, vocab_file, show_topics=None, display_result=None, type='word', distribution=False)

  Display words of topics on the screen or save into file

  - **Parameters**:
    - **num_words**: int,

      number of words of each topic is displayed

    - **vocab_file**: string,

      path of file vocabulary

    - **show_topics**: int, default: None

      number of topics is displayed. By default, all of topics are displayed

    - **display_result**: string, default: None

      path of file to save words into, or 'screen' if you want to display topics on the screen. By default, if display_result=None, nothing happen and the method return the result as list python

  - **type**: string, default: 'word'

    You can print index of word in the vocabulary file by set type='index'. If type='word' by default, the result returned is words (type string)

  - **distribution**:

    If you only want to display words, then distribution=False. But you can display words along with term distribution of it by set distribution = True

- **load_model** (path_file)

  loading the learned model (λ or β) from file named *path_file*

- **save_model** (path_file, file_type='binary')

  saving model into a file named path_file. By default, the type of file is binary. We can change type of file to text by set file_type='txt'

- **save** (path_file)

  you can save the object by using this function. All attributes of object LdaModel are saved in path_file and after that, you can restore object by using **load** function

- **load** (path_file)

  load the attributes from path_file to restore object LdaModel.

5.6.2 Example

```
from tmlib.lda import OnlineOPE
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train.txt', batch_size=100, passes=5, shuffle_
↪every=2, vocab_file='data/vocab.txt')
# learning
onl_ope = OnlineOPE(data=data, num_topics=20, alpha=0.2)
model = onl_vb.learn_model()
# save model (beta or lambda)
model.save_model('topic_distribution.npy')
# display top 10 words of each topic
model.print_top_words(10, data.vocab_file, display_result='screen')
```

Topics:

```
topic 0: two, new, people, i, years, first, officials, time, fire, day,
topic 1: israel, minister, prime, vietnam, thatcher, party, opec, ministers,␣
↪demjanjuk, labor,
topic 2: million, percent, futures, year, market, bank, analysts, new, cbs, nbc,
topic 3: dukakis, jackson, democratic, presidential, campaign, candidates, candidate,␣
↪vote, voters, delegates,
topic 4: million, company, new, billion, inc, corp, board, year, court, federal,
topic 5: bush, united, states, president, trade, billion, house, congress, new,␣
↪budget,
topic 6: stock, market, dollar, trading, exchange, yen, prices, late, index, rose,
topic 7: korean, korea, city, village, police, north, st, traffic, koreas, citys,
topic 8: police, people, killed, two, government, army, military, officials, three,␣
↪city,
topic 9: south, africa, african, black, elections, party, national, war, mandela,␣
↪blacks,
```

(continues on next page)

(continued from previous page)

```
topic 10: states, united, nicaragua, noriega, drug, contras, court, coup, humphrey,␣
↪manila,
topic 11: reagan, china, nuclear, study, b, prisoners, fitzwater, researchers, games,␣
↪animals,
topic 12: i, new, people, years, percent, year, last, state, time, two,
topic 13: trial, case, prison, charges, convicted, jury, attorney, guilty, sentence,␣
↪prosecutors,
topic 14: rain, northern, texas, inches, california, central, damage, santa, hospital,
↪ valley,
topic 15: soviet, government, gorbachev, union, party, president, political, two,␣
↪news, people,
topic 16: service, offer, court, companies, firm, ruling, information, appeals,␣
↪operations, services,
topic 17: water, care, homeless, environmental, pollution, fair, species, air,␣
↪disaster, farm,
topic 18: percent, year, cents, oil, prices, west, german, rate, sales, price,
topic 19: air, plane, flight, two, iraq, soviet, force, kuwait, airport, iraqi,
```

If you change the last statement to:

```
model.print_top_words(5, data.vocab_file, display_result='screen', distribution=True)
```

```
topic 0: (two 1006.872532), (new 997.382525), (people 957.472761), (i 847.205429),␣
↪(years 793.221432),
topic 1: (israel 280.810816), (minister 264.384617), (prime 236.849663), (vietnam 227.
↪907314), (thatcher 184.359115),
topic 2: (million 990.763364), (percent 990.581342), (futures 676.483861), (year 554.
↪327385), (market 487.330349),
topic 3: (dukakis 1443.205724), (jackson 961.280235), (democratic 666.250053),␣
↪(presidential 530.063395), (campaign 374.716751),
topic 4: (million 2094.650569), (company 1817.952267), (new 1233.623336), (billion␣
↪1064.094612), (inc 940.197448),
topic 5: (bush 3039.197872), (united 2500.479748), (states 2209.172288), (president␣
↪2184.088408), (trade 1752.369137),
topic 6: (stock 1781.654725), (market 1612.880852), (dollar 1321.883897), (trading␣
↪1136.304384), (exchange 984.091480),
topic 7: (korean 315.153683), (korea 250.382257), (city 236.793215), (village 198.
↪821685), (police 151.618394),
topic 8: (police 4616.893623), (people 2455.625615), (killed 1695.193209), (two 1638.
↪251890), (government 1423.613710),
topic 9: (south 1367.724403), (africa 749.935889), (african 713.020710), (black 693.
↪012008), (elections 511.215701),
topic 10: (states 472.340775), (united 281.539123), (nicaragua 261.313058), (noriega␣
↪209.555378), (drug 192.748472),
topic 11: (reagan 379.215621), (china 323.023153), (nuclear 283.032322), (study 275.
↪714702), (b 244.181802),
topic 12: (i 6295.795578), (new 3049.691470), (people 2959.973828), (years 2682.
↪041759), (percent 2517.415288),
topic 13: (trial 937.304453), (case 623.345300), (prison 602.181133), (charges 586.
↪927118), (convicted 564.307367),
topic 14: (rain 348.371840), (northern 346.465730), (texas 323.902501), (inches 321.
↪175532), (california 297.998834),
topic 15: (soviet 3327.753735), (government 1527.465015), (gorbachev 1422.083698),␣
↪(union 1335.393499), (party 1296.856500),
topic 16: (service 540.989098), (offer 479.423750), (court 448.157831), (companies␣
↪325.420266), (firm 258.895112),
```

(continues on next page)

(continued from previous page)

```
topic 17: (water 470.181853), (care 214.000670), (homeless 211.299046),␣
↪(environmental 189.202329), (pollution 186.071190),
topic 18: (percent 4348.697399), (year 1170.209529), (cents 1135.822631), (oil 1093.
↪987126), (prices 934.051475),
topic 19: (air 1054.797512), (plane 793.580865), (flight 762.382746), (two 597.
↪540968), (iraq 563.872035),
```

5.7 Module tmlib.datasets

This module includes some classes and utility functions which help us work with the dataset

5.7.1 class DataSet

This is the main class storing the information about your corpus such as: number of documents, size of vocabulary set, etc. You also can load the mini-batches data to implement your learning algorithm.

```
tmlib.datasets.DataSet(data_path=None, batch_size=None, passes=1, shuffle_every=None,␣
↪vocab_file=None)
```

Parameters

- **data_path**: string,

  Path of file input (corpus)

- **batch_size**: int

  size of mini-batch in each sampling from corpus.

- **passes**: int, default: 1

  passes controls how often we train the model on the entire corpus. Another word for passes might be “epochs” (in training neural network). iterations is somewhat technical, but essentially it controls how often we repeat a particular loop over each document. It is important to set the number of “passes” and “iterations” high enough.

  For example, if you set passes = 5, assume that batch_size = 100 and size of corpus is 10000 then number of training iterations is 10000/100*5 = 5000

- **shuffle_every**: int,

  This parameter help us shuffle the samples (documents) in corpus at each pass (epoch)

  If you set shuffle_every=2, it means after passing over corpus 2 times, corpus will be shuffled

- **vocab_file**: string, default: None

  File vocabulary of corpus

  If corpus is raw text format, file vocabulary is non-necessary. Otherwise, if corpus is tf or sq format, user must set it

Attributes

- **batch_size**: int
- **vocab_file**: string,
- **num_docs**: int,

  Return number of document in corpus

- **data_path**: string,

  path of file corpus which is the term-frequency or term-sequence format

- **data_format**: attribute of class `DataFormat`

  The class `DataFormat` stores name of formats data: DataFormat.RAW_TEXT, DataFormat.TERM_SEQUENCE or DataFormat.TERM_FREQUENCY

- **output_format**: attribute of class `DataFormat`, default is DataFormat.TERM_FREQUENCY

  format of mini-batch. User change the format by use method *set_output_format*

- **passes**: int
- **shuffle_every**: int
- **work_path**: string

  This path is different from data_path. If corpus is shuffled then work_path is path of the shuffled file, not the original file

Methods

- __init__(*data_path=None, batch_size=None, passes=1, shuffle_every=None, vocab_file=None*)
- **load_mini_batch** ()

  loading a mini-batch from corpus with specific format (controlled by **output_format**) Return: object class `Corpus` storing mini-batch

- **load_new_document** (path_file, vocab_file=None)

  You can load new document from `path_file`. If format of file is raw text, you need add `vocab_file` Return: object `Corpus`

- **check_end_of_data** ()

  To check out that whether we visit to the last mini-batch or not.

