Manual of Frovedis Python API

Contents

Chapter 1

Introduction

This manual contains Python API documentation. 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.

- Machine Learning
  - Agglomerative Clustering
  - Matrix Factorization using ALS
  - Bernoulli Naive Bayes
  - DBSCAN
  - DecisionTreeClassifier
  - DecisionTreeRegressor
  - FactorizationMachineClassifier
  - FactorizationMachineRegressor
  - FPGrowth
  - GaussianMixture
  - GradientBoostingClassifier
  - GradientBoostingRegressor
  - KMeans Clustering
  - KNeighborsClassifier
  - KNeighborsRegressor
  - Lasso Regression
  - Latent Dirichlet Allocation
  - Linear Regression
  - LinearSVC
  - LinearSVR
  - Logistic Regression
  - Multinomial Naive Bayes
  - NearestNeighbors
  - Principal Component Analysis
  - RandomForestClassifier
  - RandomForestRegressor
  - Ridge Regression
  - SGDClassifier
  - SGDRegressor
  - Spectral Clustering
  - Spectral Embedding
  - StandardScaler
  - SVC

  - t-Distributed Stochastic Neighbor Embedding
  - Timeseries analysis using ARIMA
  - Word2Vec
- Linalg
  - linalg
  - scalapack
- Graph
  - Graph
  - pagerank()
  - Breadth First Search
  - single_source_shortest_path()
  - connected_components()
- Matrix
  - FrovedisDvector
  - FrovedisCRSMatrix
  - FrovedisBlockcyclicMatrix
  - pblas_wrapper
  - scalapack_wrapper
  - getrf_result
  - gesvd_result
- DataFrame
  - DataFrame Introduction
  - DataFrame Indexing Operations
  - DataFrame Generic Functions
  - DataFrame Conversion Functions
  - DataFrame Sorting Functions
  - DataFrame Aggregate Functions
  - DataFrame Math Functions
  - FrovedisGroupedDataframe
  - FrovedisGroupedDataFrame Aggregate Functions
  - [Dataframe: Datetime functionality]

Chapter 2

Agglomerative Clustering

2.1 NAME

Agglomerative Clustering - The most common type of hierarchical clustering used to group objects in clusters based on their similarities.

2.2 SYNOPSIS

```
class frovedis.mllib.cluster.AgglomerativeClustering(n_clusters=2, affinity='euclidean',
                                                     memory=None, connectivity=None,
                                                     compute_full_tree='auto',
                                                     linkage='average',
                                                     distance_threshold=None,
                                                     compute_distances=False,
                                                     verbose=0)
```

2.2.1 Public Member Functions

fit(X, y = None)
fit_predict(X, y = None)
reassign(ncluster = None)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

2.3 DESCRIPTION

Clustering is a machine learning technique that involves the grouping of data points. Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively.

The Agglomerative Clustering object performs a hierarchical clustering using a bottom-up approach. Each observation starts in its own cluster, and clusters are successively merged together.

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 Agglomerative Clustering 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 Agglomerative Clustering 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.

2.3.1 Detailed Description

2.3.1.1 1. AgglomerativeClustering()

Parameters
n_clusters: An integer parameter specifying the number of clusters. The number of clusters should be greater than 0 and less than n_samples. (Default: 2)
affinity: An unused parameter. (Default: 'euclidean')
memory: An unused parameter. (Default: None)
connectivity: An unused parameter. (Default: None)
compute_full_tree: An unused parameter. (Default: 'auto')
linkage: A string parameter used to specify linkage criterion. It determines which distance to use between sets of observation. The algorithm will merge the pairs of clusters that minimize this criterion.
- **'average'**: uses the average of the distances of each observation of the two sets.
- **'complete'**: linkage uses the maximum distances between all observations of the two sets.
- **'single'**: uses the minimum of the distances between all observations of the two sets.

Only 'average', 'complete' and 'single' are supported. (Default: 'average')
distance_threshold: A float or double(float64) type parameter, is the linkage distance threshold above which the clusters will not be merged. It must be zero or positive value. (Default: None)
When it is None (not specified explicitly), it will be set as 0.0.
compute_distances: Unlike Scikit-learn, it is alwats True for frovedis. Hence, this parameter is left unused. (Default: False)
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.

