octis Documentation

Release 1.10.4

Silvia Terragni

Sep 09, 2022

CONTENTS:

CHAPTER

ONE

OCTIS : OPTIMIZING AND COMPARING TOPIC MODELS IS SIMPLE!

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

1.1 Install

You can install OCTIS with the following command:

```
pip install octis
```

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

1.2 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

1.3 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)	

Emil Rijcken kindly wrote two guides on how to use OCTIS with practical examples:

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

1.4 Datasets and Preprocessing

1.4.1 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!

1.4.2 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

1.4.3 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.

1.4.4 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,
```

(continues on next page)

(continued from previous page)

```
                                lemmatize=True, stopword_list='english',
                                min_chars=1, min_words_docs=0)
# preprocess
dataset = preprocessor.preprocess_dataset(documents_path=r'..\corpus.txt', labels_path=r
↪'..\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.

1.5 Topic Models and Evaluation

1.5.1 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.

1.5.2 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.

1.5.3 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
```

1.5.4 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()`

1.5.5 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.

1.6 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 signature
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 the results
                             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

1.7 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 rhe 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

1.8 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, Antonio},
    year={2021},
    booktitle={Proceedings of the 16th Conference of the European Chapter of the␣
↪Association for Computational Linguistics: System Demonstrations},
    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},
```

}

1.9 Team

1.9.1 Project and Development Lead

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

1.9.2 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

1.9.3 Past Contributors

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

1.10 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 :)

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install OCTIS, run this command in your terminal:

```
$ pip install octis
```

This is the preferred method to install OCTIS, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for OCTIS can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/mind-lab/octis
```

Or download the tarball:

```
$ curl -OJL https://github.com/mind-lab/octis/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

To use OCTIS in a project:

```
import octis
```

CHAPTER

FOUR

MODULES

4.1 Dataset

class octis.dataset.dataset.**Dataset**(*corpus=None*, *vocabulary=None*, *labels=None*, *metadata=None*, *document_indexes=None*)

Dataset handles a dataset and offers methods to access, save and edit the dataset data

fetch_dataset(*dataset_name*, *data_home=None*, *download_if_missing=True*)

Load the filenames and data from a dataset. Parameters ————- dataset_name: name of the dataset to download or retrieve data_home : optional, default: None

Specify a download and cache folder for the datasets. If None, all data is stored in '~/octis' subfolders.

download_if_missing
[optional, True by default] If False, raise an IOError if the data is not locally available instead of trying to download the data from the source site.

load_custom_dataset_from_folder(*path*, *multilabel=False*)

Loads all the dataset from a folder Parameters ————- path : path of the folder to read

save(*path*, *multilabel=False*)

Saves all the dataset info in a folder Parameters ————- path : path to the folder in which files are saved.

If the folder doesn't exist it will be created

4.2 Data Preprocessing

4.3 Evaluation Measures

class octis.evaluation_metrics.metrics.**AbstractMetric**

Class structure of a generic metric implementation

abstract score(*model_output*)

Retrieves the score of the metric

Parameters
model_output – output of a topic model in the form of a dictionary. See model for details on

the model output :type model_output: dict

class octis.evaluation_metrics.coherence_metrics.**Coherence**(*texts=None*, *topk=10*, *measure='c_npmi'*)

score(*model_output*)

Retrieve the score of the metric

model_output
[dictionary, output of the model] key 'topics' required.

score : coherence score

class octis.evaluation_metrics.coherence_metrics.**WECoherenceCentroid**(*topk=10*, *word2vec_path=None*, *binary=True*)

score(*model_output*)

Retrieve the score of the metric

Parameters
model_output – dictionary, output of the model. key 'topics' required.

