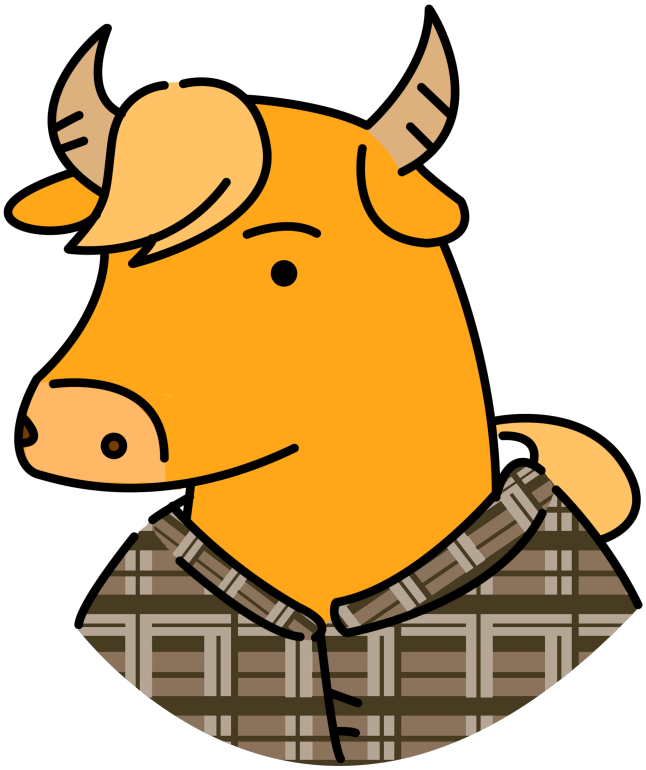
Buffalo

Release 1.2.0

Jul 29, 2022

Contents:



Buffalo is a fast and scalable production-ready open source project for recommendation systems. Buffalo effectively utilizes system resources, enabling high performance even on low-spec machines. The implementation is optimized for CPU and SSD. Even so, it shows good performance with GPU accelerator, too. Buffalo, developed by Kakao, has been reliably used in production for various Kakao services.

Buffalo provides the following algorithms:

- Alternating Least Squares[1]
- Bayesian Personalized Ranking Matrix Factorization[2]
- Word2Vec[3]
- CoFactors[4]

All algorithms are optimized for multi-threading and some support GPU accelerators.

One of the best things about this library is a very low memory usage compared to other competing libraries. With chunked data management and batch learning with HDF5, handling a large-scale data, even bigger than memory size on laptop machine, is made possible. Check out the benchmarks page(https://github.com/kakao/buffalo/tree/master/benchmark) for more details on Buffalo performance.

Plus, Buffalo provides a variety of convenient features for research and production purposes, such as tensorboard integration, hyper-parameter optimization and so on.

[1] Hu, Yifan, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets." 2008 Eighth IEEE International Conference on Data Mining. Ieee, 2008.

[2] Rendle, Steffen, et al. "BPR: Bayesian personalized ranking from implicit feedback." Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence. AUAI Press, 2009.

[3] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

[4] Liang, Dawen, et al. "Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence." Proceedings of the 10th ACM conference on recommender systems. ACM, 2016.

CHAPTER 1

Introduction

Buffalo is a fast and scalable production-ready open source project for recommendation systems. Buffalo effectively utilizes system resources, enabling high performance even on low-spec machines. The implementation is optimized for CPU and SSD. Even so, it shows good performance with GPU accelerator, too. Buffalo, developed by Kakao, has been reliably used in production for various Kakao services.

Buffalo provides the following algorithms:

- Alternating Least Squares (ALS)
- Bayesian Personalized Ranking Matrix Factorization (BPR)
- Word2Vec (W2V)
- CoFactors (CFR)
- Weighted Approximate Rank Pairwise (WARP)
- Probabilistic latent semantic indexing (pLSI)

ALS is one of the most famous matrix factorization models which decompose the observed user-item interaction matrix into user and item latent factors. More ranking optimized models are BPR and WARP. W2V and CFR mainly focus on the item co-occurrence data. Unlike other models, pLSI (a.k.a probabilistic latent semantic analysis) is a soft clustering module that performs a low-rank approximation of user-item matrix on the basis of their frequencies.

All algorithms are optimized for multi-threading and some support GPU accelerators.

One of the best things about this library is a very low memory usage compared to other competing libraries. With chunked data management and batch learning with HDF5, handling a large-scale data, even bigger than memory size on laptop machine, is made possible. Check out the benchmarks page for more details on Buffalo performance.

