Manual of Frovedis Python API

Contents

Chapter 1

Introduction

This manual contains Python API documantation. If you are new to Frovedis, please read the tutorial_python first.

Currently we only provide part of the API documentation. We are still updating the contents.

- Matrix
  - FrovedisDvector
  - FrovedisCRSMatrix
  - FrovedisBlockcyclicMatrix
  - pblas_wrapper
  - scalapack_wrapper
  - getrf_result
  - gesvd_result
- Machine Learning
  - Linear Regression
  - Lasso Regression
  - Ridge Regression
  - Logistic Regression
  - Linear SVM
  - Matrix Factorization using ALS
  - kmeans
  - spectral clustering
  - spectral embedding

Chapter 2

FrovedisDvector

2.1 NAME

FrovedisDvector - A data structure used in modeling the in-memory dvector data of frovedis server side at client python side.

2.2 SYNOPSIS

class frovedis.matrix.dvector.FrovedisDvector(vec=None)

2.2.1 Public Member Functions

load (vec)
load_numpy_array (vec)
debug_print()
release()

2.3 DESCRIPTION

FrovedisDvector is a pseudo data structure at client python side which aims to model the frovedis server side `dvector<double>` (see manual of frovedis dvector for details).

Note that the actual vector data is created at frovedis server side only. Python side FrovedisDvector contains a proxy handle of the in-memory vector data created at frovedis server, along with its size.

2.3.1 Constructor Documentation

2.3.1.1 FrovedisDvector (vec=None)

Parameters
vec: It can be any python array-like object or None. In case of None (Default), it does not make any request to server.

Purpose

This constructor can be used to construct a FrovedisDvector instance, as follows:

```
v1 = FrovedisDvector()          # empty dvector, no server request is made
v2 = FrovedisDvector([1,2,3,4]) # will load data from the given list
```

Return Type

It simply returns "self" reference.

2.3.2 Pubic Member Function Documentation

2.3.2.1 load (vec)

Parameters
vec: It can be any python array-like object (but not None).

Purpose

This function works similar to the constructor. It can be used to load a FrovedisDvector instance, as follows:

```
v = FrovedisDvector().load([1,2,3,4]) # will load data from the given list
```

Return Type

It simply returns "self" reference.

2.3.2.2 load_numpy_array (vec)

Parameters
vec: Any numpy array with values to be loaded in.

Purpose
This function can be used to load a python side numpy array data into frovedis server side dvector. It accepts a python numpy array object and converts it into the frovedis server side dvector whose proxy along size information are stored in the target FrovedisDvector object.

Return Type
It simply returns "self" reference.

2.3.2.3 size()

Purpose
It returns the size of the dvector

Return Type
An integer value containing size of the target dvector.

2.3.2.4 debug_print()

Purpose
It prints the contents of the server side distributed vector data on the server side user terminal. It is mainly useful for debugging purpose.

Return Type
It returns nothing.

2.3.2.5 release()

Purpose
This function can be used to release the existing in-memory data at frovedis server side.

Return Type
It returns nothing.

2.3.2.6 FrovedisDvector.asDvec(vec)

Parameters
vec: A numpy array or python array like object or an instance of FrovedisDvector.

Purpose
This static function is used in order to convert a given array to a dvector. If the input is already an instance of FrovedisDvector, then the same will be returned.

Return Type

An instance of FrovedisDvector.

Chapter 3

FrovedisCRSMatrix

3.1 NAME

FrovedisCRSMatrix - A data structure used in modeling the in-memory crs matrix data of frovedis server side at client python side.

3.2 SYNOPSIS

class frovedis.matrix.sparse.FrovedisCRSMatrix(mat=None)

3.2.1 Public Member Functions

load (mat)
load_scipy_matrix (mat)
load_text (filename)
load_binary (dirname)
save_text (filename)
save_binary (dirname)
debug_print()
release()

3.3 DESCRIPTION

FrovedisCRSMatrix is a pseudo matrix structure at client python side which aims to model the frovedis server side `crs_matrix<double>` (see manual of frovedis crs_matrix for details).

Note that the actual matrix data is created at frovedis server side only. Python side FrovedisCRSMatrix contains a proxy handle of the in-memory matrix data created at frovedis server, along with number of rows and number of columns information.

3.3.1 Constructor Documentation

3.3.1.1 FrovedisCRSMatrix (mat=None)

Parameters
mat: It can be a string containing filename having text data to be loaded, or any scipy sparse matrix or any python array-like object or None. In case of None (Default), it does not make any request to server.

Purpose

This constructor can be used to construct a FrovedisCRSMatrix instance, as follows:

```
mat1 = FrovedisCRSMatrix() # empty matrix, no server request is made
mat2 = FrovedisCRSMatrix("./data") # will load data from given text file
mat3 = FrovedisCRSMatrix([1,2,3,4]) # will load data from the given list
```

Return Type

It simply returns "self" reference.

3.3.2 Pubic Member Function Documentation

3.3.2.1 load (mat)

Parameters
mat: It can be a string containing filename having text data to be loaded, or any scipy sparse matrix or any python array-like object (but it can not be None).

Purpose

This works similar to the constructor.
It can be used to load a FrovedisCRSMatrix instance, as follows:

```
mat1 = FrovedisCRSMatrix().load("./data")  # will load data from given text file
mat2 = FrovedisCRSMatrix().load([1,2,3,4]) # will load data from the given list
```

Return Type

It simply returns "self" reference.

3.3.2.2 load_scipy_matrix (mat)

Parameters
mat: Any scipy matrix with values to be loaded in.

Purpose
This function can be used to load a python side scipy sparse data matrix into frovedis server side crs matrix. It accepts a scipy sparse matrix object and converts it into the frovedis server side crs matrix whose proxy along with number of rows and number of columns information are stored in the target FrovedisCRSMatrix object.

Return Type
It simply returns "self" reference.

3.3.2.3 load_text (filename)

Parameters
filename: A string object containing the text file name to be loaded.

Purpose
This function can be used to load the data from a text file into the target matrix. Note that the file must be placed at server side at the given path and it should have contents stored in libSVM format, i.e., "column_index:value" at each row (see frovedis manual of make_crs_matrix_load() for more details).

Return Type
It simply returns "self" reference.

3.3.2.4 load_binary (dirname)

Parameters
dirname: A string object containing the directory name having the binary data to be loaded.

Purpose
This function can be used to load the data from the specified directory with binary data file into the target matrix. Note that the file must be placed at server side at the given path.

Return Type
It simply returns "self" reference.

3.3.2.5 save_text (filename)

Parameters
filename: A string object containing the text file name in which the data is to be saveed.

Purpose

This function is used to save the target matrix as text file with the filename at the given path. Note that the file will be saved at server side at the given path.

Return Type
It returns nothing.

3.3.2.6 save_binary (dirname)

Parameters
dirname: A string object containing the directory name in which the data is to be saveed as little-endian binary form.

Purpose

This function is used to save the target matrix as little-endian binary file with the filename at the given path. Note that the file will be saved at server side at the given path.

Return Type
It returns nothing.

3.3.2.7 numRows()

Purpose
It returns the number of rows in the matrix

Return Type
An integer value containing rows count in the target matrix.

3.3.2.8 numCols()

Purpose
It returns the number of columns in the matrix

Return Type
An integer value containing columns count in the target matrix.

3.3.2.9 debug_print()

Purpose
It prints the contents of the server side distributed matrix data on the server side user terminal. It is mainly useful for debugging purpose.

Return Type
It returns nothing.

3.3.2.10 release()

Purpose
This function can be used to release the existing in-memory data at frovedis server side.

Return Type
It returns nothing.

3.3.2.11 FrovedisCRSMatrix.asCRS(mat)

Parameters
mat: A scipy matrix, an instance of FrovedisCRSMatrix or any python array-like data.

Purpose
This static function is used in order to convert a given matrix to a crs matrix. If the input is already an instance of FrovedisCRSMatrix, then the same will be returned.

Return Type

An instance of FrovedisCRSMatrix.

Chapter 4

FrovedisBlockcyclicMatrix

4.1 NAME

FrovedisBlockcyclicMatrix - A data structure used in modeling the in-memory blockcyclic matrix data of frovedis server side at client python side.

4.2 SYNOPSIS

class frovedis.matrix.dense.FrovedisBlockcyclicMatrix(mat=None)

4.2.1 Overloaded Operators

operator= (mat)
operator+ (mat)
operator- (mat)
operator* (mat)
operator~ (mat)

4.2.2 Public Member Functions

load (mat)
load_numpy_matrix (mat)
load_text (filename)
load_binary (dirname)
save_text (filename)
save_binary (dirname)
transpose()
to_numpy_matrix ()
debug_print()
release()

4.3 DESCRIPTION

FrovedisBlockcyclicMatrix is a pseudo matrix structure at client python side which aims to model the frovedis server side `blockcyclic_matrix<double>` (see manual of frovedis blockcyclic_matrix for details).

Note that the actual matrix data is created at frovedis server side only. Python side FrovedisBlockcyclicMatrix contains a proxy handle of the in-memory matrix data created at frovedis server, along with number of rows and number of columns information.

4.3.1 Constructor Documentation

4.3.1.1 FrovedisBlockcyclicMatrix (mat=None)

Parameters
mat: It can be a string containing filename having text data to be loaded, or another FrovedisBlockcyclicMatrix instance for copy or any python array-like object or None. In case of None (Default), it does not make any request to server.

Purpose

This constructor can be used to construct a FrovedisBlockcyclicMatrix instance, as follows:

```
mat1 = FrovedisBlockcyclicMatrix() # empty matrix, no server request is made
mat2 = FrovedisBlockcyclicMatrix("./data") # will load data from given text file
mat3 = FrovedisBlockcyclicMatrix(mat2) # copy constructor
mat4 = FrovedisBlockcyclicMatrix([1,2,3,4]) # will load data from the given list
```

Return Type

It simply returns “self” reference.

4.3.2 Overloaded Operators Documentation

4.3.2.1 operator= (mat)

Parameters
mat: An existing FrovedisBlockcyclicMatrix instance to be copied.

Purpose
It can be used to copy the input matrix in the target matrix. It returns a self reference to support operator chaining.

For example,

```
m1 = FrovedisBlockcyclicMatrix([1,2,3,4])
m2 = m1 (copy operatror)
m3 = m2 = m1
```

Return Type
It returns “self” reference.

4.3.2.2 operator+ (mat)

Parameters
mat: An instance of FrovedisBlockcyclicMatrix or an array-like structure.

Purpose
It can be used to perform addition between two blockcyclic matrices. If the input data is not a FrovedisBlockcyclicMatrix instance, internally it will get converted into a FrovedisBlockcyclicMatrix instance first and then that will be added with the source matrix.

For example,

```
m1 = FrovedisBlockcyclicMatrix([1,2,3,4])
m2 = FrovedisBlockcyclicMatrix([1,2,3,4])
m3 = m2 + m1
```

Return Type
It returns the resultant matrix of the type FrovedisBlockcyclicMatrix.

4.3.2.3 operator- (mat)

Parameters
mat: An instance of FrovedisBlockcyclicMatrix or an array-like structure.

Purpose
It can be used to perform subtraction between two blockcyclic matrices. If the input data is not a FrovedisBlockcyclicMatrix instance, internally it will get converted into a FrovedisBlockcyclicMatrix instance first and then that will be subtracted from the source matrix.

For example,

```
m1 = FrovedisBlockcyclicMatrix([1,2,3,4])
m2 = FrovedisBlockcyclicMatrix([1,2,3,4])
m3 = m2 - m1
```

Return Type
It returns the resultant matrix of the type FrovedisBlockcyclicMatrix.

4.3.2.4 operator* (mat)

Parameters
mat: An instance of FrovedisBlockcyclicMatrix or an array-like structure.

Purpose
It can be used to perform multiplication between two blockcyclic matrices. If the input data is not a FrovedisBlockcyclicMatrix instance, internally it will get converted into a FrovedisBlockcyclicMatrix instance first and then that will be multiplied with the source matrix.

For example,

```
m1 = FrovedisBlockcyclicMatrix([1,2,3,4])
m2 = FrovedisBlockcyclicMatrix([1,2,3,4])
m3 = m2 * m1
```

Return Type
It returns the resultant matrix of the type FrovedisBlockcyclicMatrix.

4.3.2.5 operator~ ()

Purpose
It can be used to obtain transpose of the target matrix. If the input data is not a FrovedisBlockcyclicMatrix instance, internally it will get converted into a FrovedisBlockcyclicMatrix instance first and then the transpose will get computed.

For example,

```
m1 = FrovedisBlockcyclicMatrix([1,2,3,4])
m2 = ~m1
```

Return Type
It returns the resultant matrix of the type FrovedisBlockcyclicMatrix.

4.3.3 Pubic Member Function Documentation

4.3.3.1 load (mat)

Parameters
mat: It can be a string containing filename having text data to be loaded, or another FrovedisBlockcyclicMatrix instance for copy or any python array-like object (but it can not be None).

Purpose

This function works similar to the constructor. It can be used to load a FrovedisBlockcyclicMatrix instance, as follows:

```
mat1 = FrovedisBlockcyclicMatrix().load("./data")  # will load data from given text file
mat2 = FrovedisBlockcyclicMatrix().load(mat1)      # copy operation
mat3 = FrovedisBlockcyclicMatrix().load([1,2,3,4]) # will load data from the given list
```

Return Type

It simply returns "self" reference.

4.3.3.2 load_numpy_matrix (mat)

Parameters
mat: A numpy matrix with values to be loaded in.

Purpose
This function can be used to load a python side dense data matrix into a frovedis server side blockcyclic matrix. It accepts a numpy matrix object and converts it into the frovedis server side blockcyclic matrix whose proxy along with number of rows and number of columns information are stored in the target FrovedisBlockcyclicMatrix object.