:return topic coherence computed on the word embeddings

class octis.evaluation_metrics.coherence_metrics.**WECoherencePairwise**(*word2vec_path=None*, *binary=False*, *topk=10*)

score(*model_output*)

Retrieve the score of the metric

model_output
[dictionary, output of the model] key 'topics' required.

score
[topic coherence computed on the word embeddings] similarities

class octis.evaluation_metrics.diversity_metrics.**InvertedRBO**(*topk=10*, *weight=0.9*)

score(*model_output*)

Retrieves the score of the metric

:param model_output : dictionary, output of the model. the 'topics' key is required.

class octis.evaluation_metrics.diversity_metrics.**KLDivergence**

score(*model_output*)

Retrieves the score of the metric

Parameters
model_output – output of a topic model in the form of a dictionary. See model for details on

the model output :type model_output: dict

class octis.evaluation_metrics.diversity_metrics.**LogOddsRatio**

score(*model_output*)

Retrieves the score of the metric

Parameters
model_output – output of a topic model in the form of a dictionary. See model for details on

the model output :type model_output: dict

class octis.evaluation_metrics.diversity_metrics.**TopicDiversity**(*topk=10*)

score(*model_output*)

Retrieves the score of the metric

model_output
[dictionary, output of the model] key 'topics' required.

td : score

class octis.evaluation_metrics.diversity_metrics.**WordEmbeddingsInvertedRBO**(*topk=10*, *weight=0.9*, *normalize=True*, *word2vec_path=None*, *binary=True*)

score(*model_output*)

Returns
rank_biased_overlap over the topics

class octis.evaluation_metrics.diversity_metrics.**WordEmbeddingsInvertedRBOCentroid**(*topk=10*, *weight=0.9*, *normalize=True*, *word2vec_path=None*, *binary=True*)

score(*model_output*)

Returns
rank_biased_overlap over the topics

class octis.evaluation_metrics.classification_metrics.**AccuracyScore**(*dataset*, *average='micro'*, *use_log=False*, *scale=True*, *kernel='linear'*, *same_svm=False*)

score(*model_output*)

Retrieves the score of the metric

model_output
[dictionary, output of the model. 'topic-document-matrix' and] 'test-topic-document-matrix' keys are required.

score : score

class octis.evaluation_metrics.classification_metrics.**ClassificationScore**(*dataset*, *average='micro'*, *use_log=False*, *scale=True*, *kernel='linear'*, *same_svm=False*)

score(*model_output*)

Retrieves the score of the metric

Parameters
model_output – output of a topic model in the form of a dictionary. See model for details on

the model output :type model_output: dict

class octis.evaluation_metrics.classification_metrics.**F1Score**(*dataset*, *average='micro'*, *use_log=False*, *scale=True*, *kernel='linear'*, *same_svm=False*)

score(*model_output*)

Retrieves the score of the metric

model_output
[dictionary, output of the model. keys 'topic-document-matrix' and] 'test-topic-document-matrix' are required.

score : score

class octis.evaluation_metrics.classification_metrics.**PrecisionScore**(*dataset*, *average='micro'*, *use_log=False*, *scale=True*, *kernel='linear'*, *same_svm=False*)

score(*model_output*)

Retrieves the score of the metric

model_output
[dictionary, output of the model. 'topic-document-matrix' and] 'test-topic-document-matrix' keys are required.

score : score

class octis.evaluation_metrics.classification_metrics.**RecallScore**(*dataset*, *average='micro'*, *use_log=False*, *scale=True*, *kernel='linear'*, *same_svm=False*)

score(*model_output*)

Retrieves the score of the metric

model_output
[dictionary, output of the model. 'topic-document-matrix' and] 'test-topic-document-matrix' keys are required.

score : score

class octis.evaluation_metrics.topic_significance_metrics.**KL_background**

score(*model_output*)

Retrieves the score of the metric

model_output
[dictionary, output of the model] 'topic-document-matrix' required

result : score

class octis.evaluation_metrics.topic_significance_metrics.**KL_uniform**

score(*model_output*, *per_topic=False*)

Retrieves the score of the metric

model_output
[dictionary, output of the model] 'topic-word-matrix' required

per_topic: if True, it returns the score for each topic

result : score

class octis.evaluation_metrics.topic_significance_metrics.**KL_vacuous**

score(*model_output*)

