OCTIS

Public

13 Branches 30 Tags Go to file t Go to file Add file

silviatti Bump version: 1.13.1 → 1.14.0 8f9a0ac · last year 1,150 Commits

.github	Update python-publish.yml	last year
docs	Update conf.py	4 years ago
examples	updating READMEs and col...	4 years ago
octis	Bump version: 1.13.1 → 1.14.0	last year
preprocessed_datasets	FIX rename dataset bbc_ne...	3 years ago
tests	Add test to ensure multipro...	2 years ago
trained_embeddings/...	feat(etm): adds support for ...	4 years ago
.editorconfig	let's start the package re-or...	5 years ago
.gitignore	.idea directory is ignored	5 years ago
AUTHORS.rst	updated readme	4 years ago
CONTRIBUTING.rst	chore(docs): typo	4 years ago
HISTORY.rst	update history	last year
LICENSE	let's start the package re-or...	5 years ago
MANIFEST.in	let's start the package re-or...	5 years ago
Makefile	refactoring	4 years ago
README.rst	Fix typo in README.rst	2 years ago
logo.png	fixing logo and history	4 years ago
requirements.txt	fix scikit-learn req	last year
requirements_dev.txt	update requirements	3 years ago
setup.cfg	Bump version: 1.13.1 → 1.14.0	last year
setup.py	Bump version: 1.13.1 → 1.14.0	last year

About

OCTIS: Comparing Topic Models is Simple! A python package to optimize and evaluate topic models (accepted at EACL2021 demo track)

#nlp #natural-language-processing #hyperparameter-optimization #topic-modeling #nlp-library #bayesian-optimization #hyperparameter-tuning #latent-dirichlet-allocation #evaluation-metrics #neural-topic-models #latent-semantic-analysis #topic-models #hyperparameter-search #non-negative-matrix-factorization #nlproc

Readme
MIT license
Contributing
Activity
Custom properties
793 stars
13 watching
117 forks
Report repository

Releases

30 tags

Packages

No packages published

Contributors 16

README Contributing MIT license

OCTIS : Optimizing and Comparing Topic Models is Simple!

pypi v1.14.0 | Python package passing | docs failing | contributors 15 | License MIT | 793 | downloads/month | Open in Colab

Deployments

languages

- **Python** 58.4%
- **HTML** 19.0%
- **JavaScript** 10.8%
- **Jupyter Notebook** 10.5%
- **Other** 1.3%

OCTIS (Optimizing and Comparing Topic models Is Simple) aims at training, analyzing and comparing Topic Models, whose optimal hyperparameters are estimated by means of a Bayesian Optimization approach. This work has been accepted to the demo track of EACL2021. Click to read the paper!

Table of Contents

Install

You can install OCTIS with the following command:

```
pip install octis
```

You can find the requirements in the requirements.txt file.

Main Features

- Preprocess your own dataset or use one of the already-preprocessed benchmark datasets
- Well-known topic models (both classical and neurals)

- Evaluate your model using different state-of-the-art evaluation metrics
- Optimize the models' hyperparameters for a given metric using Bayesian Optimization
- Python library for advanced usage or simple web dashboard for starting and controlling the optimization experiments

Examples and Tutorials

To easily understand how to use OCTIS, we invite you to try our tutorials out :)

Name	Link
How to build a topic model and evaluate the results (LDA on 20Newsgroups)	
How to optimize the hyperparameters of a neural topic model (CTM on M10)	

Some tutorials on Medium:

Two guides on how to use OCTIS with practical examples:

- A beginner's guide to OCTIS vol. 1 by Emil Rijcken
- A beginner's guide to OCTIS vol. 2 by Emil Rijcken

A tutorial on topic modeling on song lyrics:

- OCTIS - The Future of Topic Modeling by Nicolas Pogeant

Datasets and Preprocessing

Load a preprocessed dataset

To load one of the already preprocessed datasets as follows:

```
from octis.dataset.dataset import Dataset
dataset = Dataset()
dataset.fetch_dataset("20NewsGroup")
```

Just use one of the dataset names listed below. Note: it is case-sensitive!

