bitermplus

Maksim Terpilowski

Apr 18, 2024

USAGE

Bitermplus implements Biterm topic model for short texts introduced by Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. Actually, it is a cythonized version of BTM. This package is also capable of computing *perplexity* and *semantic coherence* metrics.

CHAPTER

ONE

SETUP

1.1 Linux and Windows

There should be no issues with installing *bitermplus* under these OSes. You can install the package directly from PyPi.

```
pip install bitermplus
```

Or from this repo:

```
pip install git+https://github.com/maximtrp/bitermplus.git
```

1.2 Mac OS

First, you need to install XCode CLT and Homebrew. Then, install `libomp` using `brew`:

```
xcode-select --install
brew install libomp
pip3 install bitermplus
```

1.3 Requirements

- cython
- numpy
- pandas
- scipy
- scikit-learn
- tqdm

CHAPTER

TWO

TUTORIAL

2.1 Model fitting

Here is a simple example of model fitting. It is supposed that you have already gone through the preprocessing stage: cleaned, lemmatized or stemmed your documents, and removed stop words.

```
import bitermplus as btm
import numpy as np
import pandas as pd

# Importing data
df = pd.read_csv(
    'dataset/SearchSnippets.txt.gz', header=None, names=['texts'])
texts = df['texts'].str.strip().tolist()

# Vectorizing documents, obtaining full vocabulary and biterms
# Internally, btm.get_words_freqs uses CountVectorizer from sklearn
# You can pass any of its arguments to btm.get_words_freqs
# For example, you can remove stop words:
stop_words = ["word1", "word2", "word3"]
X, vocabulary, vocab_dict = btm.get_words_freqs(texts, stop_words=stop_words)
docs_vec = btm.get_vectorized_docs(texts, vocabulary)
biterms = btm.get_biterms(docs_vec)

# Initializing and running model
model = btm.BTM(
    X, vocabulary, seed=12321, T=8, M=20, alpha=50/8, beta=0.01)
model.fit(biterms, iterations=20)
```

2.2 Inference

Now, we will calculate documents vs topics probability matrix (make an inference).

```
p_zd = model.transform(docs_vec)
```

If you need to make an inference on a new dataset, you should vectorize it using your vocabulary from the training set:

```
new_docs_vec = btm.get_vectorized_docs(new_texts, vocabulary)
p_zd = model.transform(new_docs_vec)
```

2.3 Calculating metrics

To calculate perplexity, we must provide documents vs topics probability matrix (p_zd) that we calculated at the previous step.

```
perplexity = btm.perplexity(model.matrix_topics_words_, p_zd, X, 8)
coherence = btm.coherence(model.matrix_topics_words_, X, M=20)
# or
perplexity = model.perplexity_
coherence = model.coherence_
```

2.4 Visualizing results

For results visualization, we will use tmplot package.

```
import tmplot as tmp

# Run the interactive report interface
tmp.report(model=model, docs=texts)
```

2.5 Filtering stable topics

Unsupervised topic models (such as LDA) are subject to topic instability[1][2][3]. There is a special method in tmplot package for selecting stable topics. It uses various distance metrics such as Kullback-Leibler divergence (symmetric and non-symmetric), Hellinger distance, Jeffrey's divergence, Jensen-Shannon divergence, Jaccard index, Bhattacharyya distance, Total variation distance.

```
import pickle as pkl
import tmplot as tmp
import glob

# Loading saved models
models_files = sorted(glob.glob(r'results/model[0-9].pkl'))
models = []
for fn in models_files:
    file = open(fn, 'rb')
    models.append(pkl.load(file))
    file.close()

# Choosing reference model
np.random.seed(122334)
reference_model = np.random.randint(1, 6)
```

(continues on next page)

[1] Koltcov, S., Koltsova, O., & Nikolenko, S. (2014, June). Latent dirichlet allocation: stability and applications to studies of user-generated content. In Proceedings of the 2014 ACM conference on Web science (pp. 161-165).

[2] Mantyla, M. V., Claes, M., & Farooq, U. (2018, October). Measuring LDA topic stability from clusters of replicated runs. In Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement (pp. 1-4).

[3] Greene, D., O'Callaghan, D., & Cunningham, P. (2014, September). How many topics? stability analysis for topic models. In Joint European conference on machine learning and knowledge discovery in databases (pp. 498-513). Springer, Berlin, Heidelberg.