Plus, Buffalo provides a variety of convenient features for research and production purposes, such as tensorboard integration, hyper-parameter optimization and so on.

1.1 Installation

Type *pip install buffalo*.

1.1.1 Requirements

- numpy
- cython
- n2
- cmake

1.1.2 From source code

```
$> git clone -b master https://github.com/kakao/buffalo
$> cd buffalo
$> git submodule update --init
$> pip install -r requirements.txt
$> python setup.py install
```

1.2 Basic Usage

We highly recommend starting with the unit-test codes. Checkout ./tests directory, *./tests/algo/test_algo.py* will be a good starting point.

```
$ buffalo.git/tests> nosetests ./algo/test_als.py -v
```

or

```
$ buffalo.git/tests> pytest ./algo/test_als.py -v
```

1.3 Database

We call term *database* as a data file format used by the buffalo internally. Buffalo take data that the Matrix Market or Stream format as input and converts it into a database class which store rawdata using h5py(http://www.h5py.org). The main reason to make custom database is to use the least amount of memory without compromising capacity of data volumn and learning speed.

The Stream data format consists of two files:

- main
  - Assumed that the data is reading history of users from some blog service, then each line is a reading history corresponding to each row of UID files. (i.e. users lists)
  - The reading history is seperated by spaces, and the past is the left and the right is the most recent history.
  - e.g. *A B C D D E* means that a user read the contents in the order A B C D D E.
- uid
  - Each line is repersentational value of a user corresponding to each row in the MAIN file (e.g. user name)
  - Do not allow spaces in each line

For Matrix Market format, please refer to https://math.nist.gov/MatrixMarket/formats.html.

- main

  - Matrix Market data file.
- uid
  - Each line is the actual userkey corresponding to the row id in the MM file.
- iid
  - Each line is the actual itemkey corresponding to the colum id in the MM file.

uid and iid are the data needed to provide human readable results only, not required.

1.4 Hyper paremter Optimization

The Algo classes inherited Optimizable class, a helper class for hyper parameter optimization. It inherits all features from hyperopt(http://hyperopt.github.io/hyperopt/), a well-known library.

The options for optimization should be stored in optimize field in Algo option. The following is a list of possible options. You can find the practical example from the unittests.

- loss(str): Target loss to optimize.
- max_trials(int, option): Maximum experiments for optimization. If not given, run forever.
- min_trials(int, option): Minimum experiments before deploying model. (Since the best parameter may not be found after min_trials, the first best parameter is always deployed)
- deployment(bool): Set True to train model with the best parameter. During the optimization, it dumps the model which beats the previous best loss.
- start_with_default_parameters(bool): If set to True, the loss value of the default parameter is used as the starting loss to beat.
- space(dict): Parameter space definition. For more information, please reference hyperopt documentation.
  - Note) Since hyperopt's randint does not support lower value, we changed the implementation. Please check optimize.py.

1.5 Logging

It is recommend to use the log library of buffalo for consistent log format.

```
>>> from buffalo.misc import log
>>> print(log.NOTSET, log.WARN, log.INFO, log.DEBUG, log.TRACE)
(0, 1, 2, 3, 4, 5)
>>> log.set_log_level(log.WARN)  # this set log-level on Python, C++ both sides.
>>> log.get_log_level()
1
>>>

>>> from buffalo.misc import log
>>> logger = log.get_logger()
>>> with log.pbar(logger.debug, desc='Test', mininterval=1):
    for(i in range(100)):
        time.sleep(0.1)
```

CHAPTER 2

Algorithms

Buffalo provides the following algorithm implementations:

- Alternating Least Squares
- Bayesian Personalized Ranking Matrix Factorization
- Weighted Approximate-Rank Pairwise
- Word2Vec
- CoFactors

All algorithms inherit common parent classes such as Algo, Serializable, TensorboardExtension, Optimizable, Evaluable.

2.1 Algo

2.1.1 Serializable

2.1.2 TensorboardExtension

2.1.3 Optimizable

class buffalo.algo.optimize.**Optimizable**(**args*, ***kwargs*)

Bases: object

get_optimization_data()

optimize()

2.1.4 Evaluable

class buffalo.evaluate.base.**Evaluable**(**args*, ***kargs*)
Bases: object

get_topk(*scores*, *k*, *sorted=True*, *num_threads=4*)

get_validation_results()

prepare_evaluation()

show_validation_results()

2.2 Alternating Least Squares

class buffalo.algo.options.**ALSOption**(**args*, ***kwargs*)
Bases: buffalo.algo.options.AlgoOption

get_default_optimize_option()
Optimization Options for ALS.