Attributes
n_clusters_: A positive integer value specifying the number of clusters found by the algorithm.
labels_: A python ndarray of int64 type values and has shape **(n_clusters,)**. It contains cluster labels for each point.
children_: A python ndarray of int64 type values and has shape **(n_samples - 1, 2)**. It contains the children of each non-leaf node.
distances_: A python ndarray of float or double(float64) values and has shape **(n_samples - 1,)**. It specifies the distances between nodes in the corresponding place in "children_".
n_connected_components_: An integer value used to provide the estimated number of connected components in the graph.

Purpose
It initializes an Agglomerative Clustering object with the given parameters.

The parameters: “affinity”, “memory”, “connectivity”, “compute_full_tree” and “compute_distances” are simply kept in to to make the interface uniform to the Scikit-learn Agglomerative Clustering module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns “self” reference.

2.3.1.2 2. fit(X, y = 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: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn as well.

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

For example,

```
# loading sample matrix data
mat = np.loadtxt("./input/sample_data.txt")

# fitting input matrix on AgglomerativeClustering object
from frovedis.mllib.cluster import AgglomerativeClustering
acm = AgglomerativeClustering(n_clusters = 2).fit(mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading sample matrix data
mat = np.loadtxt("./input/sample_data.txt")

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# Agglomerative Clustering with pre-constructed frovedis-like inputs
from frovedis.mllib.cluster import AgglomerativeClustering
acm = AgglomerativeClustering(n_clusters = 2).fit(rmat)
```

Return Value
It simply returns “self” reference.

2.3.1.3 3. fit_predict(X, y = 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: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn as well.

Purpose
It clusters the given data points (X) into a predefined number of clusters. In addition to fitting, it returns the cluster labels for each sample in the training set.

For example,

```
# loading sample matrix data
mat = np.loadtxt("./input/sample_data.txt")

# fitting input matrix on AgglomerativeClustering object
from frovedis.mllib.cluster import AgglomerativeClustering
acm = AgglomerativeClustering(n_clusters = 2)
print(acm.fit_predict(mat))
```

Output

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

Like in fit(), frovedis-like input can be used to speed-up the trainng at server side.

For example,

```
# loading sample matrix data
mat = np.loadtxt("./input/sample_data.txt")

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# using pre-constructed input matrix
from frovedis.mllib.cluster import AgglomerativeClustering
acm = AgglomerativeClustering(n_clusters = 2)
print(acm.fit_predict(rmat))
```

Output

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

Return Value
It returns a numpy array of int64 type values containing the cluster labels. It has a shape **(n_samples,)**.

2.3.1.4 4. reassign(ncluster = None)

Parameters
nclusters: An integer parameter specifying the number of clusters to be reassigned for the fitted data without computing the tree again. The number of clusters should be greater than 0 and less than n_samples. (Default: None)
When it is None (not specified explicitly), it simply returns the same cluster labels of already fitted clustering model. In this case, 'ncluster' becomes the 'n_cluster' value used during fit().

Purpose
It accepts the number of clusters (nclusters) in order to make prediction with different "nclusters" on same model at frovedis server.

For example,

```
# On the same AgglomerativeClustering object, predicting labels with new nclusters
print(acm.reassign(nclusters = 3))
```

Output

```
[0 0 1 1 2]
```

Return Value
It returns a numpy array of int64 type values containing the cluster labels. It has a shape **(n_samples,)**.

2.3.1.5 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: A python ndarray or an instance of FrovedisVector containing the true labels for X. It has shape **(n_samples, 1)**.
sample_weight: An unused parameter whose default value is None. It is simply ignored in frovedis implementation.

Purpose
It uses homogeneity score on given test data and labels i.e homogeneity score of self.predict(X, y) wrt. y.

For example,

```
acm.score(train_mat, [0, 0, 1, 1 1])
```

Output

```
1.0
```

Return Value
It returns a homogeneity score of float type.