(continued from previous page)

```
# Getting close topics
close_topics, close_kl = tmp.get_closest_topics(
    models, method="sklb", ref=reference_model)

# Getting stable topics
stable_topics, stable_kl = tmp.get_stable_topics(
    close_topics, close_kl, ref=reference_model, thres=0.7)

# Stable topics indices list
print(stable_topics[:, reference_model])
```

2.6 Model loading and saving

Support for model serializing with pickle was implemented in v0.5.3. Here is how you can save and load a model:

```
import pickle as pkl
# Saving
with open("model.pkl", "wb") as file:
    pkl.dump(model, file)

# Loading
with open("model.pkl", "rb") as file:
    model = pkl.load(file)
```

2.7 References

CHAPTER

THREE

BENCHMARKS

In this section, the results of a series of benchmarks done on *SearchSnippets* dataset are presented. Sixteen models were trained with different iterations number (from 10 to 2000) and default model parameters. Topics number was set to 8. Semantic topic coherence (`u_mass`) and perplexity were calculated for each model.

CHAPTER

FOUR

MODEL

class `bitermplus.BTM`(*n_dw*, *vocabulary*, *int T*, *int M=20*, *double alpha=1.*, *double beta=0.01*, *unsigned int seed=0*, *int win=15*, *bool has_background=False*)

Biterm Topic Model.

Parameters

- **n_dw** (*csr.csr_matrix*) – Documents vs words frequency matrix. Typically, it should be the output of *CountVectorizer* from sklearn package.
- **vocabulary** (*list*) – Vocabulary (a list of words).
- **T** (*int*) – Number of topics.
- **M** (*int = 20*) – Number of top words for coherence calculation.
- **alpha** (*float = 1*) – Model parameter.
- **beta** (*float = 0.01*) – Model parameter.
- **seed** (*int = 0*) – Random state seed. If seed is equal to 0 (default), use `time(NULL)`.
- **win** (*int = 15*) – Biterms generation window.
- **has_background** (*bool = False*) – Use a background topic to accumulate highly frequent words.

`alpha_`

float Model parameter.

Type
BTM.alpha_

`beta_`

float Model parameter.

Type
BTM.beta_

`biterms_`

np.ndarray Model biterms. Terms are coded with the corresponding ids.

Type
BTM.biterms_

`coherence_`

np.ndarray Semantic topics coherence.

Type
BTM.coherence_

coherence_window_

int Number of top words for coherence calculation.

Type
BTM.coherence_window_

df_words_topics_

DataFrame Words vs topics probabilities in a DataFrame.

Type
BTM.df_words_topics_

fit(*self*, *list Bs*, *int iterations=600*, *bool verbose=True*)

Biterm topic model fitting method.

Parameters

- **Bs** (`list`) – Biterms list.
- **iterations** (`int = 600`) – Iterations number.
- **verbose** (`bool = True`) – Show progress bar.

fit_transform(*self*, *docs*, *list biterms*, *unicode infer_type=u'sum_b'*, *int iterations=600*, *bool verbose=True*)

Run model fitting and return documents vs topics matrix.

Parameters

- **docs** (`list`) – Documents list. Each document must be presented as a list of words ids. Typically, it can be the output of `bitermplus.get_vectorized_docs()`.
- **biterms** (`list`) – List of biterms.
- **infer_type** (`str`) – Inference type. The following options are available:

1) `sum_b` (default).
2) `sum_w`.
3) `mix`.

- **iterations** (`int = 600`) – Iterations number.
- **verbose** (`bool = True`) – Be verbose (show progress bars).

Returns
p_zd – Documents vs topics matrix (D x T).

Return type
np.ndarray

has_background_

bool Specifies whether the model has a background topic

to accumulate highly frequent words.

Type
BTM.has_background_

iterations_

int Number of iterations the model fitting process has

gone through.

Type
BTM.iterations_

`labels_`
np.ndarray Model document labels (most probable topic for each document).

Type
BTM.labels_

`matrix_docs_topics_`
np.ndarray Documents vs topics probabilities matrix.

Type
BTM.matrix_docs_topics_

`matrix_topics_docs_`
np.ndarray Topics vs documents probabilities matrix.

Type
BTM.matrix_topics_docs_

`matrix_topics_words_`
np.ndarray Topics vs words probabilities matrix.

Type
BTM.matrix_topics_words_

`matrix_words_topics_`
np.ndarray Words vs topics probabilities matrix.

Type
BTM.matrix_words_topics_

`perplexity_`
float Perplexity.

Run *transform* method before calculating perplexity

Type
BTM.perplexity_

`theta_`
np.ndarray Topics probabilities vector.

Type
BTM.theta_

`topics_num_`
int Number of topics.

Type
BTM.topics_num_

`transform`(*self*, *list docs*, *unicode infer_type=u'sum_b'*, *bool verbose=True*)
Return documents vs topics probability matrix.

