Package ‘SSDL’

March 10, 2021

Type Package

Title Sketched Stochastic Dictionary Learning

Version 1.1

Date 2021-03-03

Maintainer Olga Permiakova <olga.permiakova@gmail.com>

Description Toolbox for learning a dictionary from large-scale data collection using the Sketched Stochastic Dictionary Learning method (see Permiakova O, Burger T. ``Sketched Stochastic Dictionary Learning for large-scale data and application to large-scale mass spectrometry data'', 2021). It includes the routines for the dictionary initialization.

License GPL (>= 2)

Encoding UTF-8

Depends R (>= 3.6)

Imports Rcpp, bigstatsr (>= 1.2.3), chickn (>= 1.2.3), RcppParallel, glmnet, parallel, foreach, doParallel, doRNG, reshape2, stats, utils, pracma, Rdpack

LinkingTo Rcpp, RcppArmadillo, RcppParallel, rmio

SystemRequirements C++11

RoxygenNote 7.1.1

RdMacros Rdpack

NeedsCompilation yes

Author Olga Permiakova [aut, cre], Thomas Burger [aut]

Repository CRAN

Date/Publication 2021-03-10 19:40:08 UTC

R topics documented:

CDL *Compressive Dictionary Learning*

Description

Implementation of the Sketched Stochastic Dictionary Learning (SSDL) method, which learns a dictionary from a large-scale matrix-like dataset by operating on a compressed version of data (a.k.a. data sketch).

Usage

```
CDL(
  Data,
  K,
  SK_Data = NULL,
  Frequencies = NULL,
  D = NULL,
  pos.dic = TRUE,
  learn_rate = 0.1,
  alpha = 0.9,
  gamma = 0,
  maxEpoch = 5,
  batch_size,
  lambda = 0,
  ncores = 1,
  typeOptim = "Nesterov",
  DIR_tmp = tempdir(),
  grad_t_1 = NULL,
  verbose = 0,
  m = nrow(Frequencies),
  ...
)
```

Arguments

`Data`	is a Filebacked Big Matrix $s \times N$ with data vectors stored in the matrix columns.
`K`	is a dictionary size.
`SK_Data`	is a data sketch. It is a $2m$-dimensional complex vector. The first m coordinates correspond to the real parts and the last m coordinates to the imaginary parts. If it is NULL, the sketch is computed using `Sketch` function of chickn package.

`Frequencies` is a frequency matrix $m \times s$ with frequency vectors in the matrix rows. If NULL, the frequencies are generated using `GenerateFrequencies` function of chickn package.

`D` is an initial dictionary. If it is NULL, the dictionary is initialized by random selection of `K` signals from `Data` and it is saved in the `DIR_tmp` directory.

`pos.dic` indicates whether the dictionary is positive (default) or not.

`learn_rate` is a learning rate value. The default value is 0.1.

`alpha` is a momentum weight.

`gamma` is a decay parameter. The default value is 0, which corresponds to the constant learning rate.

`maxEpoch` is a number of epochs.

`batch_size` is a batch size.

`lambda` is a regularization parameter.

`ncores` is a number of cores. The default value is 1.

`typeOptim` is a type of the optimization scheme used in the dictionary update. Possible values are c('Nesterov', 'Momentum'). It is suggested to use 'Nesterov' scheme.

`DIR_tmp` is a directory to save the initial dictionary and intermediate results.

`grad_t_1` is an initial momentum matrix. By default it is NULL, and it is initialized as a zero matrix.

`verbose` controls how much output is shown and saved during the optimization process. Possible values:

- 0 no output (default value)
- 1 show iteration number and value of objective function
- 2 1 + save a dictionary and a momentum matrix at the end of each epoch.

`m` is a number of the frequency vectors.

`...` are additional parameters passed to GenerateFrequencies function.

Details

`CDL` builds a dictionary by alternating two steps: calculating the code matrix that contains the weights of the dictionary elements, and updating the dictionary. For computational efficiency, the code matrix is computed only for a randomly selected subset of data vectors $x_1, \dots, x_n$ (a.k.a. batch). The code matrix is obtained as a solution of the following optimization problem: $\min\limits_{A \in R^{+}_{K \times n}} \sum_{i=1}^{n} \|x_i - D \cdot \alpha_i\|^2 + \lambda \cdot \|\alpha_i\|_1$, where n denotes a batch size, $A = \{\alpha_1, \dots, \alpha_n\}$ is a code matrix and λ is a regularization parameter which defines the data sparsity level.

The dictionary is updated by taking one step along the gradient of the objective function $F(D, A) = \|SK(Data) - SK(A \cdot D)\|^2$. Two gradient descent update rules are available: Nesterov accelerated and momentum.