2.3.1.6 6. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by AgglomerativeClustering. It is used to get parameters and their values of AgglomerativeClustering class.

For example,

```
print(acm.get_params())
```

Output

```
{'affinity': 'euclidean', 'compute_distances': True, 'compute_full_tree': 'auto',
'connectivity': None, 'distance_threshold': None, 'linkage': 'average', 'memory': None,
'n_clusters': 3, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

2.3.1.7 7. set_params(**params)

Parameters
***params**: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by AgglomerativeClustering, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(acm.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
acm.set_params(n_clusters = 4)
print("get parameters after setting:")
print(acm.get_params())
```

Output

```
get parameters before setting:
{'affinity': 'euclidean', 'compute_distances': True, 'compute_full_tree': 'auto',
'connectivity': None, 'distance_threshold': None, 'linkage': 'average', 'memory': None,
'n_clusters': 3, 'verbose': 0}
get parameters after setting:
{'affinity': 'euclidean', 'compute_distances': True, 'compute_full_tree': 'auto',
'connectivity': None, 'distance_threshold': None, 'linkage': 'average', 'memory': None,
'n_clusters': 4, 'verbose': 0}
```

Return Value
It simply returns "self" reference.

2.3.1.8 8. 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 an agglomerative clustering model stored previously from the specified file (having little-endian binary data).

For example,

```
acm.load("./out/MyAcmClusteringModel", dtype = np.float64)
```

Return Value
It simply returns "self" reference.

2.3.1.9 9. 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the agglomerative clustering model
acm.save("./out/MyAcmClusteringModel")
```

This will save the agglomerative clustering model on the path "/out/MyAcmClusteringModel".
It would raise exception if the directory already exists with same name.

The 'MyAcmClusteringModel' directory has

MyAcmClusteringModel
|—— metadata
|—— **model**

The metadata file contains the number of clusters, number of samples, model kind, input datatype used for trained model.
Here, the **model** directory contains information about dendogram.

Return Value
It returns nothing.

2.3.1.10 10. debug_print()

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

For example,

```
acm.debug_print()
```

Output

```
--- dendrogram ---
        X       Y       distance        size
5:      2       3       0.173205        2
6:      0       1       0.173205        2
7:      4       5       0.259808        3
8:      6       7       15.5019 5
```

This output will be visible on server side. It displays the dendrogram on the trained model which is currently present on the server. Using the dendrogram, the desired number of clusters may be found.

No such output will be visible on client side.

Return Value
It returns nothing.

2.3.1.11 11. release()

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

For example,

```
acm.release()
```

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

Return Value
It returns nothing.

2.3.1.12 12. is_fitted()

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

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

2.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **DBSCAN in Frovedis**
- **KMeans in Frovedis**
- **Spectral Clustering in Frovedis**

Chapter 3

Matrix Factorization using ALS

3.1 NAME

Matrix Factorization using Alternating Least Square (ALS) - is a matrix factorization algorithm commonly used for recommender systems.

3.2 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)
```

3.2.1 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()

3.3 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.

3.3.1 Detailed Description

3.3.1.1 1. ALS()

rank: A positive 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. It must be within the range of 0 to max(M, N).
max_iter: A positive integer specifying maximum iteration count. (Default: 100)
alpha: A positive double(float64) parameter containing the learning rate. (Default: 0.01)
reg_param: A positive double(float64) parameter, also called as the regularization parameter. (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: An int64 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.

3.3.1.2 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, 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, especially 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.

3.3.1.3 3. 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,

```
# prints 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.

3.3.1.4 4. recommend_users(pid, k)

Parameters
pid: An integer parameter specifying the product ID (0-based Id) 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.

3.3.1.5 5. recommend_products(uid, k)

Parameters
uid: An integer parameter specifying the user ID (0-based Id) 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.

3.3.1.6 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(metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

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

The MyALSModel contains below directory structure:
MyALSModel
|——metadata
|——**model**
|——**X**
|——**Y**

'metadata' represents the detail about model_kind and datatype of training vector. Here, the **model** directory contains information about user and product features.

If the directory already exists with the same name then it will raise an exception.

Return Value
It returns nothing.

3.3.1.7 7. 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/MyALSModel")
```

Return Value
It simply returns "self" instance.