Return Type
It simply returns "self" reference.

4.3.3.3 load_text (filename)

Parameters
filename: A string object containing the text file name to be loaded.

Purpose
This function can be used to load the data from a text file into the target matrix. Note that the file must be placed at server side at the given path.

Return Type
It simply returns "self" reference.

4.3.3.4 load_binary (dirname)

Parameters
dirname: A string object containing the directory name having the binary data to be loaded.

Purpose
This function can be used to load the data from the specified directory with binary data file into the target matrix. Note that the file must be placed at server side at the given path.

Return Type
It simply returns "self" reference.

4.3.3.5 save_text (filename)

Parameters
filename: A string object containing the text file name in which the data is to be saved.

Purpose

This function is used to save the target matrix as text file with the filename at the given path. Note that the file will be saved at server side at the given path.

Return Type
It returns nothing.

4.3.3.6 save_binary (dirname)

Parameters
dirname: A string object containing the directory name in which the data is to be saved as little-endian binary form.

Purpose

This function is used to save the target matrix as little-endian binary file with the filename at the given path. Note that the file will be saved at server side at the given path.

Return Type
It returns nothing.

4.3.3.7 transpose ()

Purpose

This function will compute the transpose of the given matrix.

Return Type
It returns the transposed blockcyclic matrix of the type FrovedisBlockcyclicMatrix.

4.3.3.8 to_numpy_matrix ()

Purpose
This function is used to convert the target blockcyclic matrix into numpy matrix.
Note that this function will request frovedis server to gather the distributed data, and send back that data in the rowmajor array form and the python client will then convert the received numpy array from frovedis server to python numpy matrix.

Return Type
It returns a two-dimensional dense numpy matrix

4.3.3.9 numRows()

Purpose
It returns the number of rows in the matrix

Return Type
An integer value containing rows count in the target matrix.

4.3.3.10 numCols()

Purpose
It returns the number of columns in the matrix

Return Type
An integer value containing columns count in the target matrix.

4.3.3.11 debug_print()

Purpose
It prints the contents of the server side distributed matrix data on the server side user terminal. It is mainly useful for debugging purpose.

Return Type
It returns nothing.

4.3.3.12 release()

Purpose
This function can be used to release the existing in-memory data at frovedis server side.

Return Type
It returns nothing.

4.3.3.13 FrovedisBlockcyclicMatrix.asBCM(mat)

Parameters
mat: An instance of FrovedisBlockcyclicMatrix or any python array-like structure.

Purpose
This static function is used in order to convert a given matrix to a blockcyclic matrix. If the input is already an instance of FrovedisBlockcyclicMatrix, then the same will be returned.

Return Type

An instance of FrovedisBlockcyclicMatrix.

Chapter 5

pblas_wrapper

5.1 NAME

pblas_wrapper - a frovedis module provides user-friendly interfaces for commonly used pblas routines in scientific applications like machine learning algorithms.

5.2 SYNOPSIS

import frovedis.matrix.wrapper.PBLAS

5.2.1 Public Member Functions

PBLAS.swap (v1, v2)
PBLAS.copy (v1, v2)
PBLAS.scal (v, al)
PBLAS.axpy (v1, v2, al=1.0)
PBLAS.dot (v1, v2)
PBLAS.nrm2 (v)
PBLAS.gemv (m, v1, v2, trans=False, al=1.0, b2=0.0)
PBLAS.ger (v1, v2, m, al=1.0)
PBLAS.gemm (m1, m2, m3, trans_m1=False, trans_m2=False, al=1.0, be=0.0)
PBLAS.geadd (m1, m2, trans=False, al=1.0, be=1.0)

5.3 DESCRIPTION

PBLAS is a high-performance scientific library written in Fortran language. It provides rich set of functionalities on vectors and matrices. The computation loads of these functionalities are parallelized over the available processes in a system and the user interfaces of this library is very detailed and complex in nature. It requires a strong understanding on each of the input parameters, along with some distribution concepts.

Frovedis provides a wrapper module for some commonly used PBLAS subroutines in scientific applications like machine learning algorithms. These wrapper interfaces are very simple and user needs not to consider all the detailed distribution parameters. Only specifying the target vectors or matrices with some other parameters (depending upon need) are fine. At the same time, all the use cases of a PBLAS routine can also be performed using Frovedis PBLAS wrapper of that routine.

This python module implements a client-server application, where the python client can send the python matrix data to frovedis server side in order to create blockcyclic matrix at frovedis server and then python client can request frovedis server for any of the supported PBLAS operation on that matrix. When required, python client can request frovedis server to send back the resultant matrix and it can then create equivalent python data.

The individual detailed descriptions can be found in the subsequent sections. Please note that the term "inout", used in the below section indicates a function argument as both "input" and "output".

5.3.1 Detailed Description

5.3.1.1 swap (v1, v2)

Parameters
v1: A FrovedisBlockcyclicMatrix with single column (inout)
v2: A FrovedisBlockcyclicMatrix with single column (inout)

Purpose
It will swap the contents of v1 and v2, if they are semantically valid and are of same length.

Return Value
On success, it returns nothing. If any error occurs, it throws an exception.

5.3.1.2 copy (v1, v2)

Parameters
v1: A FrovedisBlockcyclicMatrix with single column (input)
v2: A FrovedisBlockcyclicMatrix with single column (output)

Purpose
It will copy the contents of v1 in v2 (v2 = v1), if they are semantically valid and are of same length.

Return Value
On success, it returns nothing. If any error occurs, it throws an exception.

5.3.1.3 scal (v, al)

Parameters
v: A FrovedisBlockcyclicMatrix with single column (inout)
al: A double parameter to specify the value to which the input vector needs to be scaled. (input)

Purpose
It will scale the input vector with the provided "al" value, if it is semantically valid. On success, input vector "v" would be updated (in-place scaling).

Return Value
On success, it returns nothing. If any error occurs, it throws an exception.

5.3.1.4 axpy (v1, v2, al=1.0)

Parameters
v1: A FrovedisBlockcyclicMatrix with single column (input)
v2: A FrovedisBlockcyclicMatrix with single column (inout)
al: A double parameter to specify the value to which "v1" needs to be scaled (not in-place scaling) [Default: 1.0] (input/optional)

Purpose
It will solve the expression v2 = al*v1 + v2, if the input vectors are semantically valid and are of same length. On success, "v2" will be updated with desired result, but "v1" would remain unchanged.

Return Value
On success, it returns nothing. If any error occurs, it throws an exception.

5.3.1.5 dot (v1, v2)

Parameters
v1: A FrovedisBlockcyclicMatrix with single column (input)
v2: A FrovedisBlockcyclicMatrix with single column (input)

Purpose
It will perform dot product of the input vectors, if they are semantically valid and are of same length. Input vectors would not get modified during the operation.

Return Value
On success, it returns the dot product result of the type double. If any error occurs, it throws an exception.

5.3.1.6 nrm2 (v)

Parameters
v: A FrovedisBlockcyclicMatrix with single column (input)

Purpose
It will calculate the norm of the input vector, if it is semantically valid. Input vector would not get modified during the operation.

Return Value
On success, it returns the norm value of the type double. If any error occurs, it throws an exception.

5.3.1.7 gemv (m, v1, v2, trans=False, al=1.0, be=0.0)

Parameters
m: A FrovedisBlockcyclicMatrix (input)
v1: A FrovedisBlockcyclicMatrix with single column (input)
v2: A FrovedisBlockcyclicMatrix with single column (inout)
trans: A boolean value to specify whether to transpose "m" or not [Default: False] (input/optional)
al: A double type value [Default: 1.0] (input/optional)
be: A double type value [Default: 0.0] (input/optional)

Purpose
The primary aim of this routine is to perform simple matrix-vector multiplication.
But it can also be used to perform any of the below operations:

```
(1) v2 = al*m*v1 + be*v2
(2) v2 = al*transpose(m)*v1 + be*v2
```

If trans=False, then expression (1) is solved. In that case, the size of "v1" must be at least the number of columns in "m" and the size of "v2" must be at least the number of rows in "m".
If trans=True, then expression (2) is solved. In that case, the size of "v1" must be at least the number of rows in "m" and the size of "v2" must be at least the number of columns in "m".

Since "v2" is used as input-output both, memory must be allocated for this vector before calling this routine, even if simple matrix-vector multiplication is required. Otherwise, this routine will throw an exception.

For simple matrix-vector multiplication, no need to specify values for the input parameters “trans”, “al” and “be” (leave them at their default values).

On success, “v2” will be overwritten with the desired output. But “m” and “v1” would remain unchanged.

Return Value
On success, it returns nothing. If any error occurs, it throws an exception.

5.3.1.8 ger (v1, v2, m, al=1.0)

Parameters
v1: A FrovedisBlockcyclicMatrix with single column (input)
v2: A FrovedisBlockcyclicMatrix with single column (input)
m: A FrovedisBlockcyclicMatrix (inout)
al: A double type value [Default: 1.0] (input/optional)

Purpose
The primary aim of this routine is to perform simple vector-vector multiplication of the sizes “a” and “b” respectively to form an axb matrix. But it can also be used to perform the below operations:

```
m = al*v1*v2' + m
```

This operation can only be performed if the inputs are semantically valid and the size of “v1” is at least the number of rows in matrix “m” and the size of “v2” is at least the number of columns in matrix “m”.

Since “m” is used as input-output both, memory must be allocated for this matrix before calling this routine, even if simple vector-vector multiplication is required. Otherwise it will throw an exception.

For simple vector-vector multiplication, no need to specify the value for the input parameter “al” (leave it at its default value).

On success, “m” will be overwritten with the desired output. But “v1” and “v2” will remain unchanged.

Return Value
On success, it returns nothing. If any error occurs, it throws an exception.

5.3.1.9 gemm (m1, m2, m3, trans_m1=False, trans_m2=False, al=1.0, be=0.0)

Parameters
m1: A FrovedisBlockcyclicMatrix (input)
m2: A FrovedisBlockcyclicMatrix (input)
m3: A FrovedisBlockcyclicMatrix (inout)
trans_m1: A boolean value to specify whether to transpose “m1” or not [Default: False] (input/optional)
trans_m2: A boolean value to specify whether to transpose “m2” or not [Default: False] (input/optional)
al: A double type value [Default: 1.0] (input/optional)
be: A double type value [Default: 0.0] (input/optional)

Purpose
The primary aim of this routine is to perform simple matrix-matrix multiplication.
But it can also be used to perform any of the below operations:

```
(1) m3 = al*m1*m2 + be*m3
(2) m3 = al*transpose(m1)*m2 + be*m3
(3) m3 = al*m1*transpose(m2) + be*m3
(4) m3 = al*transpose(m1)*transpose(m2) + be*m3
```

(1) will be performed, if both “trans_m1” and “trans_m2” are False.

(2) will be performed, if trans_m1=True and trans_m2 = False.

(3) will be performed, if trans_m1=False and trans_m2 = True.

(4) will be performed, if both “trans_m1” and “trans_m2” are True.

If we have four variables nrowa, nrowb, ncola, ncolb defined as follows:

```
if(trans_m1) {
  nrowa = number of columns in m1
  ncola = number of rows in m1
}
else {
  nrowa = number of rows in m1
  ncola = number of columns in m1
}

if(trans_m2) {
  nrowb = number of columns in m2
  ncolb = number of rows in m2
}
else {
  nrowb = number of rows in m2
  ncolb = number of columns in m2
}
```

Then this function can be executed successfully, if the below conditions are all true:

```
(a) "ncola" is equal to "nrowb"
(b) number of rows in "m3" is equal to or greater than "nrowa"
(b) number of columns in "m3" is equal to or greater than "ncolb"
```

Since “m3” is used as input-output both, memory must be allocated for this matrix before calling this routine, even if simple matrix-matrix multiplication is required. Otherwise it will throw an exception.

For simple matrix-matrix multiplication, no need to specify the value for the input parameters “trans_m1”, “trans_m2”, “al”, “be” (leave them at their default values).

On success, “m3” will be overwritten with the desired output. But “m1” and “m2” will remain unchanged.

Return Value
On success, it returns nothing. If any error occurs, it throws an exception.

5.3.1.10 geadd (m1, m2, trans=False, al=1.0, be=1.0)

Parameters
m1: A FrovedisBlockcyclicMatrix (input)
m2: A FrovedisBlockcyclicMatrix (inout)
trans: A boolean value to specify whether to transpose “m1” or not [Default: False] (input/optional)
al: A double type value [Default: 1.0] (input/optional)
be: A double type value [Default: 1.0] (input/optional)

Purpose
The primary aim of this routine is to perform simple matrix-matrix addition. But it can also be used to perform any of the below operations:

```
(1) m2 = al*m1 + be*m2
(2) m2 = al*transpose(m1) + be*m2
```

If trans=False, then expression (1) is solved. In that case, the number of rows and the number of columns in "m1" should be equal to the number of rows and the number of columns in "m2" respectively.
If trans=True, then expression (2) is solved. In that case, the number of columns and the number of rows in "m1" should be equal to the number of rows and the number of columns in "m2" respectively.

If it is needed to scale the input matrices before the addition, corresponding "al" and "be" values can be provided. But for simple matrix-matrix addition, no need to specify values for the input parameters "trans", "al" and "be" (leave them at their default values).

On success, "m2" will be overwritten with the desired output. But "m1" would remain unchanged.

Return Value
On success, it returns nothing. If any error occurs, it throws an exception.

5.4 SEE ALSO

scalapack_wrapper, blockcyclic_matrix

Chapter 6

scalapack_wrapper

6.1 NAME

scalapack_wrapper - a frovedis module provides user-friendly interfaces for commonly used scalapack routines in scientific applications like machine learning algorithms.