  Return True if the last mini-batch is loaded and the training is done

- **set_output_format** (output_format)

  set format for the loaded mini-batch

  - **Parameters**: output_format (DataFormat.TERM_SEQUENCE or DataFormat.TERM_FREQUENCY)

- **get_total_docs** ()

  Return number of documents which have been analyzed until the present

- **get_num_tokens** ()

  Return number of tokens in corpus

- **get_num_terms** ()

  Return number of unique terms in corpus (size of vocabulary set)

Example

- Load mini-batch with term-frequency format

```
from tmlib.datasets import DataSet

#AP corpus in folder examples/ap/data
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=4, shuffle_
↪every=2)
minibatch = data.load_mini_batch()  # The format is term-frequency by default
```

- Load mini-batch with term-sequence format

```
from tmlib.datasets import DataSet
from tmlib.datasets.utilities import DataFormat

#AP corpus in folder examples/ap/data
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=4, shuffle_
↪every=2)
data.set_output_format(DataFormat.TERM_SEQUENCE)
minibatch = data.load_mini_batch()
```

In these examples, we set **passes=4** and **shuffle_every=2**, it means: 4 times of passing over data and after every 2 times, corpus is shuffled again. Assume that size of corpus is 5000 documents, batch_size = 100, then number of iterators is: 5000/100*4 = 2000. We can check the last iterator by using method *check_end_of_data()*.

5.7.2 class DataFormat

This is class which contains 3 data-format types of library is: raw text, term_sequence, term-frequency

```
tmlib.datasets.utilities.DataFormat
```

Static Attributes

- **RAW_TEXT**: string, value is 'txt'
- **TERM_FREQUENCY**: string, value is 'tf'
- **TERM_SEQUENCE**: string, value is 'sq'

Example

This example allows checking data format for: corpus *examples/ap/ap_train_raw.txt*

```
from tmlib.datasets.utilities import DataFormat, check_input_format

input_format = check_input_format('examples/ap/ap_train_raw.txt')
print(input_format)
if input_format == DataFormat.RAW_TEXT:
    print('Corpus is raw text')
```

(continues on next page)

(continued from previous page)

```
elif input_format == DataFormat.TERM_SEQUENCE:
    print('Corpus is term-sequence format')
else:
    print(Corpus is term-frequency format')
```

Output:

```
txt
Corpus is raw text
```

5.7.3 class Corpus

This class is used to store the corpus with 2 formats: term-frequency and term-sequence

```
tmlib.datasets.utilities.Corpus(format_type)
```

Parameters

- **format_type**: DataFormat.TERM_SEQUENCE or DataFormat.TERM_FREQUENCY

Attributes

- **format_type**: format of corpus
- **word_ids_tks**: list of list,

  Each element in this list is a list which include the words of a document in corpus (words is unique terms if format is term-frequency and is list of tokens if format is term-sequence)

- **cts_lens**: list

  if format is term-frequency, each element in list is a list frequency of unique terms in respectly document of corpus. If format is term-sequence, each element in list is the number of tokens in document (number of tokens in each doc).

Methods

- **append_doc** (ids_tks, cts_lens)

  Add a document to corpus. If format of this document is term-frequency, this method will append list of unique terms to **word_ids_tks** and append list of frequency to **cts_lens**. If format is term-sequence, the list of tokens and number of tokens will be appended respectly

  - **Parameters**: ids_tks and cts_lens is format (tf or sq) of added document

    ids_tks: list of unique terms (term-frequency format) or list of tokens (term-sequence format) **cts_lens**: list of frequency of unique terms (term-frequency format) or number tokens in document (term-sequence format)

5.7.4 Utility functions

These functions below are in module `tmlib.datasets.utilities`

get_data_home

```
tmlib.datasets.utilities.get_data_home(data_home=None)
```

This folder is used by some large dataset loaders to avoid downloading the data several times.

By default the data dir is set to a folder named 'tmlib_data' in the user home folder. We can change it by change value of data_home parameter The '~' symbol is expanded to the user home folder.

If the folder does not already exist, it is automatically created.

- **Return**: path of the tmlib data dir.

```
>>> from tmlib.datasets import utilities
>>> print 100.get_data_home()
/home/kde/tmlib_data
```

clear_data_home

```
tmlib.datasets.utilities.clear_data_home(data_home=None)
```

Delete all the content of the data home cache.

check_input_format

```
tmlib.datasets.utilities.check_input_format(file_path)
```

- Check format of input file(text formatted or raw text)
- **Parameters**: file_path (string)

  Path of file input
- **Return**: format of input (DataFormat.RAW_TEXT, DataFormat.TERM_FREQUENCY or DataFormat.TERM_SEQUENCE)

```
>>> from tmlib.datasets import utilities
>>> file_path = '/home/kde/Desktop/topicmodel-lib/examples/ap/ap_train.txt'
>>> print utilities.check_input_format(file_path)
tf
>>> file_path = '/home/kde/Desktop/topicmodel-lib/examples/ap/ap_train_raw.txt'
>>> print utilities.check_input_format(file_path)
txt
```

load_batch_raw_text

```
tmlib.datasets.utilities.load_batch_raw_text(file_raw_text_path)
```

- load all of documents and store as a list. Each element in this list is a document with raw text format (string)
- **Parameters**: file_raw_text_path (string)

  Path of file input
- **Return**: list, each element in list is string type and also is text of a document

```
>>> from tmlib.datasets import utilities
>>> path_file_raw_text = '/home/kde/Desktop/topicmodel-lib/examples/ap/ap_infer_raw.
↪txt'
>>> list_docs = utilities.load_batch_raw_text(path_file_raw_text)
>>> print 'number of documents: ', len(list_docs)
number of documents:  50
>>> print list_docs[8]
 Here is a summary of developments in forest and brush fires in Western states:
```

pre_process

```
tmlib.datasets.utilities.pre_process(file_path)
```

- Preprocessing for file input if format of data is raw text
- **Paremeter**: file_path (string)

  Path of file input
- **Return**: list which respectly includes path of vocabulary file, term-frequency file, term-sequence file after preprocessing

```
>>> from tmlib.datasets import utilities
>>> path_file = '/home/kde/Desktop/topicmodel-lib/examples/ap/ap_train_raw.txt'
>>> path_vocab, path_tf, path_sq = utilities.pre_process(path_file)
Waiting...
>>> print 'path to file vocabulary extracted: ', path_vocab
path to file vocabulary extracted:  /home/kde/tmlib_data/ap_train_raw/vocab.txt
>>> print 'path to file with term-frequency format: ', path_tf
path to file with term-frequency format:  /home/kde/tmlib_data/ap_train_raw/ap_train_
↪raw.tf
>>> print 'path to file with term-sequence format: ', path_sq
path to file with term-sequence format:  /home/kde/tmlib_data/ap_train_raw/ap_train_
↪raw.sq
```

load_batch_formatted_from_file

```
tmlib.datasets.utilities.load_batch_formatted_from_file(data_path, output_
↪format=DataFormat.TERM_FREQUENCY)
```

- load all of documents in file which is formatted as term-frequency format or term-sequence format and return a corpus with format is **output_format**
- **Parameters**:
  - **data_path**: path of file data input which is formatted
  - **output_format**: format data of output, default: term-frequence format
- **Return**: object corpus which is the data input for learning

```
>>> path_file_tf = '/home/kde/Desktop/topicmodel-lib/examples/ap/ap_train.txt'
>>> corpus_tf = utilities.load_batch_formatted_from_file(path_file_tf)
>>> print 'Unique terms in the 9th documents: ', corpus_tf.word_ids_tks[8]
Unique terms in the 9th documents:  [5829 4040 2891   14 1783  381 2693]
>>> print 'Frequency of unique terms in the 9th documents: ', corpus_tf.cts_lens[8]
```

(continues on next page)

(continued from previous page)

```
Frequency of unique terms in the 9th documents:  [1 1 1 1 1 1 1]
>>> corpus_sq = utilities.load_batch_formatted_from_file(path_file_tf, output_
↪format=utilities.DataFormat.TERM_SEQUENCE)
>>> print 'List of tokens in the 9th documents: ', corpus_sq.word_ids_tks[8]
List of tokens in the 9th documents:  [5829 4040 2891   14 1783  381 2693]
>>> print 'Number of tokens in the 9th document: ', corpus_sq.cts_lens[8]
Number of tokens in the 9th document:  7
```

reformat_file_to_term_sequence

```
tmlib.datasets.utilities.reformat_file_to_term_sequence(file_path)
```

- convert the formatted file input (tf or sq) to file with format term-sequence
- **Parameter**: file_path (string)

  Path of file input

- **Return**: path of file which is formatted to term-sequence

```
>>> from tmlib.datasets import utilities
>>> path_file_tf = tmlib
>>> path_file_sq = utilities.reformat_file_to_term_sequence(path_file_tf)
>>> print 'path to file term-sequence: ', path_file_sq
path to file term-sequence:  /home/kde/tmlib_data/ap_train/ap_train.sq
```

reformat_file_to_term_frequency

```
tmlib.datasets.utilities.reformat_file_to_term_sequence(file_path)
```

- convert the formatted file input (tf or sq) to file with format term-frequency
- **Parameter**: file_path (string)