Retrieves the score of the metric

model_output
[dictionary, output of the model] 'topic-word-matrix' required 'topic-document-matrix' required

result : score

4.4 Optimization

class octis.optimization.optimizer.**Optimizer**

Class Optimizer to perform Bayesian Optimization on Topic Model

optimize(*model*, *dataset*, *metric*, *search_space*, *extra_metrics=None*, *number_of_call=5*, *n_random_starts=1*, *initial_point_generator='lhs'*, *optimization_type='Maximize'*, *model_runs=5*, *surrogate_model='RF'*, *kernel=1**2 * Matern(length_scale=1, nu=1.5)*, *acq_func='LCB'*, *random_state=False*, *x0=None*, *y0=None*, *save_models=True*, *save_step=1*, *save_name='result'*, *save_path='results/'*, *early_stop=False*, *early_step=5*, *plot_best_seen=False*, *plot_model=False*, *plot_name='B0_plot'*, *log_scale_plot=False*, *topk=10*)

Perform hyper-parameter optimization for a Topic Model

Parameters

- **model** (*OCTIS Topic Model*) – model with hyperparameters to optimize
- **dataset** (*OCTIS dataset*) – dataset for the model dataset
- **metric** (*OCTIS metric*) – metric used for the optimization
- **search_space** (*skopt space object*) – a dictionary of hyperparameters to optimize (each parameter is defined as a skopt space)
- **extra_metrics** (*list of metrics, optional*) – list of extra-metrics to compute during the optimization
- **number_of_call** (*int, optional*) – number of evaluations of metric
- **n_random_starts** (*int, optional*) – number of evaluations of metric with random points before approximating it with surrogate model

- **initial_point_generator** (*str, optional*) – set an initial point generator. Can be either "random", "sobol", "halton" ,"hammersly","lhs"
- **optimization_type** – Set "Maximize" if you want to maximize metric, "Minimize" if you want to minimize
- **model_runs** –
- **surrogate_model** – set a surrogate model. Can be either "GP" (Gaussian Process), "RF" (Random Forest) or "ET" (Extra-Tree)
- **kernel** – set a kernel function
- **acq_func** – Function to minimize over the surrogate model. Can be either: "LCB" (Lower Confidence Bound), "EI" (Expected improvement) OR "PI" (Probability of Improvement)
- **random_state** – Set random state to something other than None for reproducible results.
- **x0** – List of initial input points.
- **y0** – Evaluation of initial input points.
- **save_models** – if 'True' save all the topic models generated during the optimization process
- **save_step** – decide how much to save the results of the optimization
- **save_name** – name of the file where the results of the optimization will be saved
- **save_path** (*str, optional*) – Path where the results of the optimization (json file) will be saved
- **early_stop** (*bool, optional*) – if "True" stop the optimization if there is no improvement after early_step evaluations
- **early_step** (*int, optional*) – number of iterations with no improvement after which optimization will be stopped (if early_stop is True)
- **plot_best_seen** (*bool, optional*) – If "True" save a convergence plot of the result of a Bayesian_optimization (i.e. the best seen for each iteration)
- **plot_model** (*bool, optional*) – If "True" save the boxplot of all the model runs
- **plot_name** (*str, optional*) – Set the name of the plots (best_seen and model_runs).
- **log_scale_plot** (*bool, optional*) – if "True" use the logarithmic scale for the plots.
- **topk** (*int, optional*) –

Type
int, optional

Type
str, optional

Type
str, optional

Type
int, optional

Type
list, optional

Type
list, optional

Type
bool, optional

Type
int, optional

Type
str, optional

Returns
OptimizerEvaluation object

Return type
class

resume_optimization(*name_path*, *extra_evaluations=0*)

Restart the optimization from the json file.

Parameters

- **name_path** (*str*) – path of the json file
- **extra_evaluations** (*int*) – extra iterations for the BO optimization

Returns
object with the results of the optimization

Return type
object

octis.optimization.optimizer_tool.**check_instance**(*obj*)

Check if a specific object con be inserted in the json file.