Variables

- **loss** (*str*) – Target loss to optimize.
- **max_trials** (*int*) – The maximum experiments for optimization. If not given, run forever.
- **min_trials** (*int*) – The minimum experiments before deploying model. (Since the best parameter may not be found after *min_trials*, the first best parameter is always deployed)
- **deployment** (*bool*) – Set True to train model with the best parameter. During the optimization, it try to dump the model which beated the previous best loss.
- **start_with_default_parameters** (*bool*) – If set to True, the loss value of the default parameter is used as the starting loss to beat.
- **space** (*dict*) – The parameter space definition. For more information, please check reference hyperopt's express. Note) Due to hyperopt's *randint* does not provide lower value, we had to implement it a bait tricky. Please see optimize.py to check how we deal with *randint*.

get_default_option()
Options for Alternating Least Squares.

Variables

- **adaptive_reg** (*bool*) – Set True, for adaptive regularization. (default: False)
- **save_factors** (*bool*) – Set True, to save models. (default: False)
- **accelerator** (*bool*) – Set True, to accelerate training using GPU. (default: False)
- **d** (*int*) – The number of latent feature dimension. (default: 20)
- **num_iters** (*int*) – The number of iterations for training. (default: 10)
- **num_workers** (*int*) – The number of threads. (default: 1)
- **hyper_threads** (*int*) – The number of hyper threads when using cuda cores. (default: 256)

- **reg_u** (*float*) – The L2 regularization coefficient for user embedding matrix. (default: 0.1)
- **reg_i** (*float*) – The L2 regularization coefficient for item embedding matrix. (default: 0.1)
- **alpha** (*float*) – The coefficient of giving more weights to losses on positive samples. (default: 8)
- **eps** (*float*) – epsilon for numerical stability (default: 1e-10)
- **cg_tolerance** (*float*) – tolerance of conjugate gradient for early stopping iterations (default: 1e-10)
- **optimizer** (*str*) – The name of optimizer, should be in [llt, ldlt, manual_cg, eigen_cg, eigen_bicg, eigen_gmres, eigen_dgmres, eigen_minres]. (default: manual_cg)
- **num_cg_max_iters** (*int*) – The number of maximum iterations for conjugate gradient optimizer. (default: 3)
- **model_path** (*str*) – Where to save model.
- **data_opt** (*dict*) – This option will be used to load data if given.

2.3 Bayesian Personalized Ranking Matrix Factorization

class buffalo.algo.options.**BPRMFOption**(**args*, ***kwargs*)

Bases: buffalo.algo.options.AlgoOption

get_default_optimize_option()

Optimization options for BPRMF.

get_default_option()

Options for Bayesian Personalized Ranking Matrix Factorization.

Variables

- **accelerator** (*bool*) – Set True, to accelerate training using GPU. (default: False)
- **use_bias** (*bool*) – Set True, to use bias term for the model.
- **evaluation_period** (*int*) – (default: 100)
- **num_workers** (*int*) – The number of threads. (default: 1)
- **hyper_threads** (*int*) – The number of hyper threads when using cuda cores. (default: 256)
- **num_iters** (*int*) – The number of iterations for training. (default: 100)
- **d** (*int*) – The number of latent feature dimension. (default: 20)
- **update_i** (*bool*) – Set True, to update positive item feature. (default: True)
- **update_j** (*bool*) – Set True, to update negative item feature. (default: True)
- **reg_u** (*float*) – The L2 regularization coefficient for user embedding matrix. (default: 0.025)
- **reg_i** (*float*) – The L2 regularization coefficient for positive item embedding matrix. (default: 0.025)
- **reg_j** (*float*) – The L2 regularization coefficient for negative item embedding matrix. (default: 0.025)