  Path of file input

- **Return**: path of file which is formatted to term-frequency

```
>>> from tmlib.datasets import utilities
>>> path_file = '/home/kde/Desktop/topicmodel-lib/examples/ap/ap_train.txt'
>>> path_file_tf = utilities.reformat_file_to_term_sequence(path_file)
>>> print 'path to file term-frequency: ', path_file_tf
path to file term-frequency:  /home/kde/tmlib_data/ap_train/ap_train.tf
```

convert_corpus_format

```
tmlib.datasets.utilities.convert_corpus_format(corpus, data_format)
```

- convert corpus (object of class `Corpus`) to desired format
- **Parameters**:
  - **corpus**: object of class Corpus,

  - **data_format**: format type desired (DataFormat.TERM_SEQUENCE or DataFormat.TERM_FREQUENCY)

- **Return**: object corpus with desired format

```
>>> from tmlib.datasets import utilities
>>> path_file_tf = '/home/kde/Desktop/topicmodel-lib/examples/ap/ap_train.txt'
>>> corpus = utilities.load_batch_formatted_from_file(path_file_tf)
>>> corpus_sq = utilities.convert_corpus_format(corpus, utilities.DataFormat.TERM_
↪SEQUENCE)
>>> print 'Unique terms in the 22th documents: ', corpus.word_ids_tks[21]
Unique terms in the 22th documents:  [  32  396  246   87  824 3259  316  285]
>>> print 'Frequency of unique terms in the 22th documents: ', corpus.cts_lens[21]
Frequency of unique terms in the 22th documents:  [1 1 1 2 1 1 2 1]
>>> print 'List of tokens in the 22th documents: ', corpus_sq.word_ids_tks[21]
List of tokens in the 22th documents:  [32, 396, 246, 87, 87, 824, 3259, 316, 316,␣
↪285]
>>> print 'Number of tokens in the 22th document: ', corpus_sq.cts_lens[21]
Number of tokens in the 22th document:  10
```

compute_sparsity

```
tmlib.datasets.utilities.compute_sparsity(doc_tp, num_docs, num_topics, _type)
```

- Compute document sparsity.
- **Parameters**:
  - **doc_tp**: numpy.array, 2-dimention, the estimated topic mixtures of all documents in corpus
  - **num_docs**: int, the number of documents in corpus
  - **num_topics**: int, is the number of requested latent topics to be extracted from the training corpus.
  - **_type**: string, if the value is 'z', the topic mixtures is estimated by the sampling method as CGS or CVB0, so we have the individual caculation for this. Otherwise, if the value of it isn't 'z', this is for the methods as: VB, OPE or FW
- **Return**: float, sparsity of documents

```
>>> import numpy as np
>>> from tmlib.datasets import utilities
>>> theta = np.array([[0.1, 0.3, 0.2, 0.2, 0.1, 0.1], [0.02, 0.05, 0.03, 0.5, 0.2, 0.
↪2]], dtype='float32')
>>> utilities.compute_sparsity(theta, theta.shape[0], theta.shape[1], _type='t')
1.0
```

write_topic_proportions

tmlib.datasets.utilities.write_topic_proportions(theta, file_name)

- save topic mixtures (theta) to a file
- **Parameters**:
  - **theta**: numpy.array, 2-dimention
  - **file_name**: name (path) of file which is written

5.8 Online VB

Online VB stand for Online Variational Bayes which is proposed by Hoffman, 2010[1]. The learning problem of LDA is to estimate full joint distribution **P** (**z**, θ, β | C) given a corpus C. This problem is intractable and to sovle this, VB[2] approximate that distribution by a distribution Q

$$Q(z, \theta, \beta) = \prod_{d \in C} Q(z_d|\phi_d) \prod_{d \in C} Q(\theta_d|\gamma_d) \prod_k Q(\beta_k|\lambda_k)$$

(k is index of topic)

and now, the learning problem is reduced to estimation the variational parameters {ϕ, γ, λ}

The Online VB using *stochastic variational inference* includes 2 steps:

- Inference for each document in corpus C to find out ϕ_d, γ_d
- Update global variable λ by online fashion

5.8.1 class OnlineVB

```
tmlib.lda.OnlineVB (data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0,␣
↪kappa=0.9, conv_infer=0.0001, iter_infer=50, lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **alpha**: float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topic proportions θ

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **tau0** (τ_0): float, default: 1.0

  In the update λ step, a parameter used is step-size ρ (it is similar to the learning rate in gradient descent optimization). The step-size changes after each training iteration t

  $$\rho_t = (t + \tau_0)^{-\kappa}$$

  And in this, the *delay* tau0 (τ_0) >= 0 down-weights early iterations

- **kappa** (κ): float, default: 0.9

  kappa (κ) $\in$ (0.5, 1] is the forgetting rate which controls how quickly old information is forgotten

[1] M.D. Hoffman, D.M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," The Journal of Machine Learning Research, vol. 14, no. 1, pp. 1303–1347, 2013.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," Journal of Machine Learning Research, vol. 3, no. 3, pp. 993–1022, 2003.

- **conv_infer**: float, default: 0.0001

  The relative improvement of the lower bound on likelihood of VB inference. If If bound hasn't changed much, the inference will be stopped

- **iter_infer**: int, default: 50.

  number of iterations to do inference step

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **alpha**: float,
- **eta** (η): float,
- **tau0** (τ_0): float,
- **kappa** (κ): float,
- **conv_infer**: float,
- **iter_infer**: int,
- **lda_model**: object of class LdaModel
- **_Elogbeta**: float,

  This is expectation of random variable β (topics of model).

- **_expElogbeta**: float, this is equal exp(**_Elogbeta**)

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.9, conv_infer=0.0001, iter_infer=50, lda_model=None*)
- **static_online** (*wordids, wordcts*)

  Excute the learning algorithm, includes: inference for individual document and update λ. 2 parameters *wordids*, *wordcts* represent for term-frequency data of mini-batch. It is the value of 2 attribute **word_ids_tks** and **cts_lens** in class Corpus

  Return: tuple (time of E-step, time of M-step, gamma). gamma (γ) is variational parameter of θ

- **e_step** (*wordids, wordcts*)

  Do inference for indivial document (E-step)

  Return: tuple (gamma, sstats), where, sstats is the sufficient statistics for the M-step

- **update_lambda** (*batch_size, sstats*)

  Update λ by stochastic way.

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

  - **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
  - **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
  - **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
  - **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
  - **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
  - **model_folder**: string, default: None. The place which model file, statistics file are saved.
  - **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

  Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return γ

5.8.2 Example

```
from tmlib.lda import OnlineVB
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-online-vb'
onl_vb = OnlineVB(data=data, num_topics=20, alpha=0.2)
model = onl_vb.learn_model(save_model_every=1, compute_sparsity_every=1,␣
↪save_statistic=True, save_top_words_every=1, num_top_words=10, model_
↪folder='models-online-vb')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
gamma = onl_vb.infer_new_docs(new_corpus)
```

5.9 Online CVB0

CVB0[2] is derived from CVB[1] (Collapsed Variational Bayes), it's an improved version of CVB. Similar to VB, it applies the variational inference to estimate the latent variables. But instead of estimating both θ and **z**, CVB0 actually only estimates the topic assignment **z** . The approximation of distribution **P** (**z**, θ, β | C) as follow:

$$Q(z, \theta, \beta) = Q(\theta, \beta | z, \gamma, \lambda) * \prod_{d \in C} Q(z_d | \phi_d)$$

Online CVB0[3] is an online version of CVB0. It'll infer to the variational parameter ϕ and update the global variable by a stochastic algorithm. The global variable here is a statistic N^{β} (*topic statistic*) updated from ϕ. This statistic plays a similar role with λ in VB and we can estimate topics from this statistic. To estimate the topic proportions, Online CVB0 also used the other statistic is N^{θ} called *document statistic*, and it also is updated from ϕ in a stochastic way.

5.9.1 class OnlineCVB0

```
tmlib.lda.OnlineCVB0(data=None, num_topics=100, alpha=0.01, eta=0.01, tau_phi=1.0,␣
↪kappa_phi=0.9, s_phi=1.0, tau_theta=10.0, kappa_theta=0.9, s_theta=1.0, burn_in=25,␣
↪lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **alpha**: float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topic proportions θ

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **tau_phi** : float, default: 1.0

  In the update global variable step, a parameter used is step-size ρ (it is similar to the learning rate in gradient descent optimization). The step-size changes after each training iteration t

$$\rho_t = s * (t + \tau_0)^{-\kappa}$$

2

A. Asuncion, M. Welling, P. Smyth, and Y. Teh, "On smoothing and inference for topic models," in Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, 2009, pp. 27–34

1

Y. Teh, D. Newman, and M. Welling, "A collapsed variational bayesian inference algorithm for latent dirichlet allocation," in Advances in Neural Information Processing Systems, vol. 19, 2007, p.1353.

3

J. Foulds, L. Boyles, C. DuBois, P. Smyth, and M. Welling, "Stochastic collapsed variational bayesian inference for latent dirichlet allocation," in Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2013, pp. 446–454.

The learning parameters s, τ_0 and κ can be changed manually and called the step-size schedule. But there exist some constrain: $\tau_0 \geq 0$ and $\kappa \in (0.5, 1]$.

The step-size schedule (s_phi, tau_phi, kappa_phi) is used for update N^{β} and (s_theta, tau_theta, kappa_theta) used for update N^{θ}.