Available Datasets

Name in OCTIS	Source	# Docs	# Words	# Labels	Language
20NewsGroup	20Newsgroup	16309	1612	20	English
BBC_News	BBC-News	2225	2949	5	English
DBLP	DBLP	54595	1513	4	English
M10	M10	8355	1696	10	English
DBPedia_IT	DBPedia_IT	4251	2047	5	Italian
Europarl_IT	Europarl_IT	3613	2000	NA	Italian

Load a Custom Dataset

Otherwise, you can load a custom preprocessed dataset in the following way:

```
from octis.dataset.dataset import Dataset
dataset = Dataset()
dataset.load_custom_dataset_from_folder("../path/to/the/dataset/folder")
```

Make sure that the dataset is in the following format:

- corpus file: a .tsv file (tab-separated) that contains up to three columns, i.e. the document, the partitition, and the label associated to the document (optional).
- vocabulary: a .txt file where each line represents a word of the vocabulary

The partition can be "train" for the training partition, "test" for testing partition, or "val" for the validation partition. An example of dataset can be found here: sample_dataset.

Disclaimer

Similarly to TensorFlow Datasets and HuggingFace's nlp library, we just downloaded and prepared public datasets. We do not host or distribute these datasets, vouch for their quality or fairness, or claim that you have license to use the dataset. It is your responsibility to determine whether you have permission to use the dataset under the dataset's license and to cite the right owner of the dataset.

If you're a dataset owner and wish to update any part of it, or do not want your dataset to be included in this library, please get in touch through a GitHub issue.

If you're a dataset owner and wish to include your dataset in this library, please get in touch through a GitHub issue.

Preprocess a Dataset

To preprocess a dataset, import the preprocessing class and use the preprocess_dataset method.

```
import os
import string
from octis.preprocessing.preprocessing import Preprocessing
os.chdir(os.path.pardir)

# Initialize preprocessing
preprocessor = Preprocessing(vocabulary=None, max_features=None,
                             remove_punctuation=True, punctuation=string.punctuation,
                             lemmatize=True, stopword_list='english',
                             min_chars=1, min_words_docs=0)
# preprocess
```

```
dataset = preprocessor.preprocess_dataset(documents_path='..\corpus.txt', labels_path='..\labels.txt')

# save the preprocessed dataset
dataset.save('hello_dataset')
```

For more details on the preprocessing see the preprocessing demo example in the examples folder.

Topic Models and Evaluation

Train a model

To build a model, load a preprocessed dataset, set the model hyperparameters and use `train_model()` to train the model.

```
from octis.dataset.dataset import Dataset
from octis.models.LDA import LDA

# Load a dataset
dataset = Dataset()
dataset.load_custom_dataset_from_folder("dataset_folder")

model = LDA(num_topics=25)  # Create model
model_output = model.train_model(dataset) # Train the model
```

If the dataset is partitioned, you can:

- Train the model on the training set and test it on the test documents
- Train the model with the whole dataset, regardless of any partition.

Available Models

Name	Implementation
CTM (Bianchi et al. 2021)	https://github.com/MilaNLProc/contextualized-topic-models
ETM (Dieng et al. 2020)	https://github.com/adjidieng/ETM
HDP (Blei et al. 2004)	https://radimrehurek.com/gensim/
LDA (Blei et al. 2003)	https://radimrehurek.com/gensim/
LSI (Landauer et al. 1998)	https://radimrehurek.com/gensim/
NMF (Lee and Seung 2000)	https://radimrehurek.com/gensim/
NeuralLDA (Srivastava and Sutton 2017)	https://github.com/estebandito22/PyTorchAVITM
ProdLda (Srivastava and Sutton 2017)	https://github.com/estebandito22/PyTorchAVITM

If you use one of these implementations, make sure to cite the right paper.

If you implemented a model and wish to update any part of it, or do not want your model to be included in this library, please get in touch through a GitHub issue.