Parameters

- **docs** (*list*) – Documents list. Each document must be presented as a list of words ids. Typically, it can be the output of `bitermplus.get_vectorized_docs()`.

- **infer_type** (*str*) – Inference type. The following options are available:

  1) `sum_b` (default).

  2) `sum_w`.

  3) `mix`.

- **verbose** (*bool = True*) – Be verbose (show progress bar).

Returns
p_zd – Documents vs topics probability matrix (D vs T).

Return type
np.ndarray

vocabulary_

np.ndarray Vocabulary (list of words).

Type
BTM.vocabulary_

vocabulary_size_

int Vocabulary size (number of words).

Type
BTM.vocabulary_size_

window_

int Biterms generation window size.

Type
BTM.window_

CHAPTER

FIVE

METRICS

`bitermplus.coherence(double[:, :] p_wz, n_dw, double eps=1., int M=20)`

Semantic topic coherence calculation [1]_.

Parameters

- **p_wz** (*np.ndarray*) – Topics vs words probabilities matrix (T x W).
- **n_dw** (*scipy.sparse.csr_matrix*) – Words frequency matrix for all documents (D x W).
- **eps** (*float*) – Calculation parameter. It is summed with a word pair conditional probability.
- **M** (*int*) – Number of top words in a topic to take.

Returns

coherence – Semantic coherence estimates for all topics.

Return type

np.ndarray

References

Example

```
>>> import bitermplus as btm
>>> # Preprocessing step
>>> # ...
>>> # X, vocabulary, vocab_dict = btm.get_words_freqs(texts)
>>> # Model fitting step
>>> # model = ...
>>> # Coherence calculation
>>> coherence = btm.coherence(model.matrix_topics_words_, X, M=20)
```

`bitermplus.perplexity(double[:, :] p_wz, double[:, :] p_zd, n_dw, long T)` → double

Perplexity calculation [1]_.

Parameters

- **p_wz** (*np.ndarray*) – Topics vs words probabilities matrix (T x W).
- **p_zd** (*np.ndarray*) – Documents vs topics probabilities matrix (D x T).
- **n_dw** (*scipy.sparse.csr_matrix*) – Words frequency matrix for all documents (D x W).
- **T** (*int*) – Number of topics.

Returns
perplexity – Perplexity estimate.

Return type
float

References

Example

```
>>> import bitermplus as btm
>>> # Preprocessing step
>>> # ...
>>> # X, vocabulary, vocab_dict = btm.get_words_freqs(texts)
>>> # Model fitting step
>>> # model = ...
>>> # Inference step
>>> # p_zd = model.transform(docs_vec_subset)
>>> # Coherence calculation
>>> perplexity = btm.perplexity(model.matrix_topics_words_, p_zd, X, 8)
```

`bitermplus.`**`entropy`**(*double[:, :] p_wz*, *bool max_probs=True*)

Renyi entropy calculation routine [1]_.

Renyi entropy can be used to estimate the optimal number of topics: just fit several models with a different number of topics and choose the number of topics for which the Renyi entropy is the least.

Parameters
`p_wz` (*`np.ndarray`*) – Topics vs words probabilities matrix (T x W).

Returns

- **renyi** (*double*) – Renyi entropy value.
- **max_probs** (*bool*) – Use maximum probabilities of terms per topics instead of all probability values.

References

Example

```
>>> import bitermplus as btm
>>> # Preprocessing step
>>> # ...
>>> # Model fitting step
>>> # model = ...
>>> # Entropy calculation
>>> entropy = btm.entropy(model.matrix_topics_words_)
```

CHAPTER SIX

UTILITY FUNCTIONS

bitermplus.**get_words_freqs**(*docs: List[str] | ndarray | Series*, ***kwargs: dict*) → Tuple[csr_matrix, ndarray, Dict]

Compute words vs documents frequency matrix.

Parameters

- **docs** (*Union[List[str], np.ndarray, Series]*) – Documents in any format that can be passed to `sklearn.feature_extraction.text.CountVectorizer()` method.
- **kwargs** (*dict*) – Keyword arguments for `sklearn.feature_extraction.text.CountVectorizer()` method.

Returns

Documents vs words matrix in CSR format, vocabulary as a numpy.ndarray of terms, and vocabulary as a dictionary of {term: id} pairs.