3.3.1.8 8. 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
```

This output will be visible on server side. It will print the matrix and labels of training data.

No such output will be visible on client side.

Return Value
It returns nothing.

3.3.1.9 9. 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.

3.3.1.10 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 model first.

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

3.4 SEE ALSO

- **Introduction to FrovedisCRSMatrix**

Chapter 4

Timeseries analysis using ARIMA

4.1 NAME

ARIMA - Autoregressive Integrated Moving Average model is a time series model that is used to forecast data based on the dataset of past to predict/forecast the future.

4.2 SYNOPSIS

```
class frovedis.mllib.tsa.arima.model.ARIMA(endog, exog=None, order=(1, 0, 0),
                                           seasonal_order=(0, 0, 0, 0),
                                           trend=None,
                                           enforce_stationarity=True,
                                           enforce_invertibility=True,
                                           concentrate_scale=False, trend_offset=1,
                                           dates=None,freq=None, missing='none',
                                           validate_specification=True,
                                           seasonal=None, auto_arima=False,
                                           solver='lapack', verbose=0)
```

4.2.1 Public Member Functions

fit()
predict(start = None, end = None, dynamic = False, **kwargs)
forecast(steps = 1, exog = None, alpha = 0.05)
get_params(deep = True)
set_params(**params)
release()
is_fitted()

4.3 DESCRIPTION

Frovedis provides a timeseries model in order to predict the future values based on the past values.
Each component in ARIMA functions as a parameter with a standard notation. For ARIMA models, a standard notation would be ARIMA with p(AR), d(I), and q(MA) which indicate the type of ARIMA model

to be used. A `0` value can be used as a parameter and would mean that the particular component should not be used in the model. This way, the ARIMA model can be constructed to perform the function of an ARMA model, or even simple AR (1,0,0), I(0,1,0), or MA(0,0,1) models. **However, the current implementaion cannot be used to construct a pure MA model.**

Also, it provides the feature of auto ARIMA which can fit the best lag for AR and MA. This is a useful feature for the users who do not have knowledge about data analytics.

Unlike statsmodel ARIMA, it does not use MLE (Maximum Likelihood Estimation), rather it uses OLS (Ordinary Least Squares).
Note:- Also, rather than converging around the mean after some number of predictions, it tends to follow the trend i.e it diverges towards increasing or decreasing trend.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis interface is almost same as statsmodel ARIMA interface, but it does not have any dependency with statsmodel. It can be used simply even if the system does not have statsmodel 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 ARIMA 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.

4.3.1 Detailed Description

4.3.1.1 1. ARIMA()

Parameters
endog: It contains the timeseries data.
Currently, it accepts the below mentioned array-like inputs such as:
- A numpy array of shape (n_samples,)
- A list or tuple of shape (n_samples,)
- A matrix of shape (n_samples, 1)

Also, it accepts a pandas Series or DataFrame having single column with a numeric or datetime index.
Currently, it does not support frovedis DataFrame as timeseries data.
exog: An unused parameter. (Default: =None)
order: A tuple parameter having 3 elements (p,d,q) that specifies the order of the model for the autoregressive(p), differences(d), and moving average(q) components. (Default: (1, 0, 0))
Currently, autoregressive order cannot be 0 i.e it cannot be used to create pure MA model.
Also, these components (p, d, q) of the model cannot be negative values.
seasonal_order: An unused parameter. (Default: (0, 0, 0, 0))
trend: An unused parameter. (Default: None)
enforce_stationarity: An unused parameter. (Default: True)
enforce_invertibility: An unused parameter. (Default: True)
concentrate_scale: An unused parameter. (Default: False)
trend_offset: An unused parameter. (Default: 1)
dates: An unused parameter. (Default: None)
freq: This parameter specifies the frequency of the timeseries. It is an offset string or Pandas offset. It is used only when time series data is a pandas Series or DataFrame having an index. For example, freq = '3D' or freq = to_offset('3D'). (Default: None)
missing: An unused parameter. (Default: 'none')

validate_specification: An unused parameter. (Default: True)
seasonal: A zero or a positive integer parameter that specifies the interval of seasonal differencing. In case the data has some seasonality , then it can handle it . (Default: None)
auto_arima: A boolean parameter that specifies whether to use auto (brute) ARIMA. (Default: False)
If set to True, it treats the autoregressive and moving average component of the order parameter as the highest limit for its iteration and auto fits these components with the best RMSE score.
solver: A string object parameter which specifies the solver to be used for linear regression. It supports `lapack`, `scalapack`, `lbfgs` and `sag` solvers. (Default: 'lapack')
When specified, e.g `lbfgs`, then it uses lbfgs solver for linear regression.
Note:- To get the best performance and accuracy from ARIMA, use solver='lapack'.
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. (Default: 0)

Attributes
fittedvalues:
When endog is an array-like input:
It is a numpy array (containing float or double (float64) typed values depending on data-type of input array) of shape (n_samples,)
When it is a pandas Series or DataFrame:
It is a pandas Series containing the predicted values of the model after training is completed.

Purpose
It initializes the ARIMA instance with the given parameters.

The parameters: "exog", "seasonal_order", "trend", "enforce_stationarity", "enforce_invertibility", "concentrate_scale", "trend_offset", "dates" and "missing" are simply kept in to to make the interface uniform to the statsmodel ARIMA module. They are not used anywhere within the frovedis implementation.

Currently, the number of samples in the timeseries data must be greater than sum of ARIMA order and seasonal parameter.