Parameters
obj (*[str,float, int, bool, etc.]*) – an object of the optimization to be saved

Returns
'True' if the object can be inserted in a json file, 'False' otherwise

Return type
bool

octis.optimization.optimizer_tool.**choose_optimizer**(*optimizer*)

Choose a surrogate model for Bayesian Optimization

Parameters
optimizer (Optimizer) – list of setting of the BO experiment

Returns
surrogate model

Return type
scikit object

octis.optimization.optimizer_tool.**convergence_res**(*values*, *optimization_type='minimize'*)

Compute the list of values to plot the convergence plot (i.e. the best seen at each iteration)

Parameters

- **values** (*list*) – the result(s) for which to compute the convergence trace.
- **optimization_type** (*str*) – "minimize" if the problem is a minimization problem, "maximize" otherwise

Returns
a list with the best min seen for each iteration

Return type
list

octis.optimization.optimizer_tool.**convert_type**(*obj*)

Convert a numpy object to a python object

Parameters
obj (*numpy object*) – object to be checked

Returns
python object

Return type
python object

octis.optimization.optimizer_tool.**early_condition**(*values*, *n_stop*, *n_random*)

Compute the early-stop criterium to stop or not the optimization.

Parameters

- **values** (*list*) – values obtained by Bayesian Optimization
- **n_stop** (*int*) – Range of points without improvement
- **n_random** (*int*) – Random starting points

Returns
'True' if early stop condition reached, 'False' otherwise

Return type
bool

octis.optimization.optimizer_tool.**importClass**(*class_name*, *module_name*, *module_path*)

Import a class runtime based on its module and name

Parameters

- **class_name** (*str*) – name of the class
- **module_name** (*str*) – name of the module
- **module_path** (*str*) – absolute path to the module

Returns
class object

Return type
class

octis.optimization.optimizer_tool.**load_model**(*optimization_object*)

Load the topic model for the resume of the optimization

Parameters
optimization_object (*dict*) – dictionary of optimization attributes saved in the jaon file

Returns
topic model used during the BO.

Return type
object model

octis.optimization.optimizer_tool.**load_search_space**(*search_space*)

Load the search space from the json file

Parameters
search_space – dictionary of the search space (insertable in a json file)

Returns
dictionary for the search space (for scikit optimize)

Return type
dict

octis.optimization.optimizer_tool.**plot_bayesian_optimization**(*values*, *name_plot*, *log_scale=False*, *conv_max=True*)

Save a convergence plot of the result of a Bayesian_optimization.

Parameters

- **values** (*list*) – List of objective function values
- **name_plot** (*str*) – Name of the plot
- **log_scale** (*bool, optional*) – 'True' if you want a log scale for y-axis, 'False' otherwise
- **conv_max** (*bool, optional*) – 'True' for a minimization problem, 'False' for a maximization problem

octis.optimization.optimizer_tool.**plot_model_runs**(*model_runs*, *current_call*, *name_plot*)

Save a boxplot of the data (Works only when optimization_runs is 1).

Parameters

- **model_runs** (*dict*) – dictionary of all the model runs.
- **current_call** (*int*) – number of calls computed by BO
- **name_plot** (*str*) – Name of the plot

octis.optimization.optimizer_tool.**save_search_space**(*search_space*)

Save the search space in the json file

Parameters
search_space (*dict*) – dictionary of the search space (scikit-optimize object)

Returns
dictionary for the seach space, which can be saved in a json file

Return type
dict

octis.optimization.optimizer_tool.**select_metric**(*metric_parameters*, *metric_name*)

Select the metric for the resume of the optimization

Parameters

- **metric_parameters** (*list*) – metric parameters
- **metric_name** (*str*) – name of the metric

Returns
metric

Return type
metric object

4.5 Models

class octis.models.model.**AbstractModel**

Class structure of a generic Topic Modeling implementation

set_hyperparameters(***kwargs*)

Set model hyperparameters

Parameters
****kwargs** – a dictionary of in the form {hyperparameter name: value}

abstract train_model(*dataset*, *hyperparameters*, *top_words=10*)

Train the model. :param dataset: Dataset :param hyperparameters: dictionary in the form {hyperparameter name: value} :param top_words: number of top significant words for each topic (default: 10)