6.2 SYNOPSIS

import frovedis.matrix.wrapper.SCALAPACK

6.3 WRAPPER FUNCTIONS

SCALAPACK.getrf (m)
SCALAPACK.getri (m, ipivPtr)
SCALAPACK.getrs (m1, m2, ipivPtr, trans=False)
SCALAPACK.gesv (m1, m2)
SCALAPACK.gels (m1, m2, trans=False)
SCALAPACK.gesvd (m, wantU=False, wantV=False)

6.4 DESCRIPTION

ScaLAPACK is a high-performance scientific library written in Fortran language. It provides rich set of linear algebra functionalities whose computation loads are parallelized over the available processes in a system and the user interfaces of this library is very detailed and complex in nature. It requires a strong understanding on each of the input parameters, along with some distribution concepts.

Frovedis provides a wrapper module for some commonly used ScaLAPACK subroutines in scientific applications like machine learning algorithms. These wrapper interfaces are very simple and user needs not to consider all the detailed distribution parameters. Only specifying the target vectors or matrices with some other parameters (depending upon need) are fine. At the same time, all the use cases of a ScaLAPACK routine can also be performed using Frovedis ScaLAPACK wrapper of that routine.

This python module implements a client-server application, where the python client can send the python matrix data to frovedis server side in order to create blockcyclic matrix at frovedis server and then python

client can request frovedis server for any of the supported ScaLAPACK operation on that matrix. When required, python client can request frovedis server to send back the resultant matrix and it can then create equivalent python data (see manuals for FrovedisBlockcyclicMatrix to python data conversion).

The individual detailed descriptions can be found in the subsequent sections. Please note that the term "inout", used in the below section indicates a function argument as both "input" and "output".

6.4.1 Detailed Description

6.4.1.1 getrf (m)

Parameters
m: A FrovedisBlockcyclicMatrix (inout)

Purpose
It computes an LU factorization of a general M-by-N distributed matrix, "m" using partial pivoting with row interchanges.

On successful factorization, matrix "m" is overwritten with the computed L and U factors. Along with the return status of native scalapack routine, it also returns the proxy address of the node local vector "ipiv" containing the pivoting information associated with input matrix "m" in the form of GetrfResult. The "ipiv" information will be useful in computation of some other routines (like getri, getrs etc.)

Return Value
On success, it returns the object of the type GetrfResult as explained above. If any error occurs, it throws an exception explaining cause of the error.

6.4.1.2 getri (m, ipivPtr)

Parameters
m: A FrovedisBlockcyclicMatrix (inout)
ipiv: A long object containing the proxy of the ipiv vector (from GetrfResult) (input)

Purpose
It computes the inverse of a distributed square matrix using the LU factorization computed by getrf(). So in order to compute inverse of a matrix, first compute it's LU factor (and ipiv information) using getrf() and then pass the factored matrix, "m" along with the "ipiv" information to this function.

On success, factored matrix "m" is overwritten with the inverse (of the matrix which was passed to getrf()) matrix. "ipiv" will be internally used by this function and will remain unchanged.

For example,

```
res = SCALAPACK.getrf(m)      // getting LU factorization of "m"
SCALAPACK.getri(m,res.ipiv()) // "m" will have inverse of the initial value
```

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

6.4.1.3 getrs (m1, m2, ipiv, trans=False)

Parameters
m1: A FrovedisBlockcyclicMatrix (input)
m2: A FrovedisBlockcyclicMatrix (inout)

ipiv: A long object containing the proxy of the ipiv vector (from GetrfResult) (input)
trans: A boolean value to specify whether to transpose "m1" [Default: False] (input/optional)

Purpose
It solves a real system of distributed linear equations, AX=B with a general distributed square matrix (A) using the LU factorization computed by getrf(). Thus before calling this function, it is required to obtain the factored matrix "m1" (along with "ipiv" information) by calling getrf().

For example,

```
res = SCALAPACK.getrf(m1) // getting LU factorization of "m1"
SCALAPACK.getrs(m1,m2,res.ipiv())
```

If trans=False, the linear equation AX=B is solved.
If trans=True, the linear equation transpose(A)X=B (A'X=B) is solved.

The matrix "m2" should have number of rows >= the number of rows in "m1" and at least 1 column in it.

On entry, "m2" contains the distributed right-hand-side (B) of the equation and on successful exit it is overwritten with the distributed solution matrix (X).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

6.4.1.4 gesv (m1, m2)

Parameters
m1: A FrovedisBlockcyclicMatrix (inout)
m2: A FrovedisBlockcyclicMatrix (inout)

Purpose
It solves a real system of distributed linear equations, AX=B with a general distributed square matrix, "m1" by computing it's LU factors internally. This function internally computes the LU factors and ipiv information using getrf() and then solves the equation using getrs().

The matrix "m2" should have number of rows >= the number of rows in "m1" and at least 1 column in it.

On entry, "m1" contains the distributed left-hand-side square matrix (A), "m2" contains the distributed right-hand-side matrix (B) and on successful exit "m1" is overwritten with it's LU factors, "m2" is overwritten with the distributed solution matrix (X).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

6.4.1.5 gels (m1, m2, trans=False)

Parameters
m1: A FrovedisBlockcyclicMatrix (input)
m2: A FrovedisBlockcyclicMatrix (inout)
trans: A boolean value to specify whether to transpose "m1" [Default: False] (input/optional)

Purpose
It solves overdetermined or underdetermined real linear systems involving an M-by-N distributed matrix (A) or its transpose, using a QR or LQ factorization of (A). It is assumed that distributed matrix (A) has full rank.

If trans=False and M >= N: it finds the least squares solution of an overdetermined system.
If trans=False and M < N: it finds the minimum norm solution of an underdetermined system.
If trans=True and M >= N: it finds the minimum norm solution of an underdetermined system.
If trans=True and M < N: it finds the least squares solution of an overdetermined system.

The matrix "m2" should have number of rows >= max(M,N) and at least 1 column.

On entry, "m1" contains the distributed left-hand-side matrix (A) and "m2" contains the distributed right-hand-side matrix (B). On successful exit, "m1" is overwritten with the QR or LQ factors and "m2" is overwritten with the distributed solution matrix (X).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

6.4.1.6 gesvd (m, wantU=False, wantV=False)

Parameters
m: A FrovedisBlockcyclicMatrix (inout)
wantU: A boolean value to specify whether to compute U matrix [Default: False] (input)
wantV: A boolean value to specify whether to compute V matrix [Default: False] (input)

Purpose
It computes the singular value decomposition (SVD) of an M-by-N distributed matrix.

On entry "m" contains the distributed matrix whose singular values are to be computed.

If wantU = wantV = False, then it computes only the singular values in sorted oder, so that sval(i) >= sval(i+1). Otherwise it also computes U and/or V (left and right singular vectors respectively) matrices.

On successful exit, the contents of "m" is destroyed (internally used as workspace).

Return Value
On success, it returns the object of the type GesvdResult containing the singular values and U and V components (based on the requirement) along with the exit status of the native scalapack routine. If any error occurs, it throws an exception explaining cause of the error.

6.5 SEE ALSO

blockcyclic_matrix, pblas_wrapper, arpack_wrapper, getrf_result, gesvd_result

Chapter 7

getrf_result

7.1 NAME

getrf_result - a structure to model the output of frovedis wrapper of scalapack getrf routine.

7.2 SYNOPSIS

import frovedis.matrix.results.GetrfResult

7.2.1 Public Member Functions

release()
ipiv()
stat()

7.3 DESCRIPTION

GetrfResult is a client python side pseudo result structure containing the proxy of the in-memory scalapack getrf result (node local ipiv vector) created at frovedis server side.

7.3.1 Public Member Function Documentation

7.3.1.1 release()

Purpose
This function can be used to release the in-memory result component (ipiv vector) at frovedis server.

Return Type
It returns nothing.

7.3.1.2 ipiv()

Purpose
This function returns the proxy of the node_local "ipiv" vector computed during getrf calculation. This value will be required in other scalapack routine calculation, like getri, getrs etc.

Return Type
A long value containing the proxy of ipiv vector.

7.3.1.3 stat()

Purpose
This function returns the exit status of the scalapack native getrf routine on calling of which the target result object was obtained.

Return Type
It returns an integer value.

Chapter 8

gesvd_result

8.1 NAME

gesvd_result - a structure to model the output of frovedis singular value decomposition methods.

8.2 SYNOPSIS

import frovedis.matrix.results.GesvdResult

8.2.1 Public Member Functions

to_numpy_results()
save(svec, umat=None, vmat=None)
save_binary(svec, umat=None, vmat=None)
load (svec, umat=None, vmat=None, mtype='B')
load_binary (svec, umat=None, vmat=None, mtype='B')
debug_print()
release()
stat()
getK()

8.3 DESCRIPTION

GesvdResult is a python side pseudo result structure containing the proxies of the in-memory SVD results created at frovedis server side. It can be used to convert the frovedis side SVD result to python equivalent data structures.

8.3.1 Public Member Function Documentation

8.3.1.1 to_numpy_results()

Purpose
This function can be used to convert the frovedis side SVD results to python numpy result structures.

If U and V both are computed, it returns: (numpy matrix, numpy array, numpy matrix)
indicating (umatrix, singular vector, vmatrix).

When U is calculated, but not V, it returns: (numpy matrix, numpy array, None)
When V is calculated, but not U, it returns: (None, numpy array, numpy matrix)
When neither U nor V is calculated, it returns: (None, numpy array, None)

Return Type
It returns a tuple as explained above.

8.3.1.2 save(svec, umat=None, vmat=None)

Parameters
svec: A string object containing name of the file to save singular vectors as text data. (mandatory)
umat: A string object containing name of the file to save umatrix as text data. (optional)
vmat: A string object containing name of the file to save vmatrix as text data. (ptional)

Purpose
This function can be used to save the result values in different text files at server side. If saving of U and V components are not required, "umat" and "vmat" can be None, but "svec" should have a valid filename.

Return Type
It returns nothing.

8.3.1.3 save_binary (svec, umat=None, vmat=None)

Parameters
svec: A string object containing name of the file to save singular vectors as binary data. (mandatory)
umat: A string object containing name of the file to save umatrix as binary data. (optional)
vmat: A string object containing name of the file to save vmatrix as binary data. (optional)

Purpose
This function can be used to save the result values in different files as little-endian binary data at server side. If saving of U and V components are not required, "umat" and "vmat" can be None, but "svec" should have a valid filename.

Return Type
It returns nothing.

8.3.1.4 load(svec, umat=None, vmat=None, mtype='B')

Parameters
svec: A string object containing name of the file from which to load singular vectors as text data for the target result. (mandatory)
umat: A string object containing name of the file from which to load umatrix as text data for the target result. (optional)
vmat: A string object containing name of the file from which to load vmatrix as text data for the target result. (optional)
mtype: A character value, can be either 'B' or 'C'. (optional)

Purpose
This function can be used to load the result values in different text files at server side. If loading of U and V components are not required, "umat" and "vmat" can be None, but "svec" should have a valid filename.

If mtype = 'B' and umat/vmat is to be loaded, then they will be loaded as blockcyclic matrices at server side.

If mtype = 'C' and umat/vmat is to be loaded, then they will be loaded as colmajor matrices at server side.

Return Type
It returns nothing.

8.3.1.5 load_binary(svec, umat=None, vmat=None, mtype='B')

Parameters
svec: A string object containing name of the file from which to load singular vectors as binary data for the target result. (mandatory)
umat: A string object containing name of the file from which to load umatrix as binary data for the target result. (optional)
vmat: A string object containing name of the file from which to load vmatrix as binary data for the target result. (optional)
mtype: A character value, can be either 'B' or 'C'. (optional)

Purpose
This function can be used to load the result values in different little-endian binary files at server side. If loading of U and V components are not required, "umat" and "vmat" can be None, but "svec" should have a valid filename.

If mtype = 'B' and umat/vmat is to be loaded, then they will be loaded as blockcyclic matrices at server side.

If mtype = 'C' and umat/vmat is to be loaded, then they will be loaded as colmajor matrices at server side.

Return Type
It returns nothing.

8.3.1.6 debug_print()

Purpose
This function can be used to print the result components at server side user terminal. This is useful in debugging purpose.

Return Type
It returns nothing.

8.3.1.7 release()

Purpose
This function can be used to release the in-memory result components at frovedis server.

Return Type
It returns nothing.

8.3.1.8 stat()

Purpose
This function returns the exit status of the scalapack native gesvd routine on calling of which the target result object was obtained.

Return Type
An integer value.

8.3.1.9 getK()

Purpose
This function returns the number of singular values computed.

Return Type
An integer value.

Chapter 9

Linear Regression

9.1 NAME

Linear Regression - A regression algorithm to predict the continuous output without any regularization.

9.2 SYNOPSIS

class frovedis.mllib.linear_model.LinearRegression(fit_intercept=True, normalize=False,
copy_X=True, n_jobs=None,
max_iter=None, tol=0.0001,
lr_rate=1e-8, solver=None,
verbose=0, warm_start = False)

9.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
score(X, y, sample_weight = None)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

9.3 DESCRIPTION

Linear least squares is the most common formulation for regression problems. It is a linear method with the loss function given by the **squared loss**:

```
L(w;x,y) := 1/2(wTx-y)^2
```

Where the vectors x are the training data examples and y are their corresponding labels which we want to predict. w is the linear model (also known as weight) which uses a single weighted sum of features to

make a prediction. The method is called linear since it can be expressed as a function of wTx and y. Linear Regression does not use any regularizer.