- **reg_b** (*float*) – The L2 regularization coefficient for bias term. (default: 0.025)
- **optimizer** (*str*) – The name of optimizer, should be one of [sgd, adagrad, adam]. (default: sgd)
- **lr** (*float*) – The learning rate.
- **min_lr** (*float*) – The minimum of learning rate, to prevent going to zero by learning rate decaying. (default: 0.0001)
- **beta1** (*float*) – The parameter of Adam optimizer. (default: 0.9)
- **beta2** (*float*) – The parameter of Adam optimizer. (default: 0.999)
- **per_coordinate_normalize** (*bool*) – This is a bit tricky option for Adam optimizer. Before update factors with gradients, do normalize gradients per class by its number of contributed samples. (default: False)
- **sampling_power** (*float*) – This parameter control sampling distribution. When it set to 0, it draw negative items from uniform distribution, while to set 1, it draw from the given data popularation. (default: 0.0)
- **random_positive** (*bool*) – Set True, to draw positive sample uniformly instead of using straight forward positive sample, only implemented in cuda mode, according to the original paper, set True, but we found out False usually produces better results) (default: False)
- **verify_neg** (*bool*) – Set True, to ensure negative sample does not belong to positive samples. (default True)
- **model_path** (*str*) – Where to save model.
- **data_opt** (*dict*) – This option will be used to load data if given.

2.4 Weighted Approximate-Rank Pairwise

class buffalo.algo.options.**WARPOption**(**args*, ***kwargs*)

Bases: buffalo.algo.options.AlgoOption

get_default_optimize_option()

Optimization options for WARP.

get_default_option()

Options for WARP Matrix Factorization.

Variables

- **accelerator** (*bool*) – Set True, to accelerate training using GPU. (default: False)
- **use_bias** (*bool*) – Set True, to use bias term for the model.
- **evaluation_period** (*int*) – (default: 15)
- **num_workers** (*int*) – The number of threads. (default: 1)
- **hyper_threads** (*int*) – The number of hyper threads when using cuda cores. (default: 256)
- **num_iters** (*int*) – The number of iterations for training. (default: 15)
- **d** (*int*) – The number of latent feature dimension. (default: 30)

- **max_trials** (*int*) – The maximum number of attempts to find a violating negative sample during training.
- **update_i** (*bool*) – Set True, to update positive item feature. (default: True)
- **update_j** (*bool*) – Set True, to update negative item feature. (default: True)
- **reg_u** (*float*) – The L2 regularization coefficient for user embedding matrix. (default: 0.0)
- **reg_i** (*float*) – The L2 regularization coefficient for positive item embedding matrix. (default: 0.0)
- **reg_j** (*float*) – The L2 regularization coefficient for negative item embedding matrix. (default: 0.0)
- **reg_b** (*float*) – The L2 regularization coefficient for bias term. (default: 0.0)
- **optimizer** (*str*) – The name of optimizer, should be one of [adagrad, adam]. (default: adagrad)
- **lr** (*float*) – The learning rate. (default: 0.1)
- **min_lr** (*float*) – The minimum of learning rate, to prevent going to zero by learning rate decaying. (default: 0.0001)
- **beta1** (*float*) – The parameter of Adam optimizer. (default: 0.9)
- **beta2** (*float*) – The parameter of Adam optimizer. (default: 0.999)
- **per_coordinate_normalize** (*bool*) – This is a bit tricky option for Adam optimizer. Before update factors with gradients, do normalize gradients per class by its number of contributed samples. (default: False)
- **random_positive** (*bool*) – Set True, to draw positive sample uniformly instead of using straight forward positive sample, only implemented in cuda mode, according to the original paper, set True, but we found out False usually produces better results) (default: False)
- **model_path** (*str*) – Where to save model.
- **data_opt** (*dict*) – This option will be used to load data if given.

2.5 CoFactors

class buffalo.algo.options.**CFROption**(**args*, ***kwargs*)

Bases: buffalo.algo.options.AlgoOption

get_default_optimize_option()

Optimization options for CoFactor.

Variables

- **loss** (*str*) – Target loss to optimize.
- **max_trials** (*int*) – Maximum experiments for optimization. If not given, run forever.
- **min_trials** (*int*) – Minimum experiments before deploying model. (Since the best parameter may not be found after *min_trials*, the first best parameter is always deployed)
- **deployment(** (*bool*) – Set True to train model with the best parameter. During the optimization, it try to dump the model which beated the previous best loss.

- **start_with_default_parameters** (*bool*) – If set to True, the loss value of the default parameter is used as the starting loss to beat.
- **space** (*dict*) – Parameter space definition. For more information, please check reference hyperopt's express. Note) Due to hyperopt's *randint* does not provide lower value, we had to implement it a bait tricky. Please see optimize.py to check how we deal with *randint*.

get_default_option()
Basic Options for CoFactor.