- **kappa_phi** : float, default: 0.9
- **s_phi**: float, default: 1.0
- **tau_theta**: float, default: 10.0
- **kappa_theta**: float, default: 0.9
- **s_theta**: float, default: 1.0
- **burn_in**: int, default: 25

  Online CVB0 needs to perform a small number of extra passes per document to learn the document statistics before updating the topic statistics. And here, the parameter burn-in is number of passes we use

- **lda_model**: object of class LdaModel, default: None.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **data**: object `DataSet`
- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **alpha**: float,
- **eta** (η): float,
- **tau_phi**: float,
- **kappa_phi**: float,
- **s_phi**: float,
- **tau_theta**: float,
- **kappa_theta**: float,
- **s_theta**: float,
- **burn_in**: int,
- **lda_model**: object of class LdaModel

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, tau_phi=1.0, kappa_phi=0.9, s_phi=1.0, tau_theta=10.0, kappa_theta=0.9, s_theta=1.0, burn_in=25, lda_model=None*)
- **static_online** (*wordtks, lengths*)

  Execute the Online CVB0. 2 parameters *wordtks*, *lengths* represent for term-sequence data of mini-batch. It is the value of 2 attribute **word_ids_tks** and **cts_lens** in class Corpus

  Return: tuple (time of E-step, time of M-step, N_theta).

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

  - **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
  - **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
  - **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
  - **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
  - **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
  - **model_folder**: string, default: None. The place which model file, statistics file are saved.
  - **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

  Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return the document statistics N^{θ} in new corpus

5.9.2 Example

```
from tmlib.lda import OnlineCVB0
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-online-cvb0'
onl_cvb0 = OnlineCVB0(data=data, num_topics=20, alpha=0.2)
model = onl_cvb0.learn_model(save_model_every=1, compute_sparsity_every=1,␣
↪save_statistic=True, save_top_words_every=1, model_folder='models-online-
↪cvb0')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
N_theta = onl_cvb0.infer_new_docs(new_corpus)
```

5.10 Online CGS

Originally, Collapsed Gibbs Sampling (CGS) was proposed by[1] for learning LDA from data. It recently has been successfully adapted to posterior inference for individual documents by[2]. It tries to estimate **P** (z|d, α, η) by iteratively resampling the topic indicator at each token in document d from the conditional distribution over that position given the remaining topic indicator variables (z^i):

$$P(z_i = k|z^{-i}) \propto (\alpha + \sum_{t \neq i} I(z_t = k)) * exp[\psi(\lambda_{kz_i} - \psi(\sum_t \lambda_{kt})].$$

Note that this adaptation makes the inference more local, i.e., posterior inference for a document does not need to modify any global variable. This property is similar with VB, but very different with CVB and CVB0

Online CGS[2] includes: inference to find out topic indicator (z) at each token in document and update global variable (variational parameter λ) after that.

5.10.1 class OnlineCGS

```
tmlib.lda.OnlineCGS(data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0,␣
↪kappa=0.9, burn_in=25, samples=25, lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **alpha**: float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topic proportions θ

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **tau0** (τ_0): float, default: 1.0

  In the update λ step, a parameter used is step-size ρ (it is similar to the learning rate in gradient descent optimization). The step-size changes after each training iteration t

  $$\rho_t = (t + \tau_0)^{-\kappa}$$

  And in this, the *delay* tau0 (τ_0) >= 0 down-weights early iterations

- **kappa** (κ): float, default: 0.9

  kappa (κ) $\in$ (0.5, 1] is the forgetting rate which controls how quickly old information is forgotten

[1] T. Griffiths and M. Steyvers, "Finding scientific topics," Proceedings of the National Academy of Sciences of the United States of America, vol.101, no. Suppl 1, p. 5228, 2004.

[2] D. Mimno, M. D. Hoffman, and D. M. Blei, "Sparse stochastic inference for latent dirichlet allocation," in Proceedings of the 29th Annual International Conference on Machine Learning, 2012.

- **burn_in**: int, default: 25

  Topic indicator at each token in indivisual document is sampled many times. But at the first several iterations, the samples will be discarded. The parameter **burn_in** is number of the first iterations that we discard the samples

- **samples**: int, default: 25

  After burn-in sweeps, we begin saving sampled topic indicators and we have saved S samples $z^{1,\ldots,S}$ (by default, S = 25)

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **alpha**: float,
- **eta** (η): float,
- **tau0** (τ_0): float,
- **kappa** (κ): float,
- **burn_in**: int,
- **samples**: int,
- **lda_model**: object of class LdaModel
- **_Elogbeta**: float,

  This is expectation of random variable β (topics of model).

- **_expElogbeta**: float, this is equal exp(**_Elogbeta**)

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.9, burn_in=25, samples=25, lda_model=None*)
- **static_online** (*wordtks, lengths*)

  Excute the learning algorithm, includes: inference for individual document and update λ. 2 parameters *wordtks*, *lengths* represent for term-sequence data of mini-batch. It is the value of 2 attribute **word_ids_tks** and **cts_lens** in class Corpus

  Return: tuple (time of E-step, time of M-step, statistic_theta). statistic_theta is a statistic estimated from sampled topic indicators $z^{1,\ldots,S}$. It plays a similar role with γ in VB

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

- **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
- **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
- **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
- **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
- **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
- **model_folder**: string, default: None. The place which model file, statistics file are saved.
- **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return a statistic which used for estimating topic proportions θ

5.10.2 Example

```
from tmlib.lda import OnlineCGS
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-online-cgs'
onl_cgs = OnlineCGS(data=data, num_topics=20, alpha=0.2)
model = onl_cgs.learn_model(save_model_every=1, compute_sparsity_every=1,␣
↪save_statistic=True, save_top_words_every=1, num_top_words=10, model_
↪folder='models-online-cgs')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
statistic_theta = onl_cgs.infer_new_docs(new_corpus)
```

5.11 Online FW

Inference FW[1] estimates directly topic proportions θ for a document by maximum a posteriori (MAP)

Given document d and model $\{\beta, \alpha\}$

$$\theta^* = argmax_{\theta \in \Delta_K} P(\theta, d|\beta, \alpha)$$

[1] Khoat Than, Tu Bao Ho, "Inference in topic models: sparsity and trade-off". [Online]. Available: https://arxiv.org/abs/1512.03300

where θ is a vector K-dimention (K is number of topics) and $\theta \in \Delta_K$, it means:

$$\begin{cases} \theta_k > 0, \quad k = 1, ..., K \\ \theta_1 + \theta_2 + ... + \theta_K = 1 \end{cases}$$

With some assumptions, we can convert the MAP problem above to the concave maximazation over simplex. And here, the Frank-Wolfe algorithm[2] is applied

After finding out θ for each document, we update the global parameter (λ) by online scheme

5.11.1 class OnlineFW

```
tmlib.lda.OnlineFW(data=None, num_topics=100, eta=0.01, tau0=1.0, kappa=0.9, iter_
↪infer=50, lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **tau0** (τ_0): float, default: 1.0

  In the update λ step, a parameter used is step-size ρ (it is similar to the learning rate in gradient descent optimization). The step-size changes after each training iteration t

  $$\rho_t = (t + \tau_0)^{-\kappa}$$

  And in this, the *delay* tau0 (τ_0) >= 0 down-weights early iterations

- **kappa** (κ): float, default: 0.9

  kappa (κ) $\in$ (0.5, 1] is the forgetting rate which controls how quickly old information is forgotten

- **iter_infer**: int, default: 50.

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

[2] K. L. Clarkson, "Coresets, sparse greedy approximation, and the frank-wolfe algorithm," ACM Trans. Algorithms, vol. 6, pp. 63:1–63:30, 2010. [Online]. Available: http://doi.acm.org/10.1145/1824777.1824783

Attributes

- **num_docs**: int,

  Number of documents in the corpus.

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **eta** (η): float,
- **tau0** (τ_0): float,
- **kappa** (κ): float,
- **INF_MAX_ITER**: int,

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.9, iter_infer=50, lda_model=None*)
- **static_online** (*wordids, wordcts*)

  First does an E step on the mini-batch given in wordids and wordcts, then uses the result of that E step to update the topics in M step.

  Parameters:

  - **wordids**: A list whose each element is an array (terms), corresponding to a document. Each element of the array is index of a unique term, which appears in the document, in the vocabulary.
  - **wordcts**: A list whose each element is an array (frequency), corresponding to a document. Each element of the array says how many time the corresponding term in wordids appears in the document.

  Return: tuple (time of E-step, time of M-step, theta): time the E and M steps have taken and the list of topic mixtures of all documents in the mini-batch.

- **e_step** (*wordids, wordcts*)

  Does e step

  Note that, FW can provides sparse solution (theta:topic mixture) when doing inference for each documents. It means that the theta have few non-zero elements whose indexes are stored in list of lists 'index'.

  Return: tuple (theta, index): topic mixtures and their nonzero elements' indexes of all documents in the mini-batch.