```
len(endog) >= (order[0] + order[1] + order[2] + seasonal)
```

Return Value
It simply returns "self" reference.

4.3.1.2 2. fit()

Purpose
It is used to fit the model parameters on the basis of given parameters and data provided at frovedis server.

For example,

```
# loading an array-like data
import numpy as np
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5,
                 231.8, 224.5, 192.8, 122.9, 336.5, 185.9,
                 194.3, 149.5, 210.1, 273.3, 191.4, 287,
                 226, 303.6, 289.9, 421.6, 264.5, 342.3,
                 339.7, 440.4, 315.9, 439.3, 401.3, 437.4,
                 575.5, 407.6, 682, 475.3, 581.3, 646.9])

# fitting input array-like data on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(data, order=(2,1,2)).fit()
```

```
# displaying the fittedvalues
print(arima.fittedvalues)
```

Output

```
[  0.           0.           0.          34.80616713  82.04476428
 199.9205314  197.46227245 198.75614907 229.51624583 263.0131258
 270.18526491 192.05449775 272.3933017  284.84889246 266.61188466
 226.95908704 211.39748037 271.90632761 256.99495549 290.21370532
 281.66600813 303.1955228  304.63779347 393.97882909 375.17286126
 364.56323191 359.48470979 426.81421461 402.55377056 427.14878515
 442.66867406 432.34417321 540.18491169 486.09468617 596.96746024
 589.29064598]
```

Here, fittedvalues will be displayed after training is completed on array-like timeseries data.

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 input can be used to speed up the training time, especially when same data would be used for multiple executions.

For example,

```
# loading an array-like data
import numpy as np
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5,
                 231.8, 224.5, 192.8, 122.9, 336.5, 185.9,
                 194.3, 149.5, 210.1, 273.3, 191.4, 287,
                 226, 303.6, 289.9, 421.6, 264.5, 342.3,
                 339.7, 440.4, 315.9, 439.3, 401.3, 437.4,
                 575.5, 407.6, 682, 475.3, 581.3, 646.9])

# Since "data" is numpy array, we have created FrovedisDvector.
from frovedis.matrix.dvector import FrovedisDvector
data = FrovedisDvector(data)

# ARIMA with pre-constructed frovedis-like input
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(data, order=(2,1,2)).fit()