Return model_output
a dictionary containing up to 4 keys: *topics*, *topic-word-matrix*,

topic-document-matrix, *test-topic-document-matrix*. *topics* is the list of the most significant words for each topic (list of lists of strings). *topic-word-matrix* is the matrix (num topics x ||vocabulary||) containing the probabilities of a word in a given topic. *topic-document-matrix* is the matrix (||topics|| x ||training documents||) containing the probabilities of the topics in a given training document. *test-topic-document-matrix* is the matrix (||topics|| x ||testing documents||) containing the probabilities of the topics in a given testing document.

octis.models.model.**load_model_output**(*output_path*, *vocabulary_path=None*, *top_words=10*)

Loads a model output from the choosen directory

Parameters

- **output_path** – path in which th model output is saved
- **vocabulary_path** – path in which the vocabulary is saved (optional, used to retrieve the top k words of each topic)
- **top_words** – top k words to retrieve for each topic (in case a vocabulary path is given)

octis.models.model.**save_model_output**(*model_output*, *path='.'*, *appr_order=7*)

Saves the model output in the chosen directory

Parameters

- **model_output** – output of the model
- **path** – path in which the file will be saved and name of the file
- **appr_order** – approximation order (used to round model_output values)

class octis.models.LDA.**LDA**(*num_topics=100*, *distributed=False*, *chunksize=2000*, *passes=1*, *update_every=1*, *alpha='symmetric'*, *eta=None*, *decay=0.5*, *offset=1.0*, *eval_every=10*, *iterations=50*, *gamma_threshold=0.001*, *random_state=None*)

hyperparameters_info()

Returns hyperparameters informations

info()

Returns model informations

partitioning(*use_partitions*, *update_with_test=False*)

Handle the partitioning system to use and reset the model to perform new evaluations

use_partitions: True if train/set partitioning is needed, False
otherwise

update_with_test: True if the model should be updated with the test set,
False otherwise

set_hyperparameters(***kwargs*)

Set model hyperparameters

train_model(*dataset*, *hyperparams=None*, *top_words=10*)

Train the model and return output

dataset : dataset to use to build the model hyperparams : hyperparameters to build the model top_words : if greater than 0 returns the most significant words for each topic in the output

(Default True)

result
[dictionary with up to 3 entries,] 'topics', 'topic-word-matrix' and 'topic-document-matrix'

class octis.models.NMF_scikit.**NMF_scikit**(*num_topics=100*, *init=None*, *alpha=0*, *l1_ratio=0*, *regularization='both'*, *use_partitions=True*)

hyperparameters_info()

Returns hyperparameters informations

partitioning(*use_partitions*, *update_with_test=False*)

Handle the partitioning system to use and reset the model to perform new evaluations

use_partitions: True if train/set partitioning is needed, False
otherwise

update_with_test: True if the model should be updated with the test set,
False otherwise

train_model(*dataset*, *hyperparameters=None*, *topics=10*)

Train the model and return output

dataset : dataset to use to build the model hyperparameters : hyperparameters to build the model topics : if greather than 0 returns the most significant words

for each topic in the output Default True

result
[dictionary with up to 3 entries,] 'topics', 'topic-word-matrix' and 'topic-document-matrix'

class octis.models.CTM.**CTM**(*num_topics=10*, *model_type='prodLDA'*, *activation='softplus'*, *dropout=0.2*, *learn_priors=True*, *batch_size=64*, *lr=0.002*, *momentum=0.99*, *solver='adam'*, *num_epochs=100*, *reduce_on_plateau=False*, *prior_mean=0.0*, *prior_variance=None*, *num_layers=2*, *num_neurons=100*, *use_partitions=True*, *num_samples=10*, *inference_type='zeroshot'*, *bert_path=''*, *bert_model='bert-base-nli-mean-tokens'*)

train_model(*dataset*, *hyperparameters=None*, *top_words=10*)

trains CTM model

Parameters

- **dataset** – octis Dataset for training the model
- **hyperparameters** – dict, with optionally) the following information:
- **top_words** – number of top-n words of the topics (default 10)

class octis.models.ETM.**ETM**(*num_topics=10*, *num_epochs=100*, *t_hidden_size=800*, *rho_size=300*, *embedding_size=300*, *activation='relu'*, *dropout=0.5*, *lr=0.005*, *optimizer='adam'*, *batch_size=128*, *clip=0.0*, *wdecay=1.2e-06*, *bow_norm=1*, *device='cpu'*, *top_word=10*, *train_embeddings=True*, *embeddings_path=None*, *embeddings_type='pickle'*, *binary_embeddings=True*, *headerless_embeddings=False*, *use_partitions=True*)

train_model(*dataset*, *hyperparameters=None*, *top_words=10*)