The gradient of the squared loss is: (wTx-y).x

Frovedis provides implementation of linear regression with the following optimizers:
(1) stochastic gradient descent with minibatch
(2) LBFGS optimizer
(3) least-square

The simplest method to solve optimization problems of the form **min f(w)** is gradient descent. Such first-order optimization methods well-suited for large-scale and distributed computation. Whereas, L-BFGS is an optimization algorithm in the family of quasi-Newton methods to solve the optimization problems of the similar form.

Like the original BFGS, L-BFGS (Limited Memory BFGS) uses an estimation to the inverse Hessian matrix to steer its search through feature space, but where BFGS stores a dense nxn approximation to the inverse Hessian (n being the number of features in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. L-BFGS often achieves rapider convergence compared with other first-order optimization.

For least-square solver, we have used LAPACK routine "gelsd" and ScaLAPACK routine "gels" when input data is dense in nature. For the sparse-input we have provided a least-square implementation, similar to scipy.sparse.linalg.lsqr.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis interface is almost same as Scikit-learn Linear Regression interface, but it doesn't have any dependency with Scikit-learn. It can be used simply even if the system doesn't have Scikit-learn installed. Thus in this implementation, a python client can interact with a frovedis server by sending the required python data for training at frovedis side. Python data is converted into frovedis compatible data internally and the python ML call is linked with the respective frovedis ML call to get the job done at frovedis server.

Python side calls for Linear Regression on the frovedis server. Once the training is completed with the input data at the frovedis server, it returns an abstract model with a unique model ID to the client python program.

When prediction-like request would be made on the trained model, python program will send the same request to the frovedis server. After the request is served at the frovedis server, the output would be sent back to the python client.

9.3.1 Detailed Description

9.3.1.1 LinearRegression()

Parameters
fit_intercept: A boolean parameter specifying whether a constant(intercept) should be added to the decision function. (Default: True)
normalize: A boolean parameter. (unused)
copy_X: A boolean parameter. (unused)
n_jobs: An integer parameter. (unused)
max_iter: A positive integer value used to set the maximum number of iterations. When it is None(not specified explicitly), it will be set as 1000 for "sag", "lbfgs", "lapack" and "scalapack" solvers and for "sparse_lsqr" solver, it will be 2 * (n_features). (Default: None)
tol: Zero or a positive value of double(float64) type specifying the convergence tolerance value. (Default: 0.001)
lr_rate: A positive value of double(float64) type containing the learning rate. (Default: 1e-8)
solver: A string parameter specifying the solver to use. (Default: None). In case it is None (not explicitly specified), the value will be set to "lapack" when dense input matrix (X) is provided and for sparse input

matrix (X), it is set as "sparse_lsqr". Frovedis supports "sag", "lbfgs", "lapack", "scalapack", "sparse_lsqr" solvers.
"lapack" and "scalapack" solvers can only work with dense data.
"sparse_lsqr" solver can only work with sparse data.
verbose: An integer parameter specifying the log level to use. Its value is 0 by default(for INFO mode and not specified explicitly). But it can be set to 1 (for DEBUG mode) or 2 (for TRACE mode) for getting training time logs from frovedis server.
warm_start: A boolean parameter which when set to True, reuses the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. Only supported by "sag" and "lbfgs" solvers. (Default: False)

Attributes
coef_: It is a python ndarray(containing float or double(float64) typed values depending on data-type of input matrix (X)) of estimated coefficients for the linear regression problem. It has shape (n_features,).
rank_: An integer value used to store rank of matrix (X). It is only available when matrix (X) is dense and "lapack" solver is used.
singular_: It is a python ndarray(contaning float or double(float64) typed values depending on data-type of input matrix (X)) and of shape(min(X,y),) which is used to store singular values of X. It is only available when X is dense and "lapack" solver is used.
intercept(bias)_: It is a python ndarray(contaning float or double(float64) typed values depending on data-type of input matrix (X)). If fit_intercept is set to False, the intercept is set to zero. It has shape (1,).
n_iter_: A positive integer value used to get the actual iteration point at which the problem is converged. It is only available for "sag", "lbfgs" and "sparse-lsqr" solvers.

Purpose
It initializes a Linear Regression object with the given parameters.

The parameters: "normalize", "copy_X" and "n_jobs" are simply kept in to to make the interface uniform to the Scikit-learn Linear Regression module. They are not used anywhere within frovedis implementation.

"solver" can be "sag" for frovedis side stochastic gradient descent, "lbfgs" for frovedis side LBFGS optimizer, "sparse_lsqr", "lapack" and "scalapack" when optimizing the linear regression model.

"max_iter" can be used to set the maximum interations to achieve the convergence. In case the convergence is not achieved, it displays a warning for the same.

Return Value
It simply returns "self" reference.

9.3.1.2 fit(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisColmajorMatrix for dense data.
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of (nsamples,). When it is None (not specified), an uniform weight vector is assigned on each input sample.

Purpose
It accepts the training feature matrix (X) and corresponding output labels (y) as inputs from the user and trains a linear regression model with those data at frovedis server.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_boston
```

```
mat, label = load_boston(return_X_y = True)

# fitting input matrix and label on linear regression object
from frovedis.mllib.linear_model import LinearRegression
lr = LinearRegression(solver = 'sag').fit(mat,label)
```

When native python data is provided, it is converted to frovedis-like inputs and sent to frovedis server which consumes some data transfer time. Pre-constructed frovedlis-like inputs can be used to speed up the training time, specially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_boston
mat, label = load_boston(return_X_y = True)

# Since "mat" is numpy dense data, we have created FrovedisColmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisColmajorMatrix
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisColmajorMatrix(mat)
dlbl = FrovedisDvector(lbl)

# Linear Regression with pre-constructed frovedlis-like inputs
from frovedis.mllib.linear_model import LinearRegression
lr = LinearRegression(solver = 'sag').fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

9.3.1.3 predict(X)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data.

Purpose
It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server.

For example,

```
# predicting on linear regression model
lr.predict(mat[:10])
```

Output

```
[30.00384338 25.02556238 30.56759672 28.60703649 27.94352423 25.25628446
 23.00180827 19.53598843 11.52363685 18.92026211]
```

If the above pre-constructed training data (cmat) is to be used during prediction, the same can be used as follows:

```
# predicting on sag linear regression model using pre-constructed input
lr.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[30.00384338 25.02556238 30.56759672 28.60703649 27.94352423 25.25628446
 23.00180827 19.53598843 11.52363685 18.92026211]
```

Return Value
It returns a numpy array of float or double(float64) type and has shape (n_samples,) containing the predicted outputs.

9.3.1.4 score(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data.
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of (nsamples,). When it is None (not specified), an uniform weight vector is assigned on each input sample.

Purpose
Calculate the root mean square value on the given test data and labels i.e. R2(r-squared) of self.predict(X) wrt. y.

The coefficient 'R2' is defined as (1 - (u/v)),
where 'u' is the residual sum of squares ((y_true - y_pred) ** 2).sum() and,
'v' is the total sum of squares ((y_true - y_true.mean()) ** 2).sum().
The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R2 score of 0.0.

For example,

```
# calculate R2 score on given test data and labels
lr.score(mat[:10], label[:10])
```

Output

```
0.40
```

Return Value
It returns an R2 score of float type.

9.3.1.5 load(fname, dtype = None)

Parameters
fname: A string object containing the name of the file having model information to be loaded.
dtype: A data-type is inferred from the input data. Currently, expected input data-type is either float or double(float64). (Default: None)

Purpose
It loads the model from the specified file (having little-endian binary data).

For example,

```
lr.load("./out/LNRModel")
```

Return Value
It simply returns "self" instance.

9.3.1.6 save(fname)

Parameters
fname: A string object containing the name of the file on which the target model is to be saved.

Purpose
On success, it writes the model information (weight values etc.) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the linear regression model
lr.save("./out/LNRModel")
```

This will save the linear regression model on the path "/out/LNRModel".

Return Value
It returns nothing.

9.3.1.7 debug_print()

Purpose
It shows the target model information (weight values, intercept) on the server side user terminal. It is mainly used for debugging purpose.

For example,

```
lr.debug_print()
```

Output

```
-------- Weight Vector:: --------
-0.108011 0.0464205 0.0205586 2.68673 -17.7666 3.80987 0.000692225 -1.47557 0.306049
-0.0123346 -0.952747 0.00931168 -0.524758
Intercept:: 36.4595
```

It displays the weights and intercept values on the trained model which is currently present on the server.

Return Value
It returns nothing.

9.3.1.8 release()

Purpose
It can be used to release the in-memory model at frovedis server.

For example,

```
lr.release()
```

This will reset the after-fit populated attributes to None, along with releasing server side memory.

Return Value
It returns nothing.

9.3.1.9 is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not. In case, predict() is used before training the model, then it can prompt the user to train the linear regression model first.

Return Value
It returns 'True', if the model is already fitted otherwise, it returns 'False'.

9.4 SEE ALSO

lasso_regression, ridge_regression, dvector, crs_matrix

Chapter 10

Lasso Regression

10.1 NAME

Lasso Regression - A regression algorithm to predict the continuous output with L1 regularization.

10.2 SYNOPSIS

class frovedis.mllib.linear_model.Lasso (alpha=0.01, fit_intercept=True, normalize=False,
precompute=False, copy_X=True, max_iter=1000,
tol=1e-4, warm_start=False, positive=False,
random_state=None, selection='cyclic',
verbose=0, solver='sag')

10.2.1 Public Member Functions

fit(X, y, sample_weight=None)
predict(X)
save(filename)
load(filename)
debug_print()
release()

10.3 DESCRIPTION

Linear least squares is the most common formulation for regression problems. It is a linear method with the loss function given by the **squared loss**:

```
L(w;x,y) := 1/2(wTx-y)^2
```

Where the vectors x are the training data examples and y are their corresponding labels which we want to predict. w is the linear model (also known as weight) which uses a single weighted sum of features to make a prediction. The method is called linear since it can be expressed as a function of wTx and y. Lasso regression uses L1 regularization to address the overfit problem.

The gradient of the squared loss is: (wTx-y).x
The gradient of the regularizer is: sign(w)

Frovedis provides implementation of lasso regression with two different optimizers: (1) stochastic gradient descent with minibatch and (2) LBFGS optimizer.

The simplest method to solve optimization problems of the form **min f(w)** is gradient descent. Such first-order optimization methods well-suited for large-scale and distributed computation. Whereas, L-BFGS is an optimization algorithm in the family of quasi-Newton methods to solve the optimization problems of the similar form.

Like the original BFGS, L-BFGS (Limited Memory BFGS) uses an estimation to the inverse Hessian matrix to steer its search through feature space, but where BFGS stores a dense nxn approximation to the inverse Hessian (n being the number of features in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. L-BFGS often achieves rapider convergence compared with other first-order optimization.

This module provides a client-server implementation, where the client application is a normal python scikit-learn program. Scikit-learn has its own linear_model providing the Lasso Regression support. But that algorithm is non-distributed in nature. Hence it is slower when comparing with the equivalent Frovedis algorithm (see frovedis manual for ml/lasso_regression) with big dataset. Thus in this implementation, a scikit-learn client can interact with a frovedis server sending the required python data for training at frovedis side. Python data is converted into frovedis compatible data internally and the scikit-learn ML call is linked with the respective frovedis ML call to get the job done at frovedis server.

Scikit-learn side call for Lasso Regression quickly returns, right after submitting the training request to the frovedis server with a unique model ID for the submitted training request.

When operations like prediction will be required on the trained model, scikit-learn client sends the same request to frovedis server on the same model (containing the unique ID) and the request is served at frovedis server and output is sent back to the scikit-learn client.

10.3.1 Detailed Description

10.3.1.1 Lasso()

Parameters
alpha: A double parameter containing the learning rate. (Default: 0.01)
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)
normalize: A boolean parameter (unused)
precompute: A boolean parameter (unused)
copy_X: A boolean parameter (unsed)
max_iter: An integer parameter specifying maximum iteration count. (Default: 1000)
tol: A double parameter specifying the convergence tolerance value, (Default: 1e-4)
warm_start: A boolean parameter (unused)
positive: A boolean parameter (unused)
random_state: An integer, None or RandomState instance. (unused)
selection: A string object. (unused)
verbose: An integer object specifying the log level to use. (Default: 0)
solver: A string object specifying the solver to use. (Default: 'sag')

Purpose
It initialized a Lasso object with the given parameters.

The parameters: "normalize", "precompute", "copy_X", "warm_start", "positive", "random_state" and "selection" are not yet supported at frovedis side. Thus they don't have any significance in this call. They are simply provided for the compatibility with scikit-learn application.

"solver" can be either 'sag' for frovedis side stochastic gradient descent or 'lbfgs' for frovedis side LBFGS optimizer when optimizing the linear regression model.

"verbose" value is set at 0 by default. But it can be set to 1 (for DEBUG mode) or 2 (for TRACE mode) for getting training time logs from frovedis server.

Return Value
It simply returns "self" reference.

10.3.1.2 fit(X, y, sample_weight=None)

Parameters
X: A scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix.
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like optional parameter. (unused)

Purpose
It accepts the training feature matrix (X) and corresponding output labels (y) as inputs from the user and trains a linear regression model with L1 regularization with those data at frovedis server.

It doesn't support any initial weight to be passed as input at this moment. Thus the "sample_weight" parameter will simply be ignored. It starts with an initial guess of zeros for the model vector and keeps updating the model to minimize the cost function until convergence is achieved or maximum iteration count is reached.

For example,

```
# loading sample CRS data file
mat = FrovedisCRSMatrix().load("./sample")
lbl = FrovedisDvector([1.1,0.2,1.3,1.4,1.5,0.6,1.7,1.8])

# fitting input matrix and label on lasso object
lr = Lasso(solver='sgd', verbose=2).fit(mat,lbl)
```

Return Value
It simply returns "self" reference.
Note that the call will return quickly, right after submitting the fit request at frovedis server side with a unique model ID for the fit request. It may be possible that the training is not completed at the frovedis server side even though the client scikit-learn side fit() returns.