- **m_step** (*wordids, wordcts, theta, index*)

  Does M-step

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

- **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
- **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
- **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
- **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
- **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
- **model_folder**: string, default: None. The place which model file, statistics file are saved.
- **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return topic proportions θ for each document in new corpus

5.11.2 Example

```
from tmlib.lda import OnlineFW
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-online-fw'
onl_fw = OnlineFW(data=data, num_topics=20)
model = onl_fw.learn_model(save_model_every=1, compute_sparsity_every=1,␣
↪save_statistic=True, save_top_words_every=1, num_top_words=10, model_
↪folder='models-online-fw')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
theta = onl_fw.infer_new_docs(new_corpus)
```

5.12 Online OPE

Similar to FW, OPE [1] is a inference method allowing us estimate directly topic proportions θ for individual document. The problem of posterior inference for each document d, given a model $\{\beta, \alpha\}$, is to estimate the full joint distribution **P** (z_d, θ_d, d | β, α). Direct estimation of this distribution is intractable. Hence existing approaches uses different schemes. VB, CVB0 try to estimate the distribution by maximizing a lower bound of the likelihood **P** (d | β, α), whereas CGS tries to estimate **P** (z_d | d, β, α).

OPE will estimate θ by maximize the posterior distribution **P** (θ, d | β, α):

$$\theta^* = argmax_{\theta \in \Delta_K} P(\theta, d|\beta, \alpha)$$

where θ is a vector K-dimention (K is number of topics) and $\theta \in \Delta_K$, it means:

$$\begin{cases} \theta_k > 0, \quad k = 1, ..., K \\ \theta_1 + \theta_2 + ... + \theta_K = 1 \end{cases}$$

The objective function sounds like FW, but OPE hasn't a constraint about α like FW and so, the optimization algorithm of OPE is also different from FW

The update λ (variational parameter of β) is designed followed by online scheme

5.12.1 class OnlineOPE

```
tmlib.lda.OnlineOPE(data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0,␣
↪kappa=0.9, iter_infer=50, lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **alpha**: float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topic proportions θ

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **tau0** (τ_0): float, default: 1.0

  In the update λ step, a parameter used is step-size ρ (it is similar to the learning rate in gradient descent optimization). The step-size changes after each training iteration t

  $$\rho_t = (t + \tau_0)^{-\kappa}$$

  And in this, the *delay* tau0 (τ_0) >= 0 down-weights early iterations

- **kappa** (κ): float, default: 0.9

  kappa (κ) $\in$ (0.5, 1] is the forgetting rate which controls how quickly old information is forgotten

- **iter_infer**: int, default: 50.

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_docs**: int,

  Number of documents in the corpus.

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **alpha** (α): float,
- **eta** (η): float,
- **tau0** (τ_0): float,
- **kappa** (κ): float,
- **INF_MAX_ITER**: int,

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.9, iter_infer=50, lda_model=None*)
- **static_online** (*wordids, wordcts*)

  First does an E step on the mini-batch given in wordids and wordcts, then uses the result of that E step to update the topics in M step.

  Parameters:

  - **wordids**: A list whose each element is an array (terms), corresponding to a document. Each element of the array is index of a unique term, which appears in the document, in the vocabulary.
  - **wordcts**: A list whose each element is an array (frequency), corresponding to a document. Each element of the array says how many time the corresponding term in wordids appears in the document.

  Return: tuple (time of E-step, time of M-step, theta): time the E and M steps have taken and the list of topic mixtures of all documents in the mini-batch.

- **e_step** (*wordids, wordcts*)

  Does e step

  Return: Returns topic mixtures theta.

- **m_step** (*wordids, wordcts, theta*)

  Does M-step

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

  - **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, ..., model will is save into a file. If setting default, model won't be saved.

- **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
- **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
- **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
- **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
- **model_folder**: string, default: None. The place which model file, statistics file are saved.
- **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return topic proportions θ for each document in new corpus

5.12.2 Example

```
from tmlib.lda import OnlineOPE
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-online-ope'
onl_ope = OnlineOPE(data=data, num_topics=20, alpha=0.2)
model = streaming_ope.learn_model(save_model_every=1, compute_sparsity_
↪every=1, save_statistic=True, save_top_words_every=1, num_top_words=10,␣
↪model_folder='models-online-ope')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
theta = onl_ope.infer_new_docs(new_corpus)
```

[1] Khoat Than, Tung Doan, "Guaranteed inference in topic models". [Online]. Available at: https://arxiv.org/abs/1512.03308

5.13 Streaming VB

Similar to Online VB, Streaming VB uses the inference VB[1] for individual document to find out the local variables γ (variational parameter of topic proportions θ) and ϕ (variational parameter of topic indicators **z**). But, the update

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," Journal of Machine Learning Research, vol. 3, no. 3, pp. 993–1022, 2003.

global variable λ (variational pamameter of β) is adapted to the stream environments. With the streaming learning, we don't need to know the number of documents in Corpus.

For more detail, you can see in[2]

We also make a simulation for the stream evironment with the articles from Wikipedia website. See simulation

5.13.1 class StreamingVB

```
tmlib.lda.StreamingVB(data=None, num_topics=100, alpha=0.01, eta=0.01, conv_infer=0.
↪0001, iter_infer=50, lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **alpha**: float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topic proportions θ

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **conv_infer**: float, default: 0.0001

  The relative improvement of the lower bound on likelihood of VB inference. If If bound hasn't changed much, the inference will be stopped

- **iter_infer**: int, default: 50.

  number of iterations to do inference step

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **alpha**: float,
- **eta** (η): float,
- **conv_infer**: float,
- **iter_infer**: int,
- **lda_model**: object of class LdaModel

[2] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael Jordan. Streaming variational bayes. In Advances in Neural Information Processing Systems, pages 1727{1735, 2013.

- **_Elogbeta**: float,

  This is expectation of random variable β (topics of model).

- **_expElogbeta**: float, this is equal exp(**_Elogbeta**)

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.9, conv_infer=0.0001, iter_infer=50, lda_model=None*)
- **static_online** (*wordids, wordcts*)

  Excute the learning algorithm, includes: inference for individual document and update λ. 2 parameters *wordids*, *wordcts* represent for term-frequency data of mini-batch. It is the value of 2 attribute **word_ids_tks** and **cts_lens** in class Corpus

  Return: tuple (time of E-step, time of M-step, gamma). gamma (γ) is variational parameter of θ

- **e_step** (*wordids, wordcts*)

  Do inference for indivial document (E-step)

  Return: tuple (gamma, sstats), where, sstats is the sufficient statistics for the M-step

- **update_lambda** (*batch_size, sstats*)

  Update λ by stochastic way.

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

  - **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
  - **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
  - **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
  - **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
  - **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
  - **model_folder**: string, default: None. The place which model file, statistics file are saved.
  - **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

  Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return γ

5.13.2 Example

```
from tmlib.lda import StreamingVB
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-streaming-vb'
streaming_vb = StreamingVB(data=data, num_topics=20, alpha=0.2)
model = streaming_vb.learn_model(save_model_every=1, compute_sparsity_
↪every=1, save_statistic=True, save_top_words_every=1, num_top_words=10,␣
↪model_folder='models-streaming-vb')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
gamma = streaming_vb.infer_new_docs(new_corpus)
```

5.14 Streaming FW

Similar to Online FW, Streaming FW uses the inference FW[1] for individual document to find out the local variables θ (topic proportions). But, the update global variable λ (variational pamameter of β) is adapted to the stream environments. With the streaming learning, we don't need to know the number of documents in Corpus.

For more detail, you can see in[1]

We also make a simulation for the stream evironment with the articles from Wikipedia website. See simulation

5.14.1 class StreamingFW

```
tmlib.lda.StreamingFW(data=None, num_topics=100, eta=0.01, iter_infer=50, lda_
↪model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **iter_infer**: int, default: 50.

  Number of iterations of FW algorithm to do inference step

[1] Khoat Than, Tu Bao Ho, "Inference in topic models: sparsity and trade-off". [Online]. Available: https://arxiv.org/abs/1512.03300

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_terms**: int,

  size of the vocabulary set of the training corpus
- **num_topics**: int,
- **eta** (η): float,
- **iter_infer**: int,

  Number of iterations of FW algorithm to do inference step
- **lda_model**: object of class `LdaModel`

Methods

- __init__ (*data=None, num_topics=100, eta=0.01, iter_infer=50, lda_model=None*)
- **static_online** (*wordids, wordcts*)

  First does an E step on the mini-batch given in wordids and wordcts, then uses the result of that E step to update the topics in M step.

  Parameters:

  - **wordids**: A list whose each element is an array (terms), corresponding to a document. Each element of the array is index of a unique term, which appears in the document, in the vocabulary.
  - **wordcts**: A list whose each element is an array (frequency), corresponding to a document. Each element of the array says how many time the corresponding term in wordids appears in the document.

  Return: tuple (time of E-step, time of M-step, theta): time the E and M steps have taken and the list of topic mixtures of all documents in the mini-batch.
- **e_step** (*wordids, wordcts*)

  Does e step

  Note that, FW can provides sparse solution (theta:topic mixture) when doing inference for each documents. It means that the theta have few non-zero elements whose indexes are stored in list of lists 'index'.

  Return: tuple (theta, index): topic mixtures and their nonzero elements' indexes of all documents in the mini-batch.
- **m_step** (*wordids, wordcts, theta, index*)

  Does M-step
- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

  - **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.

- **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
- **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
- **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
- **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
- **model_folder**: string, default: None. The place which model file, statistics file are saved.
- **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return topic proportions θ for each document in new corpus

5.14.2 Example