$SK(\cdot)$ is a sketch operator, which compresses a matrix into a fixed size complex vector referred to as a data sketch. It has been introduced in Keriven N, Bourrier A, Gribonval R, Pérez P (2018). "Sketching for large-scale learning of mixture models." *Information and Inference: A Journal of the IMA*, **7**(3), 447–508. and it is defined as $SK(Data) = \frac{1}{N} \sum_{i=1}^{N} \exp{(-1i \cdot W \cdot x_i)}$, where W

is a frequency matrix and $x_1, \ldots, x_N$ are data vectors. The data compression is performed using routines from chickn package.

CDL allows also to use the decaying learning rate, *i.e.* $\texttt{learn\_rate}^t = \frac{\texttt{learn\_rate}}{1+(t-1)\cdot\texttt{gamma}}$, where t is the iteration number.

Value

a list

- D is the obtained dictionary,
- objFunProcess is objective function values computed at the end of each iteration,
- learning_rate is learning rate values.

References

- Permiakova O, Burger T (2021). "Sketched Stochastic Dictionary Learning for large-scale data and application to large-scale mass spectrometry data." *under revision in the Statistical analysis and data mining journal*.
- Permiakova O, Guibert R, Kraut A, Fortin T, Hesse A, Burger T (2021). "CHICKN: extraction of peptide chromatographic elution profiles from large scale mass spectrometry data by means of Wasserstein compressive hierarchical cluster analysis." *BMC bioinformatics*, **22**(1), 1–30.

See Also

Gradient_D_cpp_parallel, chickn, chickn::Sketch, chickn::GenerateFrequencies

Examples

```
X = matrix(abs(rnorm(n = 1000)), ncol = 100, nrow = 10)
X_fbm = bigstatsr::as_FBM(X)$save()
W = chickn::GenerateFrequencies(Data = X_fbm, m = 64, N0 = ncol(X_fbm),
                               ncores = 1, niter= 3, nblocks = 2, sigma_start = 0.001)$W
SK= chickn::Sketch(X_fbm, W)
D = CDL(Data = X_fbm, K = 10, SK_Data = SK, Frequencies = W,
        D = NULL, pos.dic = TRUE, maxEpoch = 3, batch_size = 50,
        lambda = 0, learn_rate = 0.1, alpha = 0.9,
        gamma = 0, ncores = 2, DIR_tmp = tempdir(),
        verbose=0, typeOptim = "Nesterov")$D
```

`COMP_initialization` *COMP dictionary initialization*

Description

Dictionary initialization using the Compressive Orthogonal Matching Pursuit (COMP) method

Usage

```
COMP_initialization(
  K,
  Data,
  SK_Data = NULL,
  Frequencies = NULL,
  lower = -Inf,
  upper = Inf,
  maxIter = 1500,
  HardThreshold = FALSE,
  print_level = 0,
  ncores = 1,
  m = nrow(Frequencies),
  ...
)
```

Arguments

`K`	is a dictionary size.
`Data`	is a Filebacked Big Matrix $s \times N$ with data vectors stored in the matrix columns.
`SK_Data`	is a data sketch. It is a $2m$-dimensional complex vector. The first m coordinates correspond to the real parts and the last m coordinates to the imaginary parts. If it is NULL, the sketch is computed using `Sketch` function of chickn package.
`Frequencies`	is a frequency matrix $m \times s$ with frequency vectors in the matrix rows. If NULL, the frequencies are generated using `GenerateFrequencies` function of chickn package.
`lower`	is a lower boundary. It is an s-dimensional vector.
`upper`	is an upper boundary. It is an s-dimensional vector.
`maxIter`	is a maximum number of iterations in the computation of new dictionary element. The default value is 1500.
`HardThreshold`	indicates whether to execute the hard thresholding step. The default is FALSE.
`print_level`	controls how much output is shown during the optimization process. Possible values: • 0 no output (default value) • 1 show iteration number and value of objective function • 2 1 + show values of weights

ncores	is a number of cores. The default value is 1.
m	is a number of the frequency vectors.
...	are additional parameters passed to GenerateFrequencies function.

Details

The initialization routine is based on the Compressive Orthogonal Matching Pursuit (COMP) algorithm. COMP is an iterative greedy method that builds a dictionary operating on a compressed data version (a.k.a. data sketch). It alternates between expanding the dictionary D with a new element d_i, whose sketch $SK(d_i)$ is the most correlated to the residue, and calculating the weights of the dictionary elements $w_1, \dots, w_K$ by minimizing the difference between the data sketch $SK(Data)$ and a linear combination of dictionary sketches, *i.e.* $\|SK(Data) - \sum_{i=1}^{K} w_i \cdot SK(d_i)\|$. Unlike COMP, the implemented dictionary initialization routine does not perform an additional global optimization with respects to both variables: weights and dictionary elements.