10.3.1.3 predict(X)

Parameters
X: A scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix.

Purpose
It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server.

Return Value
It returns a numpy array of double (float64) type containing the predicted outputs.

10.3.1.4 save(filename)

Parameters
filename: A string object containing the name of the file on which the target model is to be saved.

Purpose
On success, it writes the model information (weight values etc.) in the specified file as little-endian binary data. Otherwise, it throws an exception.

Return Value
It returns nothing.

10.3.1.5 load(filename)

Parameters
filename: A string object containing the name of the file having model information to be loaded.

Purpose
It loads the model from the specified file (having little-endian binary data).

Return Value
It simply returns "self" instance.

10.3.1.6 debug_print()

Purpose
It shows the target model information (weight values etc.) on the server side user terminal. It is mainly used for debugging purpose.

Return Value
It returns nothing.

10.3.1.7 release()

Purpose
It can be used to release the in-memory model at frovedis server.

Return Value
It returns nothing.

10.4 SEE ALSO

linear_regression, ridge_regression, dvector, crs_matrix

Chapter 11

Ridge Regression

11.1 NAME

Ridge Regression - A regression algorithm to predict the continuous output with L2 regularization.

11.2 SYNOPSIS

class frovedis.mllib.linear_model.Ridge(alpha=0.01, fit_intercept=True,
normalize=False, copy_X=True,
max_iter=None, tol=1e-3,
solver='auto', random_state=None,
lr_rate=1e-8, verbose=0,
warm_start = False)

11.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
score(X, y, sample_weight = None)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

11.3 DESCRIPTION

Linear least squares is the most common formulation for regression problems. It is a linear method with the loss function given by the **squared loss**:

```
L(w;x,y) := 1/2(wTx-y)^2
```

Where the vectors x are the training data examples and y are their corresponding labels which we want to predict. w is the linear model (also known as weight) which uses a single weighted sum of features to make a

prediction. The method is called linear since it can be expressed as a function of wTx and y. Ridge regression uses L2 regularization to address the overfit problem.

The gradient of the squared loss is: (wTx-y).x
The gradient of the regularizer is: w

Frovedis provides implementation of ridge regression with two different optimizers:
(1) stochastic gradient descent with minibatch
(2) LBFGS optimizer

The simplest method to solve optimization problems of the form **min f(w)** is gradient descent. Such first-order optimization methods well-suited for large-scale and distributed computation. Whereas, L-BFGS is an optimization algorithm in the family of quasi-Newton methods to solve the optimization problems of the similar form.

Like the original BFGS, L-BFGS (Limited Memory BFGS) uses an estimation to the inverse Hessian matrix to steer its search through feature space, but where BFGS stores a dense nxn approximation to the inverse Hessian (n being the number of features in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. L-BFGS often achieves rapider convergence compared with other first-order optimization.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis interface is almost same as Scikit-learn Ridge Regression interface, but it doesn't have any dependency with Scikit-learn. It can be used simply even if the system doesn't have Scikit-learn installed. Thus in this implementation, a python client can interact with a frovedis server sending the required python data for training at frovedis side. Python data is converted into frovedis compatible data internally and the python ML call is linked with the respective frovedis ML call to get the job done at frovedis server.

Python side calls for Ridge Regression on the frovedis server. Once the training is completed with the input data at the frovedis server, it returns an abstract model with a unique model ID to the client python program.

When prediction-like request would be made on the trained model, python program will send the same request to the frovedis server. After the request is served at the frovedis server, the output would be sent back to the python client.

11.3.1 Detailed Description

11.3.1.1 Ridge()

Parameters
alpha: A postive value of double(float64) type is called the regularization strength parameter. (Default: 0.01)
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)
normalize: A boolean parameter (unused)
copy_X: A boolean parameter (unsed)
max_iter: An integer parameter specifying maximum iteration count. (Default: None) When it is None(not specified explicitly), it will be set as 1000.
tol: Zero or a positive value of double(float64) type specifying the convergence tolerance value. (Default: 1e-3)
solver: A string object specifying the solver to use. It can be "sag" for frovedis side stochastic gradient descent or "lbfgs" for frovedis side LBFGS optimizer when optimizing the ridge regression model. Initilaly solver is "auto" by default. In such cases, it will select "sag" solver. Both "sag" and "lbfgs" handle L2 penalty.
random_state: An integer, None or RandomState instance. (unused)
lr_rate: Zero or a positive value of double(float64) type containing the learning rate. (Default: 1e-8)
verbose: An integer parameter specifying the log level to use. Its value is 0 by default(for INFO mode and not specified explicitly). But it can be set to 1 (for DEBUG mode) or 2 (for TRACE mode) for getting

training time logs from frovedis server.
warm_start: A boolean parameter which when set to True, reuses the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. (Default: False)

Attributes
coef_: It is a python ndarray(containing float or double(float64) typed values depending on data-type of input matrix (X)) of estimated coefficients for the ridge regression problem. It has shape (n_features,).
intercept(bias)_: It is a python ndarray(contaning float or double(float64) typed values depending on data-type of input matrix (X)). If fit_intercept is set to False, the intercept is set to zero. It has shape (1,).
n_iter_: A positive integer value used to get the actual iteration point at which the problem is converged.

Purpose
It initialized a Ridge object with the given parameters.

The parameters: "normalize", "copy_X" and "random_state" are simply kept in to to make the interface uniform to the Scikit-learn Ridge Regression module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

11.3.1.2 fit(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisColmajorMatrix for dense data.
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of (nsamples,). When it is None (not specified), an uniform weight vector is assigned on each input sample.

Purpose
It accepts the training feature matrix (X) and corresponding output labels (y) as inputs from the user and trains a ridge regression model with L2 regularization with those data at frovedis server.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_boston
mat, label = load_boston(return_X_y = True)

# fitting input matrix and label on ridge regression object
from frovedis.mllib.linear_model import Ridge
rr = Ridge(solver = 'lbfgs').fit(mat, label)
```

When native python data is provided, it is converted to frovedis-like inputs and sent to frovedis server which consumes some data transfer time. Pre-constructed frovedlis-like inputs can be used to speed up the training time, specially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_boston
mat, label = load_boston(return_X_y = True)

# Since "mat" is numpy dense data, we have created FrovedisColmajorMatrix.
```

```
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisColmajorMatrix
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisColmajorMatrix(mat)
dlbl = FrovedisDvector(lbl)

# Ridge Regression with pre-constructed frovedlis-like inputs
from frovedis.mllib.linear_model import Ridge
rr = Ridge(solver = 'lbfgs').fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

11.3.1.3 predict(X)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix as for dense data.

Purpose
It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server.

For example,

```
# predicting on ridge model
rr.predict(mat)
```

Output

```
[28.71021961 23.76178249 30.4133597  28.97666397 28.67315792 24.66094155
 20.57933251 17.64381951  9.47525481 16.75502233 17.6363261  19.15371596
 18.82699337 20.73240606 20.44989254 20.31470466 21.85948797 18.28896663
 ...
 16.35934999 20.59831667 21.23383603 17.60978119 14.01230367 19.28714569
 21.74889541 18.19015854 20.88375846 26.19279806 24.06841151 30.327149
 28.62134714 23.7732722 ]
```

If the above pre-constructed training data (cmat) is to be used during prediction, the same can be used as follows:

```
# predicting on ridge regression model using pre-constructed input
rr.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[28.71021961 23.76178249 30.4133597  28.97666397 28.67315792 24.66094155
 20.57933251 17.64381951  9.47525481 16.75502233 17.6363261  19.15371596
 18.82699337 20.73240606 20.44989254 20.31470466 21.85948797 18.28896663
 ...
 16.35934999 20.59831667 21.23383603 17.60978119 14.01230367 19.28714569
 21.74889541 18.19015854 20.88375846 26.19279806 24.06841151 30.327149
 28.62134714 23.7732722 ]
```

Return Value
It returns a numpy array of double(float64) type type and has shape (n_samples,) containing the predicted outputs.

11.3.1.4 score(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data.
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of (nsamples,). When it is None (not specified), an uniform weight vector is assigned on each input sample.

Purpose
Calculate the root mean square value on the given test data and labels i.e. R2(r-squared) of self.predict(X) wrt. y.

The coefficient 'R2' is defined as (1 - (u/v)),
where 'u' is the residual sum of squares ((y_true - y_pred) ** 2).sum() and
'v' is the total sum of squares ((y_true - y_true.mean()) ** 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R2 score of 0.0.

For example,

```
# calculate R2 score on given test data and labels
rr.score(mat, label)
```

Output

```
0.70
```

Return Value
It returns an R2 score of float type.

11.3.1.5 load(fname, dtype = None)

Parameters
fname: A string object containing the name of the file having model information to be loaded.
dtype: A data-type is inferred from the input data. Currently, expected input data-type is either float or double(float64). (Default: None)

Purpose
It loads the model from the specified file (having little-endian binary data).

For example,

```
rr.load("./out/RidgeModel")
```

Return Value
It simply returns "self" instance.

11.3.1.6 save(fname)

Parameters
fname: A string object containing the name of the file on which the target model is to be saved.

Purpose
On success, it writes the model information (weight values etc.) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the ridge regression model
rr.save("./out/RidgeModel")
```

This will save the ridge regression model on the path "/out/RidgeModel".

Return Value
It returns nothing.

11.3.1.7 debug_print()

Purpose
It shows the target model information (weight values, intercept) on the server side user terminal. It is mainly used for debugging purpose.

For example,

```
rr.debug_print()
```

Output

```
-------- Weight Vector:: --------
-0.092909 0.0669578 -0.013893 -0.0253964 -0.314699 5.54217 -0.0117369 -1.29332
 0.215405 -0.0133328 -0.253942 0.0153361 -0.444395
Intercept:: 1.06476
```

It displays the weights and intercept values on the trained model which is currently present on the server.

Return Value
It returns nothing.

11.3.1.8 release()

Purpose
It can be used to release the in-memory model at frovedis server.

For example,

```
rr.release()
```

This will reset the after-fit populated attributes to None, along with releasing server side memory.

Return Value
It returns nothing.

11.3.1.9 is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not. In case, predict() is used before training the model, then it can prompt the user to train the ridge regression model first.

Return Value
It returns 'True', if the model is already fitted otherwise, it returns 'False'.

11.4 SEE ALSO

linear_regression, lasso_regression, dvector, crs_matrix

Chapter 12

Logistic Regression

12.1 NAME

Logistic Regression - A classification algorithm to predict the binary and multi-class output with logistic loss.

12.2 SYNOPSIS

class frovedis.mllib.linear_model.LogisticRegression(penalty='l2', dual=False, tol=1e-4,
C=100.0, fit_intercept=True,
intercept_scaling=1, class_weight=None,
random_state=None, solver='lbfgs',
max_iter=1000, multi_class='auto',
verbose=0, warm_start=False,
n_jobs=1, l1_ratio=None,
lr_rate=0.01, use_shrink=False)

12.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
predict_proba(X)
score(X, y, sample_weight = None)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

12.3 DESCRIPTION

Classification aims to divide the items into categories. The most common classification type is binary classification, where there are two categories, usually named positive and negative. The other is multinomial classification, where there are more than two categories. Frovedis supports both binary and multinomial logistic regression algorithms. For multinomial classification, it uses softmax probability.

Logistic regression is widely used to predict a binary response. It is a linear method with the loss function given by the **logistic loss**:

```
L(w;x,y) := log(1 + exp(-ywTx))
```

Where the vectors x are the training data examples and y are their corresponding labels (Frovedis supports any values as for labels, but internally it encodes the input binary labels to -1 and 1, and input multinomial labels to 0, 1, 2, . . . , N-1 (where N is the no. of classes) before training at Frovedis server) which we want to predict. w is the linear model (also called as weight) which uses a single weighted sum of features to make a prediction. Frovedis Logistic Regression supports ZERO, L1 and L2 regularization to address the overfit problem. However, LBFGS solver supports only L2 regularization.
The gradient of the logistic loss is: -y(1 - 1 / (1 + exp(-ywTx))).x
The gradient of the L1 regularizer is: sign(w)
And, the gradient of the L2 regularizer is: w
For binary classification problems, the algorithm outputs a binary logistic regression model. Given a new data point, denoted by x, the model makes predictions by applying the logistic function:

```
f(z) := 1 / 1 + exp(-z)
```

Where z = wTx. By default (threshold=0.5), if f(wTx) $>$ 0.5, the response is positive (1), else the response is negative (0).

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme. Currently the "multinomial" option is supported only by the "sag" solvers.
Frovedis provides implementation of logistic regression with two different optimizers:
(1) stochastic gradient descent with minibatch
(2) LBFGS optimizer
They can handle both dense and sparse input.

The simplest method to solve optimization problems of the form **min f(w)** is gradient descent. Such first-order optimization methods well-suited for large-scale and distributed computation. Whereas, L-BFGS is an optimization algorithm in the family of quasi-Newton methods to solve the optimization problems of the similar form.

Like the original BFGS, L-BFGS (Limited Memory BFGS) uses an estimation to the inverse Hessian matrix to steer its search through feature space, but where BFGS stores a dense nxn approximation to the inverse Hessian (n being the number of features in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. L-BFGS often achieves rapider convergence compared with other first-order optimization.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis interface is almost same as Scikit-learn Logistic Regression interface, but it doesn't have any dependency with Scikit-learn. It can be used simply even if the system doesn't have Scikit-learn installed. Thus in this implementation, a python client can interact with a frovedis server sending the required python data for training at frovedis side. Python data is converted into frovedis compatible data internally and the python ML call is linked with the respective frovedis ML call to get the job done at frovedis server.