```
from tmlib.lda import StreamingFW
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-streaming-fw'
streaming_fw = StreamingFW(data=data, num_topics=20)
model = streaming_fw.learn_model(save_model_every=1, compute_sparsity_
↪every=1, save_statistic=True, save_top_words_every=1, num_top_words=10,␣
↪model_folder='models-streaming-fw')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
theta = streaming_fw.infer_new_docs(new_corpus)
```

[2] K. L. Clarkson, "Coresets, sparse greedy approximation, and the frank-wolfe algorithm," ACM Trans. Algorithms, vol. 6, pp. 63:1–63:30, 2010. [Online]. Available: http://doi.acm.org/10.1145/1824777.1824783

5.15 Streaming OPE

Similar to Online OPE, Streaming OPE uses the inference OPE[1] for individual document to find out the local variables θ (topic proportions). But, the update global variable λ (variational pamameter of β) is adapted to the stream environments. With the streaming learning, we don't need to know the number of documents in Corpus.

For more detail, you can see in[1]

[1] Khoat Than, Tung Doan, "Guaranteed inference in topic models". [Online]. Available at: https://arxiv.org/abs/1512.03308

We also make a simulation for the stream evironment with the articles from Wikipedia website. See simulation

5.15.1 class StreamingOPE

```
tmlib.lda.StreamingOPE(data=None, num_topics=100, alpha=0.01, eta=0.01, iter_infer=50,
↪ lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **alpha**: float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topic proportions θ

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **iter_infer**: int, default: 50.

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,

- **alpha** (α): float,

- **eta** (η): float,

- **INF_MAX_ITER**: int,

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, iter_infer=50, lda_model=None*)

- **static_online** (*wordids, wordcts*)

  First does an E step on the mini-batch given in wordids and wordcts, then uses the result of that E step to update the topics in M step.

  Parameters:

  - **wordids**: A list whose each element is an array (terms), corresponding to a document. Each element of the array is index of a unique term, which appears in the document, in the vocabulary.
  - **wordcts**: A list whose each element is an array (frequency), corresponding to a document. Each element of the array says how many time the corresponding term in wordids appears in the document.

  Return: tuple (time of E-step, time of M-step, theta): time the E and M steps have taken and the list of topic mixtures of all documents in the mini-batch.

- **e_step** (*wordids, wordcts*)

  Does e step

  Return: Returns topic mixtures theta.

- **m_step** (*wordids, wordcts, theta*)

  Does M-step

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

  - **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
  - **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
  - **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
  - **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
  - **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
  - **model_folder**: string, default: None. The place which model file, statistics file are saved.
  - **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

  Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return topic proportions θ for each document in new corpus

5.15.2 Example

```
from tmlib.lda import StreamingOPE
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-streaming-ope'
streaming_ope = StreamingOPE(data=data, num_topics=20, alpha=0.2)
model = streaming_ope.learn_model(save_model_every=1, compute_sparsity_
↪every=1, save_statistic=True, save_top_words_every=1, num_top_words=10,␣
↪model_folder='models-streaming-ope')
```

(continues on next page)

(continued from previous page)

```
# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
theta = streaming_ope.infer_new_docs(new_corpus)
```

5.16 ML-CGS

You can see that Online CGS is also a hybrid algorithm. It infers to topic indicators **z** at each token in individual document by Gibb sampling. After that, it defines a approximatee sufficient statistics to update global variable λ. By borrowing idea from ML-FW and ML-OPE, ML-CGS will estimate directly topics β instead of λ

First, ML-CGS will estimate θ from S sampled topic indicators $z^{1,2,...,S}$ in each mini-batch[1]

And then, we can define a sufficient statistics $\hat{\beta}$ to update β following[2]

5.16.1 class tmlib.lda.MLCGS

```
tmlib.lda.MLCGS(data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.9,␣
↪burn_in=25, samples=25, lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **alpha**: float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topic proportions θ

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **tau0** (τ_0): float, default: 1.0

[1] D. Mimno, M. D. Hoffman, and D. M. Blei, "Sparse stochastic inference for latent dirichlet allocation," in Proceedings of the 29th Annual International Conference on Machine Learning, 2012.

[2] K. Than and T. B. Ho, "Fully sparse topic models," in Machine Learning and Knowledge Discovery in Databases, ser. Lecture Notes in Computer Science, P. Flach, T. De Bie, and N. Cristianini, Eds. Springer, 2012, vol. 7523, pp. 490–505.

In the update λ step, a parameter used is step-size ρ (it is similar to the learning rate in gradient descent optimization). The step-size changes after each training iteration t

$$\rho_t = (t + \tau_0)^{-\kappa}$$

And in this, the *delay* tau0 (τ_0) >= 0 down-weights early iterations

- **kappa** (κ): float, default: 0.9

  kappa (κ) $\in$ (0.5, 1] is the forgetting rate which controls how quickly old information is forgotten

- **burn_in**: int, default: 25

  Topic indicator at each token in indivisual document is sampled many times. But at the first several iterations, the samples will be discarded. The parameter **burn_in** is number of the first iterations that we discard the samples

- **samples**: int, default: 25

  After burn-in sweeps, we begin saving sampled topic indicators and we have saved S samples $z^{1,\ldots,S}$ (by default, S = 25)

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **alpha**: float,
- **eta** (η): float,
- **tau0** (τ_0): float,
- **kappa** (κ): float,
- **burn_in**: int,
- **samples**: int,
- **lda_model**: object of class LdaModel

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.9, burn_in=25, samples=25, lda_model=None*)
- **static_online** (*wordtks, lengths*)

  Excute the learning algorithm, includes: inference for individual document and update λ. 2 parameters *wordtks*, *lengths* represent for term-sequence data of mini-batch. It is the value of 2 attribute **word_ids_tks** and **cts_lens** in class Corpus

  Return: tuple (time of E-step, time of M-step, statistic_theta). statistic_theta is a statistic estimated from sampled topic indicators $z^{1,\ldots,S}$. It plays a similar role with γ in VB

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

  - **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
  - **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
  - **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
  - **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
  - **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
  - **model_folder**: string, default: None. The place which model file, statistics file are saved.
  - **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

  Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return a statistic which used for estimating topic proportions θ

5.16.2 Example

```
from tmlib.lda import MLCGS
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-ml-cgs'
ml_cgs = MLCGS(data=data, num_topics=20, alpha=0.2)
model = ml_cgs.learn_model(save_model_every=1, compute_sparsity_every=1,␣
↪save_statistic=True, save_top_words_every=1, num_top_words=10, model_
↪folder='models-ml-cgs')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
statistic_theta = ml_cgs.infer_new_docs(new_corpus)
```

5.17 ML-FW

If you've read Online FW and Streaming FW, you can see that they are hybrid algorithms which combine OPE inference with variational Bayes for estimating the posterior of the global variables. They have to maintain variational parameters (λ) for the Dirichlet distribution over topics, instead of the topics themselve. Nonetheless, the combinations are not very natural since we have to compute ϕ (variational parameter of topic indicators **z**) from topic proportions θ in order to update the model. Such a conversion might incur some information losses

It is more natural if we can use directly θ in the update of the model at each minibatch. To this end, we use an idea from[1]. Instead of following Bayesian approach to estimate the distribution over topics (P(β | λ)), one can consider the topics as parameters and estimate them directly from data. It means, we can **estimate directly** β. This is the idea of ML-FW for learning LDA

One more advance of ML-FW is which enables us enables us to learn LDA from either large corpora or data streams (both online or stream environment)

For more detail ML-FW, see in[2]

5.17.1 class tmlib.lda.MLFW

```
tmlib.lda.MLFW(data=None, num_topics=100, tau0=1.0, kappa=0.9, iter_infer=50, lda_
↪model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **tau0** (τ_0): float, default: 1.0

  In the update λ step, a parameter used is step-size ρ (it is similar to the learning rate in gradient descent optimization). The step-size changes after each training iteration t

  $$\rho_t = (t + \tau_0)^{-\kappa}$$

  And in this, the *delay* tau0 (τ_0) >= 0 down-weights early iterations

- **kappa** (κ): float, default: 0.9

  kappa (κ) $\in$ (0.5, 1] is the forgetting rate which controls how quickly old information is forgotten

- **iter_infer**: int, default: 50.

  Number of iterations of FW algorithm to do inference step

[1] K. Than and T. B. Ho, "Fully sparse topic models," in Machine Learning and Knowledge Discovery in Databases, ser. Lecture Notes in Computer Science, P. Flach, T. De Bie, and N. Cristianini, Eds. Springer, 2012, vol. 7523, pp. 490–505.

[2] Khoat Than, Tu Bao Ho, "Inference in topic models: sparsity and trade-off". [Online]. Available: https://arxiv.org/abs/1512.03300

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_docs**: int,

  Number of documents in the corpus.

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **eta** (η): float,
- **tau0** (τ_0): float,
- **kappa** (κ): float,
- **iter_infer**: int,

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`

Methods

- __init__ (*data=None, num_topics=100, tau0=1.0, kappa=0.9, iter_infer=50, lda_model=None*)
- **static_online** (*wordids, wordcts*)

  First does an E step on the mini-batch given in wordids and wordcts, then uses the result of that E step to update the topics in M step.