# displaying the fittedvalues
print(arima.fittedvalues)
```

Output

```
[  0.           0.           0.          34.80616713  82.04476428
 199.9205314  197.46227245 198.75614907 229.51624583 263.0131258
 270.18526491 192.05449775 272.3933017  284.84889246 266.61188466
 226.95908704 211.39748037 271.90632761 256.99495549 290.21370532
 281.66600813 303.1955228  304.63779347 393.97882909 375.17286126
 364.56323191 359.48470979 426.81421461 402.55377056 427.14878515
 442.66867406 432.34417321 540.18491169 486.09468617 596.96746024
 589.29064598]
```

Here, fittedvalues will be displayed after training is completed on frovedis-like timeseries data.

When pandas dataframe having single column and a numeric or datetime index is provided, then training is done based on the given index type.

If it is a numeric index with values 0, 1 , . . . , N-1 (always incrementing indices), where N is n_samples

or if it is (coerceable to) a DatetimeIndex or PeriodIndex with an associated frequency, then it is called a "Supported" index. Otherwise, it is called an "Unsupported" index.

For numeric indices type, it can be pandas Index or RangeIndex instances.
Here, for RangeIndex instances the indices can be in form:
- Indices with 0,2,4,. . . .,N-2
For example: pandas.RangeIndex(start=0, stop=10, step=2)
- Indices with 10,12,14,. . . .,N-2
For example: pandas.RangeIndex(start=10, stop=20, step=2)

They should be monotonically increasing and not need required to start with 0.
When Index instance is used as numeric indices, then they have to be monotonically increaing and starting with 0.
For example: pandas.Index([0,1,2,3,4,5])

NOTE: A warning will be given when unsupported indices are used for training.

When a supported index such as Datetimeindex instance is provided:

For example,

```
# loading a pandas dataframe having datetime index
# such indices should be monotonically increasing
# and have an associated frequency
# also, such index is considered as a supported index
import pandas as pd
index = ["01-01-1981", "01-04-1981", "01-07-1981", "01-10-1981", "01-13-1981", "01-16-1981",
         "01-19-1981", "01-22-1981", "01-25-1981", "01-28-1981", "01-31-1981", "02-03-1981",
         "02-06-1981", "02-09-1981", "02-12-1981", "02-15-1981", "02-18-1981", "02-21-1981"]

index = pd.DatetimeIndex(data = index, name="Date", freq=None)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# displaying the fittedvalues
print(arima.fittedvalues)
```

Output

```
UserWarning: A date index has been provided, but it has no associated frequency information
and so will be ignored when e.g. forecasting.

Date
1981-01-01      0.000000
1981-01-04      0.000000
1981-01-07      0.000000
1981-01-10      7.183812
1981-01-13     50.686614
1981-01-16    188.205611
1981-01-19    171.793531
1981-01-22    182.706200
1981-01-25    200.586396
```

```
1981-01-27    227.986895
1981-01-31    233.641814
1981-02-03    213.975698
1981-02-06    213.622244
1981-02-09    254.024920
1981-02-12    242.193347
1981-02-15    216.585066
1981-02-18    198.927653
1981-02-21    224.250625
dtype: float64
```

Here, fittedvalues will be displayed after training is completed on dataframe having supported index.

Note: For fittedvalues attribute, when endog is having an index, then after training is completed,

```
fittedvalues.index = endog.index
```

This will always be true even when endog is having supported or unsupported indices.

Also, in above example frequency information was inferred based on given timeseries data.

So, we can use **freq** parameter in frovedis ARIMA in order to set the frequency information of the timeseries data as well.

For example,

```
# loading a pandas dataframe having datetime index
import pandas as pd
index = ["01-01-1981", "01-04-1981", "01-07-1981", "01-10-1981", "01-13-1981", "01-16-1981",
         "01-19-1981", "01-22-1981", "01-25-1981", "01-28-1981", "01-31-1981", "02-03-1981",
         "02-06-1981", "02-09-1981", "02-12-1981", "02-15-1981", "02-18-1981", "02-21-1981"]

index = pd.DatetimeIndex(data = index, name="Date", freq=None)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object and using freq parameter
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2), freq = '3D').fit()

# displaying the fittedvalues
print(arima.fittedvalues)
```

Output

```
Date
1981-01-01      0.000000
1981-01-04      0.000000
1981-01-07      0.000000
1981-01-10      7.183812
1981-01-13     50.686614
1981-01-16    188.205611
1981-01-19    171.793531
1981-01-22    182.706200
1981-01-25    200.586396
1981-01-27    227.986895
1981-01-31    233.641814
```