Variables

- **d** (*int*) – The number of latent feature dimension. (default: 20)
- **num_iters** (*int*) – The number of iterations for training. (default: 10)
- **num_workers** (*int*) – The number of threads. (default: 1)
- **reg_u** (*float*) – The L2 regularization coefficient for user embedding matrix. (default: 0.1)
- **reg_i** (*float*) – The L2 regularization coefficient for item embedding matrix. (default: 0.1)
- **reg_c** (*float*) – The L2 regularization coefficient for context embedding matrix. (default: 0.1)
- **eps** (*float*) – epsilon for numerical stability (default: 1e-10)
- **cg_tolerance** (*float*) – The tolerance for early stopping conjugate gradient optimizer. (default: 1e-10)
- **alpha** (*float*) – The coefficient of giving more weights to losses on positive samples. (default: 8.0)
- **l** (*float*) – The relative weight of loss on user-item relation over item-context relation. (default: 1.0)
- **optimizer** (*str*) – The name of optimizer, should be in [llt, ldlt, manual_cg, eigen_cg, eigen_bicg, eigen_gmres, eigen_dgmres, eigen_minres]. (default: manual_cg)
- **num_cg_max_iters** (*int*) – The number of maximum iterations for conjugate gradient optimizer. (default: 3)
- **model_path** (*str*) – Where to save model. (default: '')
- **data_opt** (*dict*) – This option will be used to load data if given. (default: {})

is_valid_option(*opt*)

2.6 Word2Vec

class buffalo.algo.options.**W2VOption**(**args*, ***kwargs*)
Bases: buffalo.algo.options.AlgoOption

get_default_optimize_option()
Optimization options for W2V

get_default_option()
Options for Word2Vec.

Variables

- **evaluation_on_learning** (*bool*) – Set True to do run evaluation on training phrase. (default: False)
- **num_workers** (*int*) – The number of threads. (default: 1)
- **num_iters** (*int*) – The number of iterations for training. (default: 100)
- **d** (*int*) – The number of latent feature dimension. (default: 20)
- **window** (*int*) – The window size. (default: 5)
- **min_count** (*int*) – The minimum required frequency of the words to use training vocabulary. (default: 5)
- **sample** (*float*) – The sampling ratio to downsample the frequent words. (default: 0.001)
- **num_negative_samples** (*int*) – The number of negative noise words. (default: 5)
- **lr** (*float*) – The learning rate.
- **model_path** (*str*) – Where to save model.
- **data_opt** (*dict*) – This option will be used to load data if given.

2.7 pLSI

class buffalo.algo.options.**PLSIOption**(**args*, ***kwargs*)

Bases: buffalo.algo.options.AlgoOption

get_default_optimize_option()

Optimization options for pLSI.

Variables

- **loss** (*str*) – Target loss to optimize.
- **max_trials** (*int*) – Maximum experiments for optimization. If not given, run forever.
- **min_trials** (*int*) – Minimum experiments before deploying model. (Since the best parameter may not be found after *min_trials*, the first best parameter is always deployed)
- **deployment** (*bool*) – Set True to train model with the best parameter. During the optimization, it try to dump the model which beated the previous best loss.
- **start_with_default_parameters** (*bool*) – If set to True, the loss value of the default parameter is used as the starting loss to beat.
- **space** (*dict*) – Parameter space definition. For more information, please check reference hyperopt's express. Note) Due to hyperopt's *randint* does not provide lower value, we had to implement it a bait tricky. Please see optimize.py to check how we deal with *randint*.

get_default_option()

Basic Options for pLSI.

Variables

- **d** (*int*) – The number of latent feature dimension. (default: 20)
- **num_iters** (*int*) – The number of iterations for training. (default: 10)
- **num_workers** (*int*) – The number of threads. (default: 1)

- **alpha1** (*float*) – The coefficient of regularization term for clustering assignment. (default: 1.0)
- **alpha2** (*float*) – The coefficient of regularization term for item preference in each cluster. (default: 1.0)
- **eps** (*float*) – epsilon for numerical stability (default: 1e-10)
- **model_path** (*str*) – Where to save model. (default: '')
- **save_factors** (*bool*) – Set True, to save models. (default: False)
- **data_opt** (*dict*) – This option will be used to load data if given. (default: {})

CHAPTER 3

Parallels

Provides parallel processing feature to algorithm classes.

It is written in C++/OpenMP to maximize CPU utilization. Even with a single thread, it works faster than the default implementation of Algo classes. Parallels also provides a boosting feature to execute *most_similar* function, which is based on approximate nearest neighbors library N2. For performance and examples usage, please refer to the benchmark page and unit test codes.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

References

Index