Python side calls for Logistic Regression on the frovedis server. Once the training is completed with the input data at the frovedis server, it returns an abstract model with a unique model ID to the client python program.

When prediction-like request would be made on the trained model, python program will send the same request to the frovedis server. After the request is served at the frovedis server, the output would be sent back to the python client.

12.3.1 Detailed Description

12.3.1.1 LogisticRegression()

Parameters
penalty: A string object containing the regularizer type to use. Currenlty none, l1 and l2 are supported by Frovedis. (Default: 'l2')
dual: A boolean parameter. (unused)
tol: A double(float64) type value specifying the convergence tolerance value. It must be zero or a postive value. (Default: 1e-4)
C: A float parameter, it is the inverse of regularization strength. It must be a positive value of float type. Like in support vector machines, smaller values specify stronger regularization. (Default: 100.0)
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)
intercept_scaling: An integer parameter. (unused)
class_weight: A python dictionary or a string object. (unused)
random_state: An integer, None or RandomState instance. (unused)
solver: A string object specifying the solver to use. (Default: 'lbfgs')
It can be "sag" for frovedis side stochastic gradient descent or "lbfgs" for frovedis side LBFGS optimizer when optimizing the logistic regression model.
"sag" handle L1, L2 or no penalty.
"lbfgs" handle only L2 penalty.
max_iter: A positive integer value specifying maximum iteration count. (Default: 1000)
multi_class: A string object specifying type of classification. If it is "auto" or "ovr", then a binary classification is selected when N = 2, otherwise multinomial classification is selected (where N is the no. of classes in training labels). If it is "multinomial", then it always selects a multinomial problem (even when N = 2). Only "sag" solvers support multinomial classification currently. (Default: 'auto')
verbose: An integer parameter specifying the log level to use. Its value is 0 by default(for INFO mode and not explicitly specified). But it can be set to 1 (for DEBUG mode) or 2 (for TRACE mode) for getting training time logs from frovedis server.
warm_start: A boolean parameter which when set to True, reuses the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. (Default: False)
n_jobs: An integer parameter. (unused)
l1_ratio: A float parameter, also called the Elastic-Net mixing parameter. (unused)
lr_rate(alpha): A double(float64) parameter containing the learning rate. (Default: 0.01)
use_shrink: A boolean parameter applicable only for "sag" solver with "sparse" input (X). When set to True for sparse input, it can improve training performance by reducing communication overhead across participating processes. (Default: False)

Attributes
coef_: It is a python ndarray(float or double(float64) values depending on input matrix data type) of coefficient of the features in the decision function. It has shape (1, n_features) when the given problem is "binary" and (n_classes, n_features) when it is a "multinomial" problem.
intercept(bias)_: It is a python ndarray(float or double(float64) values depending on input matrix data type) If fit_intercept is set to False, the intercept is set to zero. It has shape (1,) when the given problem is "binary" and (n_classes) when its "multinomial" problem.
classes_: It is a python ndarray(any type) of unique labels given to the classifier during training. It has shape (n_classes,).
n_iter_: It is a python ndarray of shape(1,) and has integer data. It is used to get the actual iteration point at which the problem is converged.

Purpose
It initializes a Logistic Regression object with the given parameters.

The parameters: "dual", "intercept_scaling", "class_weight", "random_state", "n_jobs" and "l1_ratio" are simply kept to make the interface uniform to the Scikit-learn Logistic Regression module. They are not used

anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

12.3.1.2 fit(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisColmajorMatrix for dense data.
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of (nsamples,). When it is None (not specified), an uniform weight vector is assigned on each input sample.

Purpose
It accepts the training feature matrix (X) and corresponding output labels (y) as inputs from the user and trains a logistic regression model with specifed regularization with those data at frovedis server.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_breast_cancer
mat, lbl = load_breast_cancer(return_X_y = True)

# fitting input matrix and label on logistic regression object
from frovedis.mllib.linear_model import LogisticRegression
lr = LogisticRegression(solver = 'lbfgs').fit(mat,lbl)
```

When native python data is provided, it is converted to frovedis-like inputs and sent to frovedis server which consumes some data transfer time. Pre-constructed frovedlis-like inputs can be used to speed up the training time, specially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_breast_cancer
mat, lbl = load_breast_cancer(return_X_y = True)

# Since "mat" is numpy dense data, we have created FrovedisColmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisColmajorMatrix
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisColmajorMatrix(mat)
dlbl = FrovedisDvector(lbl)

# Logistic Regression with pre-constructed frovedlis-like inputs
from frovedis.mllib.linear_model import LogisticRegression
lr = LogisticRegression(solver = 'lbfgs').fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

12.3.1.3 predict(X)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data.

Purpose
It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server.

For example,

```
# predicting on lbfgs logistic regression model
lr.predict(mat)
```

Output

```
[0 0 0 ... 0 0 1]
```

If the above pre-constructed training data (cmat) is to be used during prediction, the same can be used as follows:

```
# predicting on lbfgs logistic regression model using pre-constructed input
lr.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[0 0 0 ... 0 0 1]
```

Return Value
It returns a numpy array of float or double(float64) type and of shape (n_samples,) containing the predicted outputs.

12.3.1.4 predict_proba(X)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data.

Purpose
It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server. But unlike predict(), it returns the softmax probability matrix of shape (n_samples, n_classes) containing the probability of each class in each sample.

For example,

```
# finds the probablity sample for each class in the model
lr.predict_proba(mat)
```

Output

```
[[1.46990588e-19 1.00000000e+00]
 [7.23344224e-10 9.99999999e-01]
 [8.43160984e-10 9.99999999e-01]
 ...
 [4.03499383e-04 9.99596501e-01]
 [3.03132738e-13 1.00000000e+00]
 [6.14030540e-03 9.93859695e-01]]
```

Return Value
It returns a numpy array of float or double(float64) type and of shape (n_samples, n_classes) containing the prediction probability values.

12.3.1.5 score(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data.
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of (nsamples,). When it is None (not specified), an uniform weight vector is assigned on each input sample.

Purpose
Calculate mean accuracy on the given test data and labels i.e. mean accuracy of self.predict(X) wrt. y.

For example,

```
# calculate mean accuracy score on given test data and labels
lr.score(mat,lbl)
```

Output

```
0.96
```

Return Value
It returns an accuracy score of float type.

12.3.1.6 load(fname, dtype = None)

Parameters
fname: A string object containing the name of the file having model information to be loaded.
dtype: A data-type is inferred from the input data. Currently, expected input data-type is either float or double(float64). (Default: None)

Purpose
It loads the model from the specified file (having little-endian binary data).

For example,

```
lr.load("./out/LRModel")
```

Return Value
It simply returns “self” instance.

12.3.1.7 save(fname)

Parameters
fname: A string object containing the name of the file on which the target model is to be saved.

Purpose
On success, it writes the model information (weight values etc.) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the logistic regression model
lr.save("./out/LRModel")
```

This will save the logistic regression model on the path '/out/LRModel'.

Return Value
It returns nothing.

12.3.1.8 debug_print()

Purpose
It shows the target model information (weight values, intercept, etc.) on the server side user terminal. It is mainly used for debugging purpose.

For example,

```
lr.debug_print()
```

Output

```
-------- Weight Vector:: --------
25.4745 47.8416 155.732 190.863 0.271114 0.0911008 -0.151433 -0.0785512 0.511576
0.203452 0.199293 3.8659 1.22203 -42.3556 0.0239707 0.0395711 0.0389786 0.017432
0.0647208 0.0105295 24.7162 60.7113 150.789 -148.921 0.354222 0.104251 -0.202345
-0.0363726 0.734499 0.22635
Intercept:: 60.7742
Threshold:: 0.5
```

It displays the weights, intercept, etc. values on the trained model which is currently present on the server.

Return Value
It returns nothing.

12.3.1.9 release()

Purpose
It can be used to release the in-memory model at frovedis server.

For example,

```
lr.release()
```

This will reset the after-fit populated attributes to None, along with releasing server side memory.

Return Value
It returns nothing.

12.3.1.10 is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not. In case, predict() is used before training the model, then it can prompt the user to train the logistic regression model first.

Return Value
It returns 'True', if the model is already fitted otherwise, it returns 'False'.

12.4 SEE ALSO

linear_svm, dvector, crs_matrix

Chapter 13

Linear SVM

13.1 NAME

Linear SVM (Support Vector Machines) - A classification algorithm to predict the binary output with hinge loss.

13.2 SYNOPSIS

class frovedis.mllib.svm.LinearSVC(penalty='l2', loss='hinge', dual=True,
tol=1e-4, C=1.0, multi_class='ovr',
fit_intercept=True, intercept_scaling=1,
class_weight=None, verbose=0,
random_state=None, max_iter=1000,
lr_rate=0.01, solver='sag',
warm_start=False)

13.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
load(fname, dtype = None)
save(fname)
score(X, y, sample_weight = None)
debug_print()
release()
is_fitted()

13.3 DESCRIPTION

Classification aims to divide items into categories. The most common classification type is binary classification, where there are two categories, usually named positive and negative. **Frovedis supports only binary Linear SVM classification algorithm**.

The Linear SVM is a standard method for large-scale classification tasks. It is a linear method with the loss function given by the **hinge loss**:

```
L(w;x,y) := max{0, 1-ywTx}
```

Where the vectors x are the training data examples and y are their corresponding labels (Frovedis supports any values as for labels, but internally it encodes the input binary labels to -1 and 1, before training at Frovedis server) which we want to predict. w is the linear model (also known as weight) which uses a single weighted sum of features to make a prediction. Linear SVM supports ZERO, L1 and L2 regularization to address the overfit problem.

The gradient of the hinge loss is: -y.x, if ywTx < 1, 0 otherwise.
The gradient of the L1 regularizer is: sign(w)
And The gradient of the L2 regularizer is: w

For binary classification problems, the algorithm outputs a binary svm model. Given a new data point, denoted by x, the model makes predictions based on the value of wTx.

By default (threshold=0), if wTx >= 0, then the response is positive (1), else the response is negative (0).

Frovedis provides implementation of linear SVM with **stochastic gradient descent with minibatch**.

The simplest method to solve optimization problems of the form **min f(w)** is gradient descent. Such first-order optimization methods well-suited for large-scale and distributed computation.

This module provides a client-server implementation, where the client application is a normal python program. Frovedis is almost same as Scikit-learn svm module providing the LinearSVC (Support Vector Classification) support, but it doesn't have any dependency with Scikit-learn. It can be used simply even if the system doesn't have Scikit-learn installed. Thus in this implementation, a python client can interact with a frovedis server sending the required python data for training at frovedis side. Python data is converted into frovedis compatible data internally and the python ML call is linked with the respective frovedis ML call to get the job done at frovedis server.

Python side calls for LinearSVC on the frovedis server. Once the training is completed with the input data at the frovedis server, it returns an abstract model with a unique model ID to the client python program.

When prediction-like request would be made on the trained model, python program will send the same request to the frovedis server. After the request is served at the frovedis server, the output would be sent back to the python client.

13.3.1 Detailed Description

13.3.1.1 LinearSVC()

Parameters
penalty: A string object containing the regularizer type to use. Currently none, l1 and l2 are supported by Frovedis. (Default: 'l2')
loss: A string object containing the loss function type to use. Currently svm supports only hinge loss. (Default: 'hinge')
dual: A boolean parameter (unused)
tol: A double(float64) parameter specifying the convergence tolerance value. It must be zero or a positive value. (Default: 1e-4)
C: A float parameter, also called as inverse of regularization strength. It must be positive. (Default: 1.0)
multi_class: A string object specifying type of classification. (unused)
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)
intercept_scaling: An integer parameter. (unused)
class_weight: A python dictionary or a string object. (unused)
verbose: An integer parameter specifying the log level to use. Its value is set as 0 by default(for INFO mode). But it can be set to 1(for DEBUG mode) or 2(for TRACE mode) for getting training time logs from frovedis

server.
random_state: An integer, None or RandomState instance. (unused)
max_iter: An integer parameter specifying maximum iteration count. It is positive interger. (Default: 1000)
lr_rate: A double(float64) parameter containing the learning rate. (Default: 0.01)
solver: A string object specifying the solver to use. (Default: 'sag')
"sag" handle L1, L2 or no penalty.
warm_start: A boolean parameter which when set to True, reuses the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. (Default: False)

Attributes
coef_: It is a python ndarray(containing float or double(float64) typed values depending on data-type of input matrix (X)). It is the weights assigned to the features. It has shape (1, n_features).
classes_: It is a python ndarray(any type) of unique labels given to the classifier during training. It has shape (n_classes,).
intercept_: It is a python ndarray(float or double(float64) values depending on input matrix data type) and has shape(1,).
n_iter: It is a python ndarray of shape(1,) and has integer data. It is used to get the actual iteration point at which the problem is converged.

Purpose
It initializes a LinearSVC object with the given parameters.

The parameters: "dual", "intercept_scaling", "class_weight", "multi_class"and "random_state" are simply kept to make the interface uniform to Scikit-learn LinearSVC module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

13.3.1.2 fit(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisColmajorMatrix for dense data. It has shape(n_samples, n_features).
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of (nsamples,). When it is None (not specified), an uniform weight vector is assigned on each input sample.