  Parameters:

  - **wordids**: A list whose each element is an array (terms), corresponding to a document. Each element of the array is index of a unique term, which appears in the document, in the vocabulary.
  - **wordcts**: A list whose each element is an array (frequency), corresponding to a document. Each element of the array says how many time the corresponding term in wordids appears in the document.

  Return: tuple (time of E-step, time of M-step, theta): time the E and M steps have taken and the list of topic mixtures of all documents in the mini-batch.

- **e_step** (*wordids, wordcts*)

  Does e step

  Note that, FW can provides sparse solution (theta:topic mixture) when doing inference for each documents. It means that the theta have few non-zero elements whose indexes are stored in list of lists 'index'.

  Return: tuple (theta, index): topic mixtures and their nonzero elements' indexes of all documents in the mini-batch.

- **m_step** (*wordids, wordcts, theta, index*)

  Does M-step

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

  Parameters:

  - **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
  - **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
  - **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
  - **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
  - **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
  - **model_folder**: string, default: None. The place which model file, statistics file are saved.
  - **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

  Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return topic proportions θ for each document in new corpus

5.17.2 Example

```
from tmlib.lda import MLFW
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-ml-fw'
ml_fw = MLFW(data=data, num_topics=20)
model = ml_fw.learn_model(save_model_every=1, compute_sparsity_every=1, save_
↪statistic=True, save_top_words_every=1, num_top_words=10, model_folder=
↪'models-ml-fw')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
theta = ml_fw.infer_new_docs(new_corpus)
```

[3] K. L. Clarkson, "Coresets, sparse greedy approximation, and the frank-wolfe algorithm," ACM Trans. Algorithms, vol. 6, pp. 63:1–63:30, 2010. [Online]. Available: http://doi.acm.org/10.1145/1824777.1824783

5.18 ML-OPE

If you've read Online OPE and Streaming OPE, you can see that they are hybrid algorithms which combine OPE inference with variational Bayes for estimating the posterior of the global variables. They have to maintain variational parameters (λ) for the Dirichlet distribution over topics, instead of the topics themselve. Nonetheless, the combinations are not very natural since we have to compute ϕ (variational parameter of topic indicators **z**) from topic proportions θ in order to update the model. Such a conversion might incur some information losses

It is more natural if we can use directly θ in the update of the model at each minibatch. To this end, we use an idea from[1]. Instead of following Bayesian approach to estimate the distribution over topics (P(β | λ)), one can consider the topics as parameters and estimate them directly from data. It means, we can **estimate directly** β. This is the idea of ML-OPE for learning LDA

One more advance of ML-OPE is which enables us enables us to learn LDA from either large corpora or data streams (both online or stream environment)

For more detail ML-OPE, see in[2]

5.18.1 class tmlib.lda.MLOPE

```
tmlib.lda.MLOPE(data=None, num_topics=100, alpha=0.01, tau0=1.0, kappa=0.9, iter_
↪infer=50, lda_model=None)
```

Parameters

- **data**: object `DataSet`

  object used for loading mini-batches data to analyze

- **num_topics**: int, default: 100

  number of topics of model.

- **alpha**: float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topic proportions θ

- **eta** (η): float, default: 0.01

  hyperparameter of model LDA that affect sparsity of topics β

- **tau0** (τ_0): float, default: 1.0

  In the update λ step, a parameter used is step-size ρ (it is similar to the learning rate in gradient descent optimization). The step-size changes after each training iteration t

  $$\rho_t = (t + \tau_0)^{-\kappa}$$

  And in this, the *delay* tau0 (τ_0) >= 0 down-weights early iterations

- **kappa** (κ): float, default: 0.9

  kappa (κ) $\in$ (0.5, 1] is the forgetting rate which controls how quickly old information is forgotten

[1] K. Than and T. B. Ho, "Fully sparse topic models," in Machine Learning and Knowledge Discovery in Databases, ser. Lecture Notes in Computer Science, P. Flach, T. De Bie, and N. Cristianini, Eds. Springer, 2012, vol. 7523, pp. 490–505.

[2] Khoat Than, Tung Doan, "Guaranteed inference in topic models". [Online]. Available at: https://arxiv.org/abs/1512.03308

- **iter_infer**: int, default: 50.

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`.

  If this is None value, a new object `LdaModel` will be created. If not, it will be the model learned previously

Attributes

- **num_terms**: int,

  size of the vocabulary set of the training corpus

- **num_topics**: int,
- **alpha** (α): float,
- **eta** (η): float,
- **tau0** (τ_0): float,
- **kappa** (κ): float,
- **INF_MAX_ITER**: int,

  Number of iterations of FW algorithm to do inference step

- **lda_model**: object of class `LdaModel`

Methods

- __init__ (*data=None, num_topics=100, alpha=0.01, eta=0.01, tau0=1.0, kappa=0.9, iter_infer=50, lda_model=None*)
- **static_online** (*wordids, wordcts*)

  First does an E step on the mini-batch given in wordids and wordcts, then uses the result of that E step to update the topics in M step.

  Parameters:

  - **wordids**: A list whose each element is an array (terms), corresponding to a document. Each element of the array is index of a unique term, which appears in the document, in the vocabulary.
  - **wordcts**: A list whose each element is an array (frequency), corresponding to a document. Each element of the array says how many time the corresponding term in wordids appears in the document.

  Return: tuple (time of E-step, time of M-step, theta): time the E and M steps have taken and the list of topic mixtures of all documents in the mini-batch.

- **e_step** (*wordids, wordcts*)

  Does e step

  Return: Returns topic mixtures theta.

- **m_step** (*wordids, wordcts, theta*)

  Does M-step

- **learn_model** (*save_model_every=0, compute_sparsity_every=0, save_statistic=False, save_top_words_every=0, num_top_words=10, model_folder=None, save_topic_proportions=None*)

  This used for learning model and to save model, statistics of model.

Parameters:

- **save_model_every**: int, default: 0. If it is set to 2, it means at iterators: 0, 2, 4, 6, . . . , model will is save into a file. If setting default, model won't be saved.
- **compute_sparsity_every**: int, default: 0. Compute sparsity and store in attribute **statistics**. The word "every" here means as same as **save_model_every**
- **save_statistic**: boolean, default: False. Saving statistics or not. The statistics here is the time of E-step, time of M-step, sparsity of document in corpus
- **save_top_words_every**: int, default: 0. Used for saving top words of topics (highest probability). Number words displayed is **num_top_words** parameter.
- **num_top_words**: int, default: 20. By default, the number of words displayed is 10.
- **model_folder**: string, default: None. The place which model file, statistics file are saved.
- **save_topic_proportions**: string, default: None. This used to save topic proportions θ of each document in training corpus. The value of it is path of file `.h5`

Return: the learned model (object of class LdaModel)

- **infer_new_docs** (*new_corpus*)

  This used to do inference for new documents. **new_corpus** is object `Corpus`. This method return topic proportions θ for each document in new corpus

5.18.2 Example

```
from tmlib.lda import MLOPE
from tmlib.datasets import DataSet

# data preparation
data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5,␣
↪shuffle_every=2)
# learning and save the model, statistics in folder 'models-ml-ope'
ml_ope = MLOPE(data=data, num_topics=20, alpha=0.2)
model = streaming_ope.learn_model(save_model_every=1, compute_sparsity_
↪every=1, save_statistic=True, save_top_words_every=1, num_top_words=10,␣
↪model_folder='models-ml-ope')


# inference for new documents
vocab_file = data.vocab_file
# create object ``Corpus`` to store new documents
new_corpus = data.load_new_documents('data/ap_infer_raw.txt', vocab_
↪file=vocab_file)
theta = ml_ope.infer_new_docs(new_corpus)
```

5.19 Preprocessing

This work will be implemented when data format of corpus is raw text. Topic models take documents that contain words as input. We still have to determine what "words" we're going to use and how to extract them from the format raw text. Recall that most topic models treat documents as a bag-of-words, so we can stop caring about the order of the tokens within the text and concentrate on how many times a particular word appears in the text. So, we need to

convert the raw format to term-sequence or term-frequency as mentioned in the quick start section. To understand in detail about technique of preprocessing, please read preprocessing[1] document.

File raw text also need a specific format type so that we can recognize it. The format of file as follow:

- Corpus includes many documents, all of that are saved into a file.
- Each document is represented as follow

```
<DOC>
# maybe is title or others or nothing
<TEXT>
# Plain text
</TEXT>
</DOC>
```

Please refer[1] to know more detail about preprocessing technique

5.19.1 class PreProcessing

```
tmlib.preprocessing.PreProcessing(file_path, stemmed=False, remove_rare_word=3,␣
↪remove_common_word=None)
```

Parameters

- **file_path**: string, not default

  Path of file corpus which has raw text format.

- **stemmed**: boolean, default: False

  Appply the stemming algorithm (Porter, 1980) to preprocess text data. The algorithm is applied when parameter is set value *True*

- **remove_rare_word**: int, default: 3

  Removing rarely words in the documents. Default, words which appeared in less 3 documents will be removed

- **remove_common_word**: int, default: None

  Removing common words (which appeared in many documents). Default, words which appeared in greater a half documents of corpus will be removed.