```
1981-02-03    213.975698
1981-02-06    213.622244
1981-02-09    254.024920
1981-02-12    242.193347
1981-02-15    216.585066
1981-02-18    198.927653
1981-02-21    224.250625
dtype: float64
```

Note: Here, the frequency information provided for freq parameter must be same as the frequency inferred for the timeseries data. Otherwise, it will raise an exception.

When a supported index such as RangeIndex instance is provided:

For example,

```
# loading a pandas DataFrame having numeric index as RangeIndex
# such indices should be monotonically increasing
# and need not start with 0.
# also, such index is considered as a supported index
import pandas as pd
index = pd.RangeIndex(start=10, stop=46, step=2)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# displaying the fittedvalues
print(arima.fittedvalues)
```

Output

```
10      0.000000
12      0.000000
14      0.000000
16      7.183812
18     50.686614
20    188.205611
22    171.793531
24    182.706200
26    200.586396
28    227.986895
30    233.641814
32    213.975698
34    213.622244
36    254.024920
38    242.193347
40    216.585066
42    198.927653
44    224.250625
dtype: float64
```

When an unsupported index such as DatetimeIndex with no associated frequencyis provided:

For example,

```
# loading a pandas dataframe having datetime index
# and having no associated frequency
# such index is considered as an unsupported index
import pandas as pd
index = ["01-01-1981", "01-05-1981", "01-07-1981", "01-10-1981", "01-12-1981", "01-16-1981",
         "01-19-1981", "01-21-1981", "01-25-1981", "01-28-1981", "01-31-1981", "02-03-1981",
         "02-06-1981", "02-10-1981", "02-12-1981", "02-16-1981", "02-18-1981", "02-22-1981"]

index = pd.DatetimeIndex(data = index, name="Date", freq=None)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# displaying the fittedvalues
print(arima.fittedvalues)
```

Output

```
UserWarning: A date index has been provided, but it has no associated frequency information and so
will be ignored when e.g. forecasting.

Date
1981-01-01      0.000000
1981-01-05      0.000000
1981-01-07      0.000000
1981-01-10      7.183812
1981-01-12     50.686614
1981-01-16    188.205611
1981-01-19    171.793531
1981-01-21    182.706200
1981-01-25    200.586396
1981-01-28    227.986895
1981-01-31    233.641814
1981-02-03    213.975698
1981-02-06    213.622244
1981-02-10    254.024920
1981-02-12    242.193347
1981-02-16    216.585066
1981-02-18    198.927653
1981-02-22    224.250625
dtype: float64
```

Here, indices in the `fittedvalues` are same as indices present in the `endog` timeseries data.

When an unsupported index such as Index instance having indices being without 0 is provided:

For example,

```
# loading a pandas DataFrame having numeric index as Index
# ideally, such indices should be monotonically increasing
```

```
# and should start with 0.
# otherwise ,such index are considered as an unsupported index
import pandas as pd
index = pd.Index([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18])
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# displaying the fittedvalues
print(arima.fittedvalues)
```

Output

```
UserWarning: An unsupported index was provided and will be ignored when e.g. forecasting.

1       0.000000
2       0.000000
3       0.000000
4       7.183812
5      50.686614
6     188.205611
7     171.793531
8     182.706200
9     200.586396
10    227.986895
11    233.641814
12    213.975698
13    213.622244
14    254.024920
15    242.193347
16    216.585066
17    198.927653
18    224.250625
dtype: float64
```

Return Value
It simply returns "self" reference.

4.3.1.3 3. predict(start = None, end = None, dynamic = False, **kwargs)

Parameters
start: This parameter can be an integer, string or datetime instance. It specifies the starting index from which the values are to be predicted.
stop: This parameter can be an integer, string or datetime instance. It specifies the index till which the values are to be predicted.
dynamic: An unused parameter. (Default: False)
****kwargs***: An unused parameter.

Purpose
It is used to perform in-sample prediction and out-of-sample prediction at frovedis server.

During prediction, end index must not be less than start index.

Below mentioned examples show frovedis ARIMA to be used to perform in-sample and out-sample predictions.

When endog is array-like input, then start and end must only be integers to perform in-sample and out-sample predictions.

For example,