Purpose
It accepts the training feature matrix (X) and corresponding output labels (y) as inputs from the user and trains a linear svm model with specifed regularization with those data at frovedis server.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_breast_cancer
mat, lbl = load_breast_cancer(return_X_y = True)

# fitting input matrix and label on linear SVC object
from frovedis.mllib.svm import LinearSVC
svm = LinearSVC().fit(mat, lbl)
```

When native python data is provided, it is converted to frovedis-like inputs and sent to frovedis server which consumes some data transfer time. Pre-constructed frovedis-like inputs can be used to speed up the training time, specially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_breast_cancer
mat, lbl = load_breast_cancer(return_X_y = True)

# Since "mat" is numpy dense data, we have created FrovedisColmajorMatrix.
and for scipy sparse data, FrovedisCRSMatrix should be used.
from frovedis.matrix.dense import FrovedisColmajorMatrix
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisColmajorMatrix(mat)

# Linear SVC with pre-constructed frovedis-like inputs
from frovedis.mllib.svm import LinearSVC
svm = LinearSVC().fit(cmat,dlbl)
```

Return Value
It simply returns "self" reference.

13.3.1.3 predict(X)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data. It has shape(n_samples, n_features).

Purpose
It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server.

For example,

```
svm.predict(mat)
```

Output:

```
[0 0 0 ... 0 0 1]
```

If the above pre-constructed training data (cmat) is to be used during prediction, the same can be used as follows:

```
# predicting on LinearSVC using pre-constructed input
svm.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[0 0 0 ... 0 0 1]
```

Return Value
It returns a numpy array of double(float64) type containing the predicted outputs. It has shape(n_samples,).

13.3.1.4 load(fname, dtype = None)

Parameters
fname: A string object containing the name of the file having model information to be loaded.
dtype: A data-type is inferred from the input data. Currently, expected input data-type is either float or double(float64). (Default: None)

Purpose
It loads the model from the specified file(having little-endian binary data).

For example,

```
# loading the svc model
svm.load("./out/SVMModel")
```

Return Value
It simply returns "self" instance.

13.3.1.5 save(fname)

Parameters
fname: A string object containing the name of the file on which the target model is to be saved.

Purpose
On success, it writes the model information (weight values etc.) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
svm.save("./out/SVMModel")
```

Return Value
It returns nothing.

13.3.1.6 score(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data. It has shape(n_samples, n_features).
y: Any python array-like object or an instance of FrovedisDvector.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of (nsamples,). When it is None (not specified), an uniform weight vector is assigned on each input sample.

Purpose
Calculate mean accuracy on the given test data and labels i.e. mean accuracy of self.predict(X) wrt. y.

For example,

```
# calculate mean accuracy score on given test data and labels
svm.score(mat, lbl)
```

Output

```
0.63
```

Return Value
It returns an accuracy score of float type.

13.3.1.7 debug_print()

Purpose
It shows the target model information(weight values etc.) on the server side user terminal. It is mainly used for debugging purpose.

For example,

```
svm.debug_print()
```

Output:

```
-------- Weight Vector:: --------
 83.7418 122.163 486.84 211.922 1.32991 0.287324 -0.867741 -0.0505454
 2.04889 1.16388 0.750738 8.61861 -2.13628 -234.118 0.582984 0.445561
 0.353854 0.519177 0.667717 0.547778 89.3196 157.824 499.367
 -293.736 1.56023 -0.636429 -2.30027 -0.061839 2.66517 1.15244
 Intercept:: 19.3242
 Threshold:: 0
```

Return Value
It returns nothing.

13.3.1.8 release()

Purpose
It can be used to release the in-memory model at frovedis server.

For example,

```
svm.release()
```

This will reset the after-fit populated attributes to None, along with releasing server side memory.

Return Value
It returns nothing.

13.3.1.9 is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not. In case, predict() is used before training the model, then it can prompt the user to train the model first.

Return Value
It returns 'True', if the model is already fitted otherwise, it returns 'False'.

13.4 SEE ALSO

logistic_regression, dvector, crs_matrix

Chapter 14

Matrix Factorization using ALS

14.1 NAME

Matrix Factorization using ALS - A matrix factorization algorithm commonly used for recommender systems.

14.2 SYNOPSIS

class frovedis.mllib.recommendation.ALS(rank=None, max_iter=100, alpha=0.01,
reg_param=0.01, similarity_factor=0.1,
seed=0, verbose=0)

14.2.1 Public Member Functions

fit(X)
predict(ids)
recommend_users(pid, k)
recommend_products(uid, k)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

14.3 DESCRIPTION

Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user-item association matrix. Frovedis currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries.

Frovedis uses the alternating least squares(ALS) algorithm to learn these latent factors. The algorithm is based on a paper "Collaborative Filtering for Implicit Feedback Datasets" by Hu, et al.

This module provides a client-server implementation, where the client application is a normal python program. Scikit-learn does not have any collaborative filtering recommender algorithms like ALS. In this implementation,

python side recommender interfaces are provided, where a python client can interact with a frovedis server sending the required python data for training at frovedis side. Python data is converted into frovedis compatible data internally and the python ALS call is linked with the frovedis ALS call to get the job done at frovedis server.

Python side calls for ALS on the frovedis server. Once the training is completed with the input data at the frovedis server, it returns an abstract model with a unique model ID to the client python program.

When recommendation-like request would be made on the trained model, python program will send the same request to the frovedis server. After the request is served at the frovedis server, the output would be sent back to the python client.

14.3.1 Detailed Description

14.3.1.1 ALS()

rank: An integer parameter containing the user given rank for the input matrix. (Default: None)
When rank is None(not specified explicitly), it will be the minimum(256, min(M,N)), where M is number of users and N is number of items in input data. Rank must be a positive integer and in a range of > 0 to <= max(M,N).
max_iter: A positive integer specifying maximum iteration count. (Default: 100)
alpha: A double(float64) parameter containing the learning rate. It must be a positive double(float64). (Default: 0.01)
reg_param: A double(float64) parameter containing the regularization parameter. It must be a positive double(float64). (Default: 0.01)
similarity_factor: A double(float64) parameter, which helps to identify whether the algorithm will optimize the computation for similar user/item or not. If similarity percentage of user or item features is more than or equal to(>=) the given similarity_factor, the algorithm will optimize the computation for similar user/item. Otherwise, each user and item feature will be treated uniquely. Similarity factor must be in range of >= 0.0 to <= 1.0. (Default: 0.1)
seed: A long parameter containing the seed value to initialize the model structures with random values. (Default: 0)
verbose: An integer parameter specifying the log level to use. Its value is 0 by default(for INFO mode and not speicifed explicitly). But it can be set to 1(for DEBUG mode) or 2(for TRACE mode) for getting training time logs from frovedis server.

Purpose

It initializes an ALS object with the given parameters.

Return Value
It simply returns "self" reference.

14.3.2 fit(X)

Parameters
X: A scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix. It has shape(n_samples, n_features).

Purpose
It accepts the training sparse matrix (X) and trains a matrix factorization model on that at frovedis server.

It starts with initializing the model structures of the size MxF and NxF(where M is the number of users and N is the products in the given rating matrix and F is the given rank) with random values and keeps updating them until maximum iteration count is reached.

For example,

```
# creating csr matrix
import numpy as np
from scipy.sparse import csr_matrix
row = np.array([0, 0, 1, 2, 2, 3, 3, 3, 4, 4, 5, 6, 6, 7, 7, 7])
col = np.array([0, 4, 0, 2, 3, 0, 1, 6, 0, 4, 0, 2, 3, 0, 1, 6])
data = np.array([2.0, 9.0, 1.0, 4.0, 8.0, 2.0, 3.0, 8.9, 2.0, 9.0, 1.0, 4.0, 8.0, 2.0,3.0, 8.9])
csr_matrix = csr_matrix((data, (row, col)), shape = (8, 7))

# fitting input matrix on ALS object
from frovedis.mllib.recommendation import ALS
als = ALS(rank = 4).fit(csr_matrix)
```

When native python data is provided, it is converted to frovedis-like inputs and sent to frovedis server which consumes some data transfer time. Pre-constructed frovedis-like inputs can be used to speed up the training time, specially when same data would be used for multiple executions.

For example,

```
# Since "mat" is scipy csr sparse matrix, we have created FrovedisCRSMatrix.
from frovedis.matrix.crs import FrovedisCRSMatrix
crs_mat = FrovedisCRSMatrix(mat)

# ALS with pre-constructed frovedis-like inputs
from frovedis.mllib.recommendation import ALS
als = ALS(rank = 4).fit(crs_mat)
```

Return Value
It simply returns "self" reference.

14.3.2.1 predict(ids)

Parameters
ids: A python tuple or list object containing the pairs of user id and product id to predict.

Purpose
It accepts a list of pair of user ids and product ids(0-based ID) in order to make prediction for their ratings from the trained model at frovedis server.

For example,

```
# this will print the predicted ratings for the given list of id pairs
als.predict([(1,1),(0,1),(2,3),(3,1)])
```

Output:

```
[ 0.00224735  0.00152505  0.99515575  0.99588757]
```

Return Value
It returns a numpy array containing the predicted ratings, of float or double(float64) type depending upon the input type.

14.3.2.2 recommend_users(pid, k)

Parameters
pid: An integer parameter specifying the product ID(0-based) for which to recommend users.
k: An integer parameter specifying the number of users to be recommended.

Purpose
It recommends the best "k" users with highest rating confidence in sorted order for the given product.

If k > number of rows(number of users in the given matrix when training the model), then it resets the k as "number of rows in the given matrix". This is done in order to recommend all the users with rating confidence values in descending order.

For example,

```
# recommend 2 users for second product
als.recommend_users(1,2)
```

Output:

```
('uids:', array([7, 3], dtype=int32))
('scores:', array([ 0.99588757,  0.99588757]))
```

Return Value
It returns a python list containing the pairs of recommended users and their corresponding rating confidence values(double(float64)) in descending order.

14.3.2.3 recommend_products(uid, k)

Parameters
uid: An integer parameter specifying the user ID(0-based) for which to recommend products.
k: An integer parameter specifying the number of products to be recommended.

Purpose
It recommends the best "k" products with highest rating confidence in sorted order for the given user.

If k > number of columns(number of products in the given matrix when training the model), then it resets the k as "number of columns in the given matrix". This is done in order to recommend all the products with rating confidence values in descending order.

For example,

```
# recommend 2 products for second user
print als.recommend_products(1,2)
```

Output:

```
(' recommend_product  pids:', array([0, 4], dtype=int32))
('scores:', array([ 0.99494576,  0.0030741 ]))[(0, 0.9949457612078868), (4, 0.0030740973144160397)]
```

Return Value
It returns a python list containing the pairs of recommended products and their corresponding rating confidence values(double(float64)) in descending order.

14.3.2.4 save(fname)

Parameters
fname: A string object containing the name of the file on which the target model is to be saved.

Purpose
On success, it writes the model information(user-product features etc.) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
als.save("./out/MyMFModel")
```

Return Value
It returns nothing.

14.3.2.5 load(fname, dtype = None)

Parameters
fname: A string object containing the name of the file having model information to be loaded.
dtype: A data-type is inferred from the input data. Currently, expected input data-type is either float or double(float64). (Default: None)

Purpose
It loads the model from the specified file(having little-endian binary data).

For example,

```
# loading the same model
als.load("./out/MyMFModel")
```

Return Value
It simply returns "self" instance.

14.3.2.6 debug_print()

Purpose
It shows the target model information on the server side user terminal. It is mainly used for debugging purpose.

For example,

```
 als.debug_print()
```

Output:

```
 ========== Matrix::X ==========
 0.829524 -0.84477 -0.152624 0.569863
 0.829528 -0.844775 -0.152625 0.569867
 0.829921 -0.845174 -0.152697 0.570136
========== Matrix::Y ==========
 0.473117 -0.481813 -0.087049 0.32502
 0.473117 -0.481813 -0.087049 0.32502
 0.473117 -0.481813 -0.087049 0.32502
```

It will print the matrix and labels of training data.

Return Value
It returns nothing.

14.3.2.7 release()

Purpose
It can be used to release the in-memory model at frovedis server.

For example,

```
als.release()
```

This will reset the after-fit populated attributes to None, along with releasing server side memory.

Return Value
It returns nothing.

14.3.2.8 is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not. In case, predict() is used before training the model, then it can prompt the user to train the model first.

Return Value
It returns 'True', if the model is already fitted otherwise, it returns 'False'.

14.4 SEE ALSO

crs_matrix

Chapter 15

kmeans

15.1 NAME

kmeans - A clustering algorithm commonly used in EDA (exploratory data analysis).

15.2 SYNOPSIS

class frovedis.mllib.cluster.KMeans (n_clusters=8, init='k-means++',
n_init=10, max_iter=300, tol=1e-4, precompute_distances='auto',
verbose=0, random_state=None, copy_x=True,
n_jobs=1, algorithm='auto')

15.2.1 Public Member Functions

fit(X, y=None)
predict(X)
save(filename)
load(filename)
debug_print()
release()

15.3 DESCRIPTION

Clustering is an unsupervised learning problem whereby we aim to group subsets of entities with one another based on some notion of similarity. K-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters (K).

This module provides a client-server implementation, where the client application is a normal python scikit-learn program. Scikit-learn has its own cluster module providing kmeans support. But that algorithm is non-distributed in nature. Hence it is slower when comparing with the equivalent Frovedis algorithm (see frovedis manual for ml/kmeans) with big dataset. Thus in this implementation, a scikit-learn client can interact with a frovedis server sending the required python data for training at frovedis side. Python data is converted into frovedis compatible data internally and the scikit-learn ML call is linked with the respective frovedis ML call to get the job done at frovedis server.

Scikit-learn side call for kmeans quickly returns, right after submitting the training request to the frovedis server with a unique model ID for the submitted training request.

When operations like prediction will be required on the trained model, scikit-learn client sends the same request to frovedis server on the same model (containing the unique ID) and the request is served at frovedis server and output is sent back to the scikit-learn client.