Train the model. :param dataset: Dataset :param hyperparameters: dictionary in the form {hyperparameter name: value} :param top_words: number of top significant words for each topic (default: 10)

Return model_output

a dictionary containing up to 4 keys: *topics*, *topic-word-matrix*,

topic-document-matrix, *test-topic-document-matrix*. *topics* is the list of the most significant words for each topic (list of lists of strings). *topic-word-matrix* is the matrix (num topics x ||vocabulary||) containing the probabilities of a word in a given topic. *topic-document-matrix* is the matrix (||topics|| x ||training documents||) containing the probabilities of the topics in a given training document. *test-topic-document-matrix* is the matrix (||topics|| x ||testing documents||) containing the probabilities of the topics in a given testing document.

CHAPTER

FIVE

HYPER-PARAMETER OPTIMIZATION

The core of OCTIS framework consists of an efficient and user-friendly way to select the best hyper-parameters for a Topic Model using Bayesian Optimization.

To inizialize an optimization, inizialize the Optimizer class:

```
from octis.optimization.optimizer import Optimizer
optimizer = Optimizer()
```

Choose the dataset you want to analyze.

```
from octis.dataset.dataset import Dataset
dataset = Dataset()
dataset.load("octis/preprocessed_datasets/M10")
```

Choose a Topic-Model.

```
from octis.models.LDA import LDA
model = LDA()
model.hyperparameters.update({"num_topics": 25})
```

Choose the metric function to optimize.

```
from octis.evaluation_metrics.coherence_metrics import Coherence
metric_parameters = {
    'texts': dataset.get_corpus(),
    'topk': 10,
    'measure': 'c_npmi'
}
npmi = Coherence(metric_parameters)
```

Create the search space for the optimization.

```
from skopt.space.space import Real
search_space = {
    "alpha": Real(low=0.001, high=5.0),
    "eta": Real(low=0.001, high=5.0)
}
```

Finally, launch the optimization.

```
optimization_result=optimizer.optimize(model,
                    dataset,
```

(continues on next page)

(continued from previous page)

```
                    npmi,
                    search_space,
                    number_of_call=10,
                    n_random_starts=3,
                    model_runs=3,
                    save_name="result",
                    surrogate_model="RF",
                    acq_func="LCB"
                     )
```

where:

- number_of_call: int, default: 5. Number of function evaluations.
- n_random_starts: int, default: 1. Number of random points used to inizialize the BO
- model_runs: int: default: 3. Number of model runs.
- save_name: str, default "results". Name of the json file where all the results are saved
- surrogate_model: str, default: "RF". Probabilistic surrogate model used to build to prior on the objective function. Can be either:
  - "RF" for Random Forest regression
  - "GP" for Gaussian Process regression
  - "ET" for Extra-tree Regression
- acq_function: str, default: "EI". function to optimize the surrogate model. Can be either:
  - "LCB" for lower confidence bound
  - "EI" for expected improvment
  - "PI" for probability of improvment

The results of the optimization are saved in the json file, by default. However, you can save the results of the optimization also in a user-friendly csv file

```
optimization_result.save_to_csv("results.csv")
```

5.1 Resume the optimization

Optimization runs, for some reason, can be interrupted. With the help of the `resume_optimization` you can restart the optimization run from the last saved iteration.

```
optimizer = Optimizer()
optimizer.resume_optimization(json_path)
```

where `json_path` is the path of json file of the previous results.

5.2 Continue the optimization

Suppose that, after an optimization process, you want to perform three extra-evaluations. You can do this using the method `resume_optimization`.

```
optimizer = Optimizer()
optimizer.resume_optimization(json_path, extra_evaluations=3)
```

where `extra_evaluations` (int, default 0) is the number of extra-evaluations to perform.