If you implemented a model and wish to include your model in this library, please get in touch through a GitHub issue. Otherwise, if you want to include the model by yourself, see the following section.

Evaluate a model

To evaluate a model, choose a metric and use the `score()` method of the metric class.

```
from octis.evaluation_metrics.diversity_metrics import TopicDiversity

metric = TopicDiversity(topk=10) # Initialize metric
topic_diversity_score = metric.score(model_output) # Compute score of the metric
```

Available metrics

- **Classification Metrics**:
  - F1-score : `F1Score(dataset)`
  - Precision : `PrecisionScore(dataset)`
  - Recall : `RecallScore(dataset)`

- Accuracy : `AccuracyScore(dataset)`

- **Coherence Metrics**:
  - UMass Coherence : `Coherence(measure='u_mass')`
  - C_V Coherence : `Coherence(measure='c_v')`
  - UCI Coherence : `Coherence(measure='c_uci')`
  - NPMI Coherence : `Coherence(measure='c_npmi')`
  - Word Embedding-based Coherence Pairwise : `WECoherencePairwise()`
  - Word Embedding-based Coherence Centroid : `WECoherenceCentroid()`

- **Diversity Metrics**:
  - Topic Diversity : `TopicDiversity()`
  - InvertedRBO : `InvertedRBO()`
  - Word Embedding-based InvertedRBO Matches : `WordEmbeddingsInvertedRBO()`
  - Word Embedding-based InvertedRBO Centroid : `WordEmbeddingsInvertedRBOCentroid()`
  - Log odds ratio : `LogOddsRatio()`
  - Kullback-Liebler Divergence : `KLDivergence()`

- **Similarity Metrics**:
  - Ranked-Biased Overlap : `RBO()`
  - Word Embedding-based RBO Matches : `WordEmbeddingsRBOMatch()`
  - Word Embedding-based RBO Centroid : `WordEmbeddingsRBOCentroid()`
  - Word Embeddings-based Pairwise Similarity : `WordEmbeddingsPairwiseSimilarity()`
  - Word Embeddings-based Centroid Similarity : `WordEmbeddingsCentroidSimilarity()`
  - Word Embeddings-based Weighted Sum Similarity : `WordEmbeddingsWeightedSumSimilarity()`
  - Pairwise Jaccard Similarity : `PairwiseJaccardSimilarity()`

- **Topic significance Metrics**:
  - KL Uniform : `KL_uniform()`
  - KL Vacuous : `KL_vacuous()`
  - KL Background : `KL_background()`

Implement your own Model

Models inherit from the class AbstractModel defined in octis/models/model.py . To build your own model your class must override the train_model(self, dataset, hyperparameters) method which always requires at least a Dataset object and a Dictionary of hyperparameters as input and should return a dictionary with the output of the model as output.

To better understand how a model work, let's have a look at the LDA implementation. The first step in developing a custom model is to define the dictionary of default hyperparameters values:

```
hyperparameters = {'corpus': None, 'num_topics': 100, 'id2word': None, 'alpha': 'symmetric',
    'eta': None, # ...
    'callbacks': None}
```

Defining the default hyperparameters values allows users to work on a subset of them without having to assign a value to each parameter.

The following step is the train_model() override:

```
def train_model(self, dataset, hyperparameters={}, top_words=10):
```

The LDA method requires a dataset, the hyperparameters dictionary and an extra (optional) argument used to select how many of the most significative words track for each topic.

With the hyperparameters defaults, the ones in input and the dataset you should be able to write your own code and return as output a dictionary with at least 3 entries:

- *topics*: the list of the most significative words foreach topic (list of lists of strings).
- *topic-word-matrix*: an NxV matrix of weights where N is the number of topics and V is the vocabulary length.
- *topic-document-matrix*: an NxD matrix of weights where N is the number of topics and D is the number of documents in the corpus.

If your model supports the training/test partitioning it should also return:

- *test-topic-document-matrix*: the document topic matrix of the test set.