Attributes

- **path_file_vocab**: string,

  path of the vocabulary file which created after calling method *extract_vocab()*

- **path_file_tf**: string,

  path of file corpus with term-frequency format. This file is created after calling method *save_format_tf()*

- **path_file_sq**: string,

  path of file corpus with term-sequence format. This file is created after calling method *save_format_sq()*

[1] Care and Feeding of Topic Models: Problems, Diagnostics, and Improvements. Jordan Boyd Graber, David Mimno, and David Newman. In Handbook of Mixed Membership Models and Their Applications, CRC/Chapman Hall, 2014.

Methods

- __init__(*file_path, stemmed=False, remove_rare_word=3, remove_common_word=None*)
- **process()**

  run the preprocessing algorithms

- **extract_vocab** (folder=None)

  Extracting to the file vocabulary of corpus after preprocessing

  - **Parameters**: folder (string, default: None)

    The position which file vocabulary is saved. By default, file is saved in a folder with path <*user home folder*> + *"tmlib_data/"* + <*name of file input*>

- **save_format_sq** (folder=None)

  Extracting to the file corpus with term-sequence format

  - **Parameters**: folder (string, default: None)

    The position which file term-sequence is saved. By default, file is saved in a same folder with file vocablary created above

- **save_format_tf** (folder=None)

  Extracting to the file corpus with term-frequency format

  - **Parameters**: folder (string, default: None)

    The position which file term-frequency is saved. By default, file is saved in a same folder with file vocablary created above

5.19.2 Example

This is a tutorial for how to preprocess a file raw text. Using AP raw corpus here

```
from tmlib.preprocessing import PreProcessing

p = PreProcessing('data/ap_train_raw.txt')
p.process()                  # run algorithm of preprocessing step
p.extract_vocab()            # extract to the vocabulary of corpus
p.save_format_sq()           # save the new format is term-sequence format
p.save_format_tf()           # save the format is term-frequency format
# display path of file vocabulary, file term-sequence, file term-frequency
print(object.path_file_vocab, object.path_file_sq, object.path_file_tf)
```

The result files is automatically saved in a folder named "tmlib_data" in the user data home. User can change the position by set value parameters in functions such as extract_vocab(), save_format_sq() or save_format_tf(). User can also change the setting parameters of preprocessing algorithm by set value when create object

5.20 Visualization

Topic modeling is an unsupervised machine learning method that learns the underlying themes in a large collection of otherwise unorganized documents. This discovered structure summarizes and organizes the documents. However, topic models are high-level statistical tools—a user must scrutinize numerical distributions to understand and explore

their results. So, we implemented a module visualization to reveal meaningful patterns in a collection, and helping end-users explore and understand its contents in new ways

You can read more detail about the methodology in[1]

To use this module, make sure that the packages **pandas**, **Tkinter** are installed in your system

```
tmlib.visualization.visualize(model, database_path, data_path, vocab_file)
```

This function will visualize the topics, some documents in a GUI App. And this function takes 4 arguments :

- **model**: string or object class `LdaModel`

  This argument can take 2 values: string if it's a path of file and object of class `LdaModel` you get after learning phase. If it is a path of file, this file is a file saving object `LdaModel`

- **database_path**: string,

  This argument is path of file which stores topic proportions (θ) of documents. You need to save topic proportions in learning phase. Note: this file should be a file `.h5` and it is a database not only stores topic proportion but also stores content of documents

- **data_path**: string,

  Path of file training data (corpus), you need the content of documents to visualize it

- **vocab_file**: string,

  File vocabulary of training corpus

5.20.1 Example

We'll demo with AP corpus. First, you need to learn model from this corpus using Online-OPE. Because you have to save topic proportions - this takes a long time. So, we need a fast learning algorithm

```
from tmlib.lda import OnlineOPE
from tmlib.datasets import DataSet

data = DataSet(data_path='data/ap_train_raw.txt', batch_size=100, passes=5, shuffle_
↪every=2)
onl_ope = OnlineOPE(data=data, num_topics=20, alpha=0.2)
# learn model and save topic proportions in file database 'database.h5'
model = streaming_ope.learn_model(save_topic_proportions='database.h5')
# save object model into file 'object_model.h5'
model.save('object_model.h5')
```

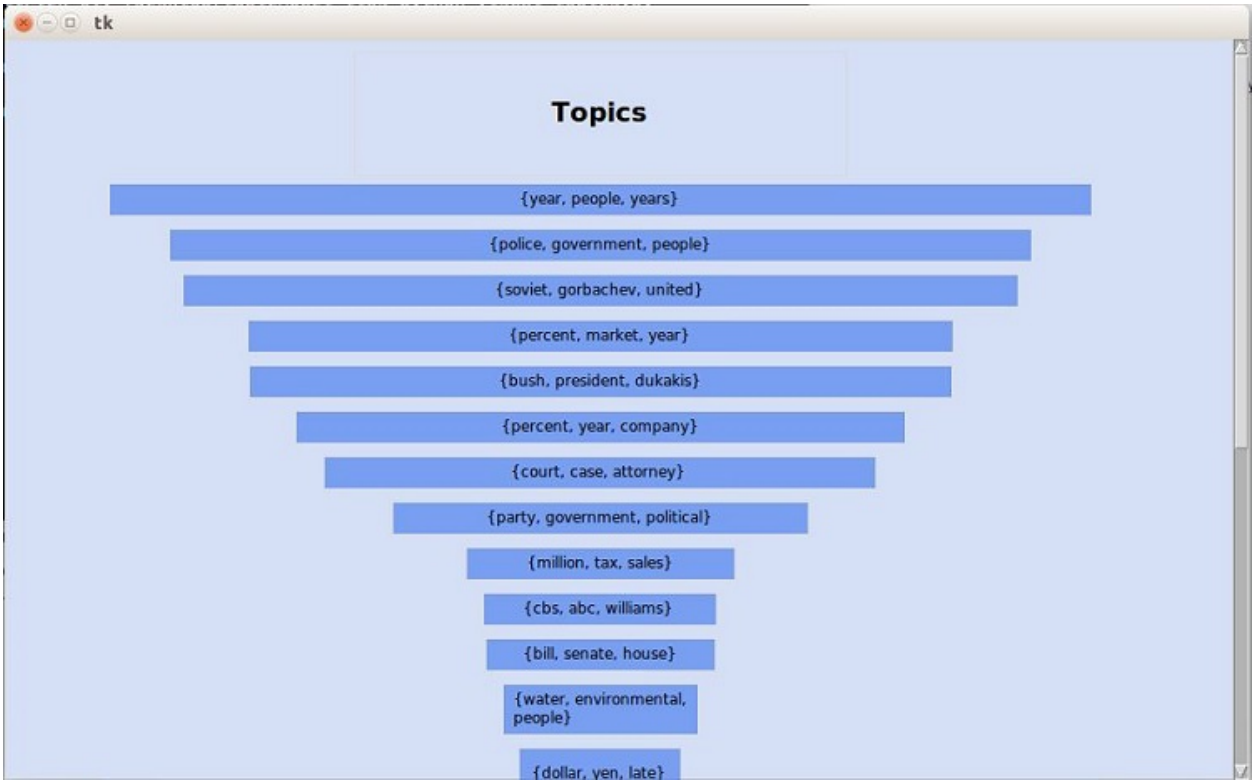
After learning, file vocabulary extracted will be saved into `~/tmlib_data/ap_train_raw/vocab.txt` (you can see by run `print data.vocab_file`). Now, we can visualize the result

```
import os
from tmlib.visualization import visualize

# get the vocabulary file
vocab_file = os.path.expanduser('~/tmlib_data/ap_train_raw/vocab.txt')
visualize('object_model.h5', 'database.h5', 'ap_train_raw.txt', vocab_file)
```

[1] Visualizing Topic Models. Allison J.B. Chaney and David M. Blei. Department of Computer Science. Princeton University, Princeton, NJ USA.

And this is the result you achieve



After you click on topic {years, people, years}

words	related documents	related topics
year	Dennis Day, the Irish tenor of radio and televisi...	{cbs, abc, williams}
people	A man stabbed his 8-year-old daughter to death an...	{court, case, attorney}
years	Today is Sunday, Oct. 9, the 283rd day of 1988. T...	{percent, year, company}
mrs	Superstitions generally cast Friday the 13th into...	{bill, senate, house}
police	At least 40 people suffered broken bones or heat ...	{campaign, state, northern}
time	EDITOR'S NOTE _ Since her one-woman show, ``Witho...	{police, government, people}
day	Pope John Paul II heard confessions from 11 pilgr...	{water, environmental, people}
family	Do all things get cheaper by the dozen _ even chi...	{soviet, gorbachev, united}
state	Just weeks ago, they were troublemakers, working ...	{bush, president, dukakis}
women	Here is a list of Irene Dunne's films. ``Cimarron...	{party, government, political}
life	The mayor handed out a harsh etiquette lesson to ...	{meese, museum, disney}
wife	The following are the most popular videocassettes...	{school, movie, film}
home	Actress Anne Ramsey, whose grotesque character in...	{million, tax, sales}
man	The following are the most popular videocassettes...	{trade, takeshita, deconcini}
prison	Television watchers of the 19th annual New York C...	{percent, market, year}
hospital	``Are you going to die?'' asks Charlie Brown when...	{dollar, yen, late}
children	In some ways, imprisoned Vietnam vets do differ f...	{students, computer, farmers}

Click on the document with short content 'Do all things get cheaper by dozen_even. . . '