5.3 Inspect an extra-metric

Suppose that, during the optimization process, you want to inspect the value of another metric. For example, suppose that you want to check the value of

```
metric_parameters = {
    'texts': dataset.get_corpus(),
    'topk': 10,
    'measure': 'c_npmi'
}
npmi2 = Coherence(metric_parameters)
```

You can add this as a parameter.

```
optimization_result=optimizer.optimize(model,
                    dataset,
                    npmi,
                    search_space,
                    number_of_call=10,
                    n_random_starts=3,
                    extra_metrics=[npmi2]
                     )
```

where `extra_metrics` (list, default None) is the list of extra metrics to inspect.

5.4 Early stopping

Suppose that you want to terminate the optimization process if there is no improvement after a certain number of iterations. You can apply an early stopping criterium during the optimization.

```
optimization_result=optimizer.optimize(model,
                    dataset,
                    npmi,
                    search_space,
                    number_of_call=10,
                    n_random_starts=3,
                    early_stop=True,
                    early_step=5,
                     )
```

where `early_step` (int, default 5) is the number of function evaluations after that the optimization process is stopped.

CHAPTER

SIX

LOCAL DASHBOARD

The local dashboard is 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 custom implementations.

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

```
python OCTIS/dashboard/server.py --port [port number] --dashboardState [path to␣
↪dashboard state file]
```

The port parameter is optional and the selected port number will be used to host the dashboard server, the default port is 5000. The dashboardState parameter is optional and the selected json file will be used to save the informations used to launch and find the experiments, the default dashboardState path is the current directory.

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

6.1 Using the Dashboard

When the dashboard opens, the home will be automatically loaded on your browser.

6.1.1 Create new experiments

To create a new experiment click on the `CREATE EXPERIMENTS` tab. In this tab have to choose:

- The folder in which you want to save the experiment results
- The name of the experiment
- The name of the batch of experiments in which the experiment is contained
- The dataset
- The model to optimize
- Hyperparameters of the model to optimize
- Search space of the hyperparameters to optimize
- The metric to optimize

- Parameters of the metric
- Metrics to track [optional]
- Parameters of the metrics to track [optional]
- Optimization parameters

After that you can click on `Start Experiment` and the experiment will be added to the Queue.

6.1.2 Visualize and compare all the experiments

To visualize the experiments click on the `VISUALIZE EXPERIMENTS` tab. In this tab, you can choose which batch (or set of batches) to visualize.

A plot of each experiment that contains the best-seen evaluation at each iteration is visualized in a grid. Alternatively, you can visualize a box plot at each iteration to understand if a given hyper-parameter configuration is noisy (high variance) or not.

You can interact with the single experiment graphic or choose to have a look at the single experiment by clicking on `Click here to inspect the results`.

It is possible to decide in which order to show the experiments and apply some filters to have a more intuitive visualization of the experiments.

6.1.3 Visualize a custom experiment

In the `VISUALIZE EXPERIMENTS` tab, after clicking on the `Click here to inspect the results` button, you will be redirected to the single experiment tab. In this tab, you can visualize all the information and statistics related to the experiment, including the best hyper-parameter configuration and the best value of the optimized metric. You can also have an outline of the statistics of the tracked metrics.

It is also possible to have a look at a word cloud obtained from the most relevant words of a given topic, scaled by their probability; the topic distribution on each document (and a preview of the document), and the weight of each word of the vocabulary for each topic.

6.1.4 Manage the experiment queue

To manage the experiment queue click on the `MANAGE EXPERIMENTS` tab. In this tab, you can pause or resume the execution of an experiment. You can also change the order of the experiments to perform or delete the ones you are no longer interested in.

6.2 Frequently used terms

6.2.1 Batch

A batch of experiments is a set of related experiments that can be recognized using a keyword referred to as batch `name`.

6.2.2 Model runs

In the optimization context of the framework, since the performance estimated by the evaluation metrics can be affected by noise, the objective function is computed as the median of a given number of `model runs` (i.e., topic models run with the same hyperparameter configuration)

CHAPTER

SEVEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/MIND-Lab/OCTIS/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

7.1.4 Write Documentation

OCTIS could always use more documentation, whether as part of the official OCTIS docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/MIND-Lab/OCTIS/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Get Started!