Hyperparameter Optimization

To optimize a model you need to select a dataset, a metric and the search space of the hyperparameters to optimize. For the types of the hyperparameters, we use `scikit-optimize` types (https://scikit-optimize.github.io/stable/modules/space.html)

```
from octis.optimization.optimizer import Optimizer
from skopt.space.space import Real

# Define the search space. To see which hyperparameters to optimize, see the topic model's initialization signatu
search_space = {"alpha": Real(low=0.001, high=5.0), "eta": Real(low=0.001, high=5.0)}

# Initialize an optimizer object and start the optimization.
optimizer=Optimizer()
optResult=optimizer.optimize(model, dataset, eval_metric, search_space, save_path="../results" # path to store th
                             number_of_call=30, # number of optimization iterations
                             model_runs=5) # number of runs of the topic model
#save the results of th optimization in a csv file
optResult.save_to_csv("results.csv")
```

The result will provide best-seen value of the metric with the corresponding hyperparameter configuration, and the hyperparameters and metric value for each iteration of the optimization. To visualize this information, you have to set 'plot' attribute of Bayesian_optimization to True.

You can find more here: optimizer README

Dashboard

OCTIS includes a user friendly graphical interface for creating, monitoring and viewing experiments. Following the implementation standards of datasets, models and metrics the dashboard will automatically update and allow you to use your own custom implementations.

To run the dashboard you need to clone the repo. While in the project directory run the following command:

```
python OCTIS/dashboard/server.py
```

The browser will open and you will be redirected to the dashboard. In the dashboard you can:

- Create new experiments organized in batch
- Visualize and compare all the experiments
- Visualize a custom experiment
- Manage the experiment queue

How to cite our work

This work has been accepted at the demo track of EACL 2021! Click to read the paper! If you decide to use this resource, please cite:

```
@inproceedings{terragni2020octis,
    title={{OCTIS}: Comparing and Optimizing Topic Models is Simple!},
    author={Terragni, Silvia and Fersini, Elisabetta and Galuzzi, Bruno Giovanni and Tropeano, Pietro and Candelieri, A
    year={2021},
    booktitle={Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguist
    month = apr,
    year = "2021",
    publisher = "Association for Computational Linguistics",
    url = "https://www.aclweb.org/anthology/2021.eacl-demos.31",
    pages = "263--270",
}

@inproceedings{DBLP:conf/clic-it/TerragniF21,
  author    = {Silvia Terragni and Elisabetta Fersini},
  editor    = {Elisabetta Fersini and Marco Passarotti and Viviana Patti},
  title     = {{OCTIS 2.0: Optimizing and Comparing Topic Models in Italian Is Even
               Simpler!}},
  booktitle = {Proceedings of the Eighth Italian Conference on Computational Linguistics,
               CLiC-it 2021, Milan, Italy, January 26-28, 2022},
  series    = {{CEUR} Workshop Proceedings},
  volume    = {3033},
  publisher = {CEUR-WS.org},
  year      = {2021},
```

```
  url       = {http://ceur-ws.org/Vol-3033/paper55.pdf},
}
```

Team

Project and Development Lead

- Silvia Terragni <s.terragni4@campus.unimib.it> Follow @TerragniSilvia
- Elisabetta Fersini <elisabetta.fersini@unimib.it> Follow @FersiniE
- Antonio Candelieri <antonio.candelieri@unimib.it>

Current Contributors

- Pietro Tropeano <p.tropeano1@campus.unimib.it> Framework architecture, Preprocessing, Topic Models, Evaluation metrics and Web Dashboard
- Bruno Galuzzi <bruno.galuzzi@unimib.it> Bayesian Optimization
- Silvia Terragni <s.terragni4@campus.unimib.it> Overall project

Past Contributors

- Lorenzo Famiglini <l.famiglini@campus.unimib.it> Neural models integration
- Davide Pietrasanta <d.pietrasanta@campus.unimib.it> Bayesian Optimization

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template. Thanks to all the developers that released their topic models' implementations. A special thanks goes to tenggaard who helped us find many bugs in early octis releases and to Emil Rijcken who kindly wrote two guides on how to use OCTIS :)