Value

a list

- `D` is the obtained dictionary,
- `weights` is the resulting weights,
- `ObjF` is the objective function values computed at each iteration.
- `Sketch` is the data sketch
- `Frequencies` is the frequency matrix

Note

COMP method has been presented in Keriven N, Tremblay N, Traonmilin Y, Gribonval R (2017). "Compressive K-means." In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6369–6373. IEEE.

See Also

`ObjFun_COMP_cpp`, `Gradient_COMP_cpp`, `chickn::Sketch`, `chickn::GenerateFrequencies`, chickn

Examples

```
X = matrix(abs(rnorm(n = 1000)), ncol = 100, nrow = 10)
lb = apply(X, 1, min)
ub = apply(X, 1, max)
X_fbm = bigstatsr::FBM(init = X, ncol = ncol(X), nrow = nrow(X))
m = 64
W = chickn::GenerateFrequencies(Data = X_fbm, m = m, N0 = ncol(X_fbm),
                               ncores = 1, niter= 3, nblocks = 2, sigma_start = 0.001)$W
SK= chickn::Sketch(X_fbm, W)
D0 = COMP_initialization(K = 10, Data = X_fbm, SK_Data = SK, Frequencies = W,
                        lower = lb, upper = ub)$Dictionary
```

`Gradient_COMP_cpp` *COMP Gradient*

Description

The gradient of the objective function from the Compressive Orthogonal Matching Pursuit with respect to a dictionary element.

Usage

```
Gradient_COMP_cpp(d, W, residue)
```

Arguments

`d`	is a dictionary element
`W`	is a frequency matrix $m \times s$ with frequency vectors in matrix rows.
`residue`	is a residue vector.

Details

`Gradient_COMP_cpp` computes the gradient of the objective function $OF(d) = -\frac{SK(d) \cdot r}{\|SK(d)\|}$, where $SK(d)$ denotes a sketch of the dictionary element d and r is the residue vector. The gradient is given as $\nabla_d OF(d) = \frac{-G(SK(d),y,W)}{\|SK(d)\|}$, where a vector $y = r - \left(r^\top \cdot SK(d)\right) \cdot SK(d)$ and a function $G(x, y, W)$ is given as: $G(x, y, W) = (x[1:m] \odot y[m+1:2m] - x[m+1] \odot y[1:m])^\top \cdot W$, where $\odot$ denotes an element-wise vector multiplication.

Value

a gradient vector

See Also

ObjFun_COMP_cpp, COMP_initialization

Examples

```
X = matrix(abs(rnorm(n = 1000)), ncol = 100, nrow = 10)
X_fbm = bigstatsr::as_FBM(X)$save()
W = chickn::GenerateFrequencies(Data = X_fbm, m = 64, N0 = ncol(X_fbm),
                               ncores = 1, niter= 3, nblocks = 2, sigma_start = 0.001)$W
SK= chickn::Sketch(X_fbm, W)
D = X_fbm[, sample(ncol(X_fbm), 10)]
weights = sample(10, 10)/10
SK_D = rbind(cos(W%*%D), sin(W%*%D))
d = D[,1]
r = SK - SK_D%*%weights
Grad = Gradient_COMP_cpp(d, W, r)
```

Gradient_D_cpp_parallel *Gradient_D_cpp_parallel*

Description

Parallel computation of the gradient with respect to a dictionary matrix and the objective function computation.

Usage

```
Gradient_D_cpp_parallel(D, A, W, SK, ComputeGrad = TRUE)
```

Arguments

D	is a dictionary $s \times K$.
A	is a code matrix $K \times n$.
W	is a frequency matrix $m \times s$ with frequency vectors in matrix rows.
SK	is a data sketch. It is a $2m$-dimensional vector.
ComputeGrad	indicates whether to compute the gradient or only the objective function value

Details

The objective function is given as $\|SK - SK(D \cdot A)\|^2$, where SK is a data sketch, $A = \{\alpha_1, \dots, \alpha_n\}$ is a code matrix and $SK(D \cdot A)$ denotes a decomposition sketch, which is defined as: $SK(D \cdot A) = \frac{1}{n}\left[\sum_{i=1}^n \cos(W \cdot D \cdot \alpha_i), \sum_{i=1}^n \sin(W \cdot D \cdot \alpha_i)\right]$. The gradient of this objective function with respect to a dictionary element $d_l \in R^s$ is given as: $-2\left(\nabla_{d_l} SK(D \cdot A)\right)^\top \cdot r$, where $r = SK - SK(D \cdot A)$, $\frac{\delta}{\delta d_l} SK^j(D \cdot A) = 1i \cdot \left(\frac{1}{n}\sum_{i=1}^n A_{li} \cdot \prod_{k=1}^K SK^j(A_{ki} \cdot d_k)\right) \cdot w_j^\top$, and $SK^j(\cdot)$ is the j^{th} coordinate of the sketch vector.