Ready to contribute? Here's how to set up *OCTIS* for local development.

1. Fork the *OCTIS* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/OCTIS.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv OCTIS
$ cd OCTIS/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 octis tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7 and 3.8, and for PyPI. Make sure you have enabled workflow actions for your GitHub fork and that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ pytest tests/test_octis.py
```

7.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

GitHub Actions will then deploy to PyPI if tests pass.

CHAPTER

EIGHT

CREDITS

8.1 Project and Development Lead

- Silvia Terragni <s.terragni4@campus.unimib.it>
- Elisabetta Fersini <elisabetta.fersini@unimib.it> University of Milano-Bicocca
- Antonio Candelieri <antonio.candelieri@unimib.it> University of Milano-Bicocca

8.2 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

8.3 Past Contributors

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

CHAPTER

NINE

HISTORY

9.1 1.10.4 (2022-05-20)

- Update metadata Italian datasets
- Fix dataset encoding (#57)
- Fix word embeddings topic coherence (#58)
- Fix dataset name BBC_News (#59)

9.2 1.10.3 (2022-02-20)

- Fix KL Divergence in diversity metrics (#51, #52)

9.3 1.10.2 (2021-12-20)

- Bug fix optimizer evaluation with additional metrics (#46)

9.4 1.10.1 (2021-12-08)

- Bug fix Coherence with word embeddings (#43, #45)

9.5 1.10.0 (2021-11-21)

- ETM now supports different formats of word embeddings (#36)
- Bug fix similarity measures (#41)
- Minor fixes

9.6 1.9.0 (2021-09-27)

- Bug fix preprocessing (#26)
- Bug fix ctm (#28)
- Bug fix weirbo_centroid (#31)
- Added new Italian datasets
- Minor fixes

9.7 1.8.3 (2021-07-26)

- Gensim migration from 3.8 to >=4.0.0

9.8 1.8.2 (2021-07-25)

- Fixed unwanted sorting of documents

9.9 1.8.1 (2021-07-08)

- Fixed gensim version (#22)

9.10 1.8.0 (2021-06-18)

- Added per-topic kl-uniform significance

9.11 1.7.1 (2021-06-09)

- Handling multilabel classification
- Fixed preprocessing when dataset is not split (#17)

9.12 1.6.0 (2021-05-20)

- Added regularization hyperparameter to NMF_scikit
- Added similarity metrics
- Fixed handling of stopwords in preprocessing
- Fixed coherence and diversity metrics
- Added new metrics tests

9.13 1.4.0 (2021-05-12)

- Fixed CTM training when only training dataset is used
- Dashboard bugs fixed
- Minor bug fixes
- Added new tests for TM training

9.14 1.3.0 (2021-04-25)

- Added parameter num_samples to CTM, NeuralLDA and ProdLDA
- Bug fix AVITM

9.15 1.2.1 (2021-04-21)

- Bug fix info dataset

9.16 1.2.0 (2021-04-20)

- Tomotopy LDA's implementation should work now

9.17 1.1.1 (2021-04-19)

- bug fix dataset download
- CTM is no longer verbose

9.18 1.1.0 (2021-04-18)

- New classification metrics
- Vocabulary downloader fix

9.19 1.0.2 (2021-04-16)

- Dataset downloader fix

9.20 1.0.0 (2021-04-16)

- New metrics initialization (do not support dictionaries as input anymore)
- Optimization, dataset and dashboard bug fixes
- Refactoring
- Updated README and documentation

9.21 0.4.0 (2021-04-15)

- Dataset preprocessing produces also an indexes.txt file containing the indexes of the documents
- Eval metrics bug fixes
- BBC news added in the correct format

9.22 0.3.0 (2021-04-10)

- Bug fixes

9.23 0.2.0 (2021-03-30)

- New dataset format

9.24 0.1.0 (2021-03-11)

- First release on PyPI.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

M

N

O

P

R

S

T

W