Return type

Tuple[scipy.sparse.csr_matrix, np.ndarray, Dict]

Example

```
>>> import pandas as pd
>>> import bitermplus as btm
```

```
>>> # Loading data
>>> df = pd.read_csv(
...     'dataset/SearchSnippets.txt.gz', header=None, names=['texts'])
>>> texts = df['texts'].str.strip().tolist()
```

```
>>> # Vectorizing documents, obtaining full vocabulary and biterms
>>> X, vocabulary, vocab_dict = btm.get_words_freqs(texts)
```

bitermplus.**get_vectorized_docs**(*docs: List[str] | ndarray*, *vocab: List[str] | ndarray*) → List[ndarray]

Replace words with their ids in each document.

Parameters

- **docs** (*Union[List[str], np.ndarray]*) – Documents (iterable of strings).
- **vocab** (*Union[List[str], np.ndarray]*) – Vocabulary (iterable of terms).

Returns

docs – Vectorised documents (list of `numpy.ndarray` objects with terms ids).

Return type
List[np.ndarray]

Example

```
>>> import pandas as pd
>>> import bitermplus as btm
```

```
>>> # Loading data
>>> df = pd.read_csv(
...     'dataset/SearchSnippets.txt.gz', header=None, names=['texts'])
>>> texts = df['texts'].str.strip().tolist()
```

```
>>> # Vectorizing documents, obtaining full vocabulary and biterms
>>> X, vocabulary, vocab_dict = btm.get_words_freqs(texts)
>>> docs_vec = btm.get_vectorized_docs(texts, vocabulary)
```

`bitermplus.`**`get_biterms`**(*docs: List[ndarray]*, *win: int = 15*) → List[List[int]]

Biterms creation routine.

Parameters

- **docs** (*List[np.ndarray]*) – List of numpy.ndarray objects containing word indices.
- **win** (*int = 15*) – Biterms generation window.

Returns
List of biterms for each document.

Return type
List[List[int]]

Example

```
>>> import pandas as pd
>>> import bitermplus as btm
```

```
>>> # Loading data
>>> df = pd.read_csv(
...     'dataset/SearchSnippets.txt.gz', header=None, names=['texts'])
>>> texts = df['texts'].str.strip().tolist()
```

```
>>> # Vectorizing documents, obtaining full vocabulary and biterms
>>> X, vocabulary, vocab_dict = btm.get_words_freqs(texts)
>>> docs_vec = btm.get_vectorized_docs(texts, vocabulary)
>>> biterms = btm.get_biterms(docs_vec)
```

`bitermplus.`**`get_top_topic_words`**(*model:* BTM, *words_num: int = 20*, *topics_idx: Sequence[Any] = None*) → DataFrame

Select top topic words from a fitted model.

Parameters

- **model** (`bitermplus._btm.BTM`) – Fitted BTM model.

- **words_num** (*int = 20*) – The number of words to select.
- **topics_idx** (*Union[List, numpy.ndarray] = None*) – Topics indices. Meant to be used to select only stable topics.

Returns
Words with highest probabilities per each selected topic.

Return type
DataFrame

Example

```
>>> stable_topics = [0, 3, 10, 12, 18, 21]
>>> top_words = btm.get_top_topic_words(
...     model,
...     words_num=100,
...     topics_idx=stable_topics)
```

bitermplus.**get_top_topic_docs**(*docs: Sequence[Any]*, *p_zd: ndarray*, *docs_num: int = 20*, *topics_idx: Sequence[Any] = None*) → DataFrame

Select top topic docs from a fitted model.

Parameters

- **docs** (*Sequence[Any]*) – Iterable of documents (e.g. list of strings).
- **p_zd** (*np.ndarray*) – Documents vs topics probabilities matrix.
- **docs_num** (*int = 20*) – The number of documents to select.
- **topics_idx** (*Sequence[Any] = None*) – Topics indices. Meant to be used to select only stable topics.

Returns
Documents with highest probabilities in all selected topics.

Return type
DataFrame

Example

```
>>> top_docs = btm.get_top_topic_docs(
...     texts,
...     p_zd,
...     docs_num=100,
...     topics_idx=[1,2,3,4])
```

bitermplus.**get_docs_top_topic**(*docs: Sequence[Any]*, *p_zd: ndarray*) → DataFrame

Select most probable topic for each document.

Parameters

- **docs** (*Sequence[Any]*) – Iterable of documents (e.g. list of strings).
- **p_zd** (*np.ndarray*) – Documents vs topics probabilities matrix.

Returns
Documents and the most probable topic for each of them.

Return type
DataFrame

Example

```
>>> import bitermplus as btm
>>> # Read documents from file
>>> # texts = ...
>>> # Build and train a model
>>> # model = ...
>>> # model.fit(...)
>>> btm.get_docs_top_topic(texts, model.matrix_docs_topics_)
```

INDEX