Value

a list

- `grad` is a computed gradient
- `ObjFun` is a objective function value
- `diff` is a vector of the difference between the data sketch and the decomposition sketch

Examples

```
RcppParallel::setThreadOptions(numThreads = 2)
X = matrix(abs(rnorm(n = 1000)), ncol = 100, nrow = 10)
X_fbm = bigstatsr::as_FBM(X)$save()
W = chickn::GenerateFrequencies(Data = X_fbm, m = 64, N0 = ncol(X_fbm),
                               ncores = 1, niter= 3, nblocks = 2, sigma_start = 0.001)$W
SK= chickn::Sketch(X_fbm, W)
```

```
D = X_fbm[, sample(ncol(X_fbm), 10)]
A = sapply(sample(ncol(X_fbm), 5), function(i){
    as.vector(glmnet::glmnet(x = D, y = X_fbm[,i],
              lambda = 0, intercept = FALSE, lower.limits = 0)$beta)})
G = Gradient_D_cpp_parallel(D, A, W, SK)$grad
```

`ObjFun_COMP_cpp` *COMP objective function*

Description

Computation of the objective function from the Compressive Orthogonal Matching Pursuit algorithm.

Usage

```
ObjFun_COMP_cpp(d, W, residue)
```

Arguments

`d`	is a dictionary element
`W`	is a frequency matrix $m \times s$ with frequency vectors in matrix rows.
`residue`	is a residue vector.

Details

The objective function of the Compressive Orthogonal Matching Pursuit is defined as: $-\frac{SK(d) \cdot r}{\|SK(d)\|}$, where $SK(d)$ denotes a sketch of the dictionary element `d` and r is the residue vector, which is defined as the difference between the data sketch SK and the weighted sum of the dictionary elements' sketches, *i.e.* $SK - \sum_{i=1}^{K} \beta_i \cdot SK(d_i)$. This function is involved in `COMP_initialization` routine.

Value

an objective function value

See Also

`COMP_initialization`, `Gradient_COMP_cpp`

Examples

```
X = matrix(abs(rnorm(n = 1000)), ncol = 100, nrow = 10)
X_fbm = bigstatsr::as_FBM(X)$save()
W = chickn::GenerateFrequencies(Data = X_fbm, m = 64, N0 = ncol(X_fbm),
                               ncores = 1, niter= 3, nblocks = 2, sigma_start = 0.001)$W
SK= chickn::Sketch(X_fbm, W)
D = X_fbm[, sample(ncol(X_fbm), 10)]
```

```
weights = sample(10, 10)/10
SK_D = rbind(cos(W%*%D), sin(W%*%D))
d = D[,1]
r = SK - SK_D%*%weights
OF = ObjFun_COMP_cpp(d, W, r)
```

SSDL *SSDL-package*

Description

R package SSDL implements the Sketched Stochastic Dictionary Learning method that builds a dictionary from large-scale data collection by operating on a compressed data version referred to as a data sketch. The chickn package is used to carry out the data compression. SSDL package is designed to handle voluminous data encoded as a matrix, which cannot be loaded in memory. To do this, SSDL package relies on the Filebacked Big Matrix class of bigstatsr package, which allows to access and manipulate matrix-like data stored in files on disk.

Author(s)

Olga Permiakova, Thomas Burger

See Also

CDL

TV_initialization *TV norm dictionary initialization*

Description

Dictionary initialization using a TV norm criterion

Usage

```
TV_initialization(
  Data,
  K,
  cutoff = 0.5,
  Npattern = 8,
  set_size = ncol(Data),
  DoCopies = FALSE,
  ncores = 4,
  DIR_tmp = tempdir()
)
```

Arguments

`Data`	is a Filebacked Big Matrix $s \times N$ with data vectors stored in the matrix columns.
`K`	is a dictionary size.
`cutoff`	is a cut off value, the default value is 0.5.
`Npattern`	is a number of patterns selected in the dataset to create the dictionary
`set_size`	is a maximum size of the set of possible patterns.
`DoCopies`	indicates whether to duplicate patterns.
`ncores`	is a number of cores
`DIR_tmp`	is a directory to save temporary files

Details

The dictionary is initialized by extracting and duplicating patterns with the highest TV norm values To limit the set of possible patterns, only signals with the correlation less then a fixed threshold `cutoff` are taken into account. If the set of possible patterns is too large, it can be further reduced by taking only `set_size` less correlated patterns. The implemented initialization routine can only be applied to positive value data.

Value

a dictionary matrix

Examples

```
X = matrix(abs(rnorm(n = 1000)), ncol = 100, nrow = 10)
X_fbm = bigstatsr::FBM(init = X, ncol = ncol(X), nrow = nrow(X))
D0 = TV_initialization(X_fbm, K = 20, Npattern = 5, DoCopies = TRUE, ncores = 1)
```

Index