15.3.1 Detailed Description

15.3.1.1 KMeans()

Parameters *n_clusters*: An integer parameter specifying the number of clusters. (Default: 8)
init: A string object. (unused)
n_init: An integer parameter. (unused)
max_iter: An integer parameter specifying the maximum iteration count. (Default: 300)
tol: A double parameter specifying the convergence tolerance. (Default: 1e-4)
precompute_distances: A string object. (unused)
verbose: An integer object specifying the log level to use. (Default: 0)
random_state: An integer, None or RandomState instance. (unused)
copy_X: A boolean parameter. (unused)
n_jobs: An integer parameter. (unused)
algorithm: A string object. (unused)

Purpose
It initialized a KMeans object with the given parameters.

The parameters: "init", "n_init", "precompute_distances", "random_state", "copy_X", "n_jobs" and "algorithms" are not yet supported at frovedis side. Thus they don't have any significance in this call. They are simply provided for the compatibility with scikit-learn application.

"verbose" value is set at 0 by default. But it can be set to 1 (for DEBUG mode) or 2 (for TRACE mode) for getting training time logs from frovedis server.

Return Value
It simply returns "self" reference.

15.3.1.2 fit(X, y=None)

Parameters
X: A scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix.
y: None (simply ignored in scikit-learn as well).

Purpose

It clusters the given data points (X) into a predefined number (k) of clusters.

For example,

```
# loading sample CRS data file
mat = FrovedisCRSMatrix().load("./sample")

# fitting input matrix on kmeans object
kmeans = KMeans(n_clusters=2, verbose=2).fit(mat)
```

Return Value
It simply returns "self" reference.

Note that the call will return quickly, right after submitting the fit request at frovedis server side with a unique model ID for the fit request. It may be possible that the training is not completed at the frovedis server side even though the client scikit-learn side fit() returns.

15.3.1.3 predict(X)

Parameters
X: A scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix.

Purpose
It accepts the test data points (X) and returns the centroid information.

Return Value
It returns a numpy array of integer (int32) type containing the centroid values.

15.3.1.4 save(filename)

Parameters
filename: A string object containing the name of the file on which the target model is to be saved.

Purpose
On success, it writes the model information in the specified file as little-endian binary data. Otherwise, it throws an exception.

Return Value
It returns nothing.

15.3.1.5 load(filename)

Parameters
filename: A string object containing the name of the file having model information to be loaded.

Purpose
It loads the model from the specified file (having little-endian binary data).

Return Value
It simply returns "self" instance.

15.3.1.6 debug_print()

Purpose
It shows the target model information on the server side user terminal. It is mainly used for debugging purpose.

Return Value
It returns nothing.

15.3.1.7 release()

Purpose
It can be used to release the in-memory model at frovedis server.

Return Value
It returns nothing.

Chapter 16

spectral clustering

16.1 NAME

spectral clustering - A clustering algorithm commonly used in EDA (exploratory data analysis), using the spectrum (eigenvalues) of the similarity matrix of the data to perform clustering.

16.2 SYNOPSIS

class frovedis.mllib.cluster.SpectralClustering (n_clusters=8, eigen_solver=None, random_state=None, n_init=10, gamma=1.0, affinity='rbf', n_neighbors=10, eigen_tol=0.0, assign_labels='kmeans', degree=3, coef0=1, kernel_params=None, n_jobs=None, verbose=0, n_iter=100, eps=0.01, n_comp=None, norm_laplacian=True, mode=1, drop_first=False)

16.2.1 Public Member Functions

fit(X, y=None)
fit_predict(X)
get_params()
set_params(params)
load(filename)
get_affinity_matrix()
save(filename)
debug_print()
release()

16.3 DESCRIPTION

Clustering is an unsupervised learning problem whereby we aim to group subsets of entities with one another based on some notion of similarity. In spectral clustering, the data points are treated as nodes of a graph. Thus, clustering is treated as a graph partitioning problem. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters. The components or features are identified as per column order in matrix data. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters.

16.3.1 Detailed Description

16.3.1.1 SpectralClustering()

Parameters

n_clusters: An integer parameter containing the number of required clusters.(Default: 2)
eigen_solver: The eigenvalue decomposition strategy to use. AMG requires pyamg to be installed. It can be faster on very large, sparse problems, but may also lead to instabilities(Default: None)[Internally skipped]
random_state: A pseudo random number generator used for the initialization of the lobpcg eigen vectors decomposition when eigen_solver == 'amg' and by the K-Means initialization.[Internally skipped]
n_init: An integer parameter containing the maximum number of iteration count (Default: 100)
gamma: The value required for computing nearby relational meaningful eigenvalues (Default: 1.0)
affinity: If a string, this may be one of 'nearest_neighbors', 'precomputed' according to input data.(Default: rbf)[Internally skipped]
n_neighbors: Number of neighbors to use when constructing the affinity matrix using the nearest neighbors method.[Internally skipped]
eigen_tol: Stopping criterion for eigen decomposition of the Laplacian matrix when using arpack eigen_solver.[Internally skipped]
assign_labels : The strategy to use to assign labels in the embedding space.[Internally skipped]
degree: Degree of the polynomial kernel. Ignored by other kernels.[Internally skipped]
coef0: Zero coefficient for polynomial and sigmoid kernels.[Internally skipped]
kernel_params: Parameters (keyword arguments) and values for kernel passed as callable object.[Internally skipped]
n_jobs: The number of parallel jobs to run.[Internally skipped]
verbose: An integer object specifying the log level to use. (Default: 0)
n_iter: An integer parameter containing the maximum number of iteration count for kmeans
eps: An double parameter containing the epsilon value for kmeans (Default: 0.01)
n_comp: An integer parameter containing the number of components for clusters (Default: 2)
norm_laplacian: A boolean parameter if set True, then compute normalized Laplacian else not (Default: true)
mode: A parameter required to set the eigen computation method. It can be either 1 or 3, 1 for generic and 3 for shift-invert(Default: 1)
drop_first: A boolean parameter if set True, then drops the first eigenvector. The first eigenvector of a normalized laplacian is full of constants, thus if drop_first is set true, compute (n_comp+1) eigenvectors and will drop the first vector. Otherwise it will calculate n_comp number of eigenvectors(Default: false)

Purpose

It clusters the given data points into a predefined number (n_clusters) of clusters. It simply returns "self" reference.

Return Value

This is a non-blocking call. The control will return quickly, right after submitting the training request at frovedis server side with a SpectralModel object containing a array of labels.

16.3.1.2 fit(X, y=None)

Parameters

X: A scipy dense matrix or any python array-like object or an instance of FrovedisRowmajorMatrix.
y: None (simply ignored in scikit-learn as well).

Purpose

It clusters the given data points (X) into a predefined number (n_clusters) of clusters.

For example,

```
# loading sample Rowmajor data file
mat = FrovedisRowmajorMatrix().load("./sample")

# fitting input matrix on spectral object
spectral = SpectralClustering(n_clusters=2, verbose=2).fit(mat)
```

Return Value

It simply returns "self" reference.
Note that the call will return quickly, right after submitting the fit request at frovedis server side with a unique model ID for the fit request. It may be possible that the training is not completed at the frovedis server side even though the client scikit-learn side fit() returns.

16.3.1.3 fit_predict(X, y=None)

Parameters

X: A scipy dense matrix or any python array-like object or an instance of FrovedisRowmajorMatrix.

Purpose

It accepts the test data points (X) and returns the centroid information.

Return Value

It returns a numpy array of integer (int32) type containing the label values.

16.3.1.4 get_params()

Purpose

It returns the values of parameters used for clustering.

Return Value

It simply returns "self" instance.

16.3.1.5 set_params()

Purpose

It intialize the values of parameters the required for clustering.

Return Value

It simply returns "self" instance.

16.3.1.6 load(filename)

Parameters

filename: A string object containing the name of the file having model information to be loaded.

Purpose

It loads the model from the specified file (having little-endian binary data).

Return Value

It simply returns "self" instance.

16.3.1.7 get_affinity_matrix()

Purpose

It returns the output value of the computed affinity matrix.

Return Value

It returns FrovedisRowmajorMatrix instance.

16.3.1.8 save(filename)

Parameters

filename: A string object containing the name of the file on which the target model is to be saved.

Purpose

On success, it writes the model information in the specified file as little-endian binary data. Otherwise, it throws an exception.

Return Value

It returns nothing.

16.3.1.9 debug_print()

Purpose

It shows the target model information on the server side user terminal. It is mainly used for debugging purpose.

Return Value

It returns nothing.

16.3.1.10 release()

Purpose

It can be used to release the in-memory model at frovedis server.

Return Value

It returns nothing.

Chapter 17

spectral embedding

17.1 NAME

spectral embedding - Spectral embedding is the accurate method for extraction of meaningful patterns in high dimensional data. It forms an affinity matrix given by the specified function and applies spectral decomposition to the corresponding graph laplacian. The resulting transformation is given by the value of the normalized eigenvectors for each data point.

17.2 SYNOPSIS

class frovedis.mllib.cluster.SpectralEmbedding (n_components=2, affinity='rbf', gamma=1.0, random_state=None, eigen_solver=None, n_neighbors=None, n_jobs=None, verbose=0, norm_laplacian=True, mode=1, drop_first=True)

17.2.1 Public Member Functions

fit(X, y=None)
get_params()
set_params(params)
get_affinity_matrix()
get_embedding_matrix()
load(filename)
save(filename)
debug_print()
release()

17.3 DESCRIPTION

Spectral embedding is particularly useful for reducing the dimensionality of data that is expected to lie on a low-dimensional manifold contained within a high-dimensional space, it yields a low-dimensional representation of the data that best preserves the structure of the original manifold in the sense that points that are close to each other on the original manifold will also be close after embedding. At the same time, the embedding emphasizes clusters in the original data.

17.3.1 Detailed Description

17.3.1.1 SpectralEmbedding()

Parameters

n_components: An integer parameter containing the number of component count (Default: 2)
affinity: If a string, this may be one of 'nearest_neighbors', 'precomputed' according to input data.(Default: rbf)[Internally skipped]
gamma: The value required for computing nearby relational meaningful eigenvalues(Default: 1.0)
random_state: A pseudo random number generator used for the initialization of the lobpcg eigen vectors decomposition when eigen_solver == 'amg' and by the K-Means initialization.[Internally skipped]
eigen_solver: The eigenvalue decomposition strategy to use. AMG requires pyamg to be installed. It can be faster on very large, sparse problems, but may also lead to instabilities(Default: None)[Internally skipped]
n_neighbors: Number of neighbors to use when constructing the affinity matrix using the nearest neighbors method.[Internally skipped]
n_jobs: The number of parallel jobs to run.[Internally skipped]
verbose: An integer object specifying the log level to use. (Default: 0)
norm_laplacian: A boolean parameter if set True, then compute normalized Laplacian else not (Default: true)
mode: A parameter required to set the eigen computation method. It can be either 1 or 3, 1 for generic and 3 for shift-invert(Default: 1)
drop_first: A boolean parameter if set True, then drops the first eigenvector. The first eigenvector of a normalized laplacian is full of constants, thus if drop_first is set true, compute (n_comp+1) eigenvectors and will drop the first vector. Otherwise it will calculate n_comp number of eigenvectors(Default: false)

Purpose

After getting the affinity matrix by computing distance co-relation, this is used to extract meaningful patterns in high dimensional data. After the successful embedding, It returns a FrovedisRowmajorMatrix containing the assigned values.

Return Value

This is a non-blocking call. The control will return quickly, right after submitting the training request at frovedis server side with a SpectralEmbedding object containing a FrovedisRowmajorMatrix with meaningful or co-related patterns obtained from eigenvectors.

17.3.1.2 fit(X, y=None)

Parameters

X: A scipy dense matrix or any python array-like object or an instance of FrovedisRowmajorMatrix.
y: None (simply ignored in scikit-learn as well).

Purpose

It extracts meaningful or co-related patterns obtained from normalized eigenvector computation.

For example,

```
# loading sample Rowmajor data file
mat = FrovedisRowmajorMatrix().load("./sample")

# fitting input matrix on embedding object
embedding = SpectralEmbedding(n_components=2, gamma=1.0, mode=1, verbose=2).fit(mat)
```

Return Value

It simply returns "self" reference.
Note that the call will return quickly, right after submitting the fit request at frovedis server side with a unique model ID for the fit request. It may be possible that the training is not completed at the frovedis server side even though the client scikit-learn side fit() returns.

17.3.1.3 get_params()

Purpose

It returns the values of parameters used for embedding.

Return Value

It simply returns "self" instance.

17.3.1.4 set_params()

Purpose

It intialize the values of parameters the required for embedding.

Return Value

It simply returns "self" instance.

17.3.1.5 get_affinity_matrix()

Purpose

It returns the output value of the computed affinity matrix.

Return Value

It returns FrovedisRowmajorMatrix instance.

17.3.1.6 get_embedding_matrix()

Purpose

It returns the output value of the computed normalized embedding matrix.

Return Value

It returns FrovedisRowmajorMatrix instance.

17.3.1.7 load(filename)

Parameters

filename: A string object containing the name of the file having model information to be loaded.

Purpose

It loads the model from the specified file (having little-endian binary data).

Return Value

It returns nothing.

17.3.1.8 save(filename)

Parameters

filename: A string object containing the name of the file on which the target model is to be saved.

Purpose

On success, it writes the model information in the specified file as little-endian binary data. Otherwise, it throws an exception.

Return Value

It returns nothing.

17.3.1.9 debug_print()

Purpose

It shows the target model information on the server side user terminal. It is mainly used for debugging purpose.

Return Value

It returns nothing.

17.3.1.10 release()

Purpose
It can be used to release the in-memory model at frovedis server.

Return Value
It returns nothing.