```
# loading an array-like data
import numpy as np
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5,
                 231.8, 224.5, 192.8, 122.9, 336.5, 185.9,
                 194.3, 149.5, 210.1, 273.3, 191.4, 287,
                 226, 303.6, 289.9, 421.6, 264.5, 342.3,
                 339.7, 440.4, 315.9, 439.3, 401.3, 437.4,
                 575.5, 407.6, 682, 475.3, 581.3, 646.9])

# fitting input array-like data on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(data, order=(2,1,2)).fit()

# perform in-sample prediction with start and end as integers
print('In-sample predictions: ', arima.predict(start=11, end=12))
```

Output

```
In-sample predictions:  [169.65455872 290.43805859]
```

When start and end are negative indices to perform in-sample prediction:

For example,

```
# perfrom in-sample prediction with start and end as negative indices
print('In-sample prediction with negative indices: ', arima.predict(start=-2, end=-1))
```

Output

```
In-sample prediction with negative indices:  [627.92270498 578.69942377]
```

Note: Here, negative indices can only be used to perform in-sample predictions.

When start and end are integers to perfrom out-sample prediction:

For example,

```
# perform out-sample prediction with start and end as integers
print('Out-sample predictions: ', arima.predict(start=40, end=41))
```

Output

```
Out-sample predictions:  [672.66044638 683.2480911 ]
```

When endog is a dataframe having a supported index (datatime index which is monotonically increasing and has associated frequency), then both `start` and `end` can be integer, dates as string or datetime instance to perform in-sample and out-sample predictions.

For example,

```
# loading a pandas dataframe having a supported index
import pandas as pd
index = ["01-01-1981", "01-04-1981", "01-07-1981", "01-10-1981", "01-13-1981", "01-16-1981",
         "01-19-1981", "01-22-1981", "01-25-1981", "01-28-1981", "01-31-1981", "02-03-1981",
         "02-06-1981", "02-09-1981", "02-12-1981", "02-15-1981", "02-18-1981", "02-21-1981"]
```

```
index = pd.DatetimeIndex(data = index, name="Date", freq=None)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# perform in-sample prediction with start and end as integers
print('In-sample predictions: ', arima.predict(start=11, end=12))
```

Output

```
UserWarning: No frequency information was provided, so inferred frequency 3D will be used.

In-sample predictions:  Date
1981-02-03    192.400565
1981-02-06    255.586914
Freq: 3D, dtype: float64
```

Note: When integer are provided as start or end, then these are referred here as positional indices.

For example,

```
# perform out-sample prediction with start and end as integers
print('Out-sample predictions: ', arima.predict(start=22, end=24))
```

Output

```
UserWarning: No frequency information was provided, so inferred frequency 3D will be used.

Out-sample predictions:  1981-03-08    294.413814
1981-03-11    297.292898
1981-03-14    303.964832
Freq: 3D, dtype: float64
```

When start and end are both dates as string to perform predictions:

For example,

```
# perform in-sample prediction with start and end both are dates as strings
print('In-sample predictions: ', arima.predict(start='01-28-1981', end='02-03-1981'))
```

Output

```
UserWarning: No frequency information was provided, so inferred frequency 3D will be used.

In-sample predictions:  Date
1981-01-28    243.389069
1981-01-31    232.545345
1981-02-03    235.522806
Freq: 3D, dtype: float64
```

For example,

```
# perform out-sample prediction with start and end both are dates as strings
print('Out-sample predictions: ', arima.predict(start='02-24-1981', end='02-24-1981'))
```

Output

```
UserWarning: No frequency information was provided, so inferred frequency 3D will be used.

Out-sample predictions:  1981-02-24    259.295505
Freq: 3D, dtype: float64
```

When start and end are both datetime instance to perform predictions:

For example,

```
# perform in-sample prediction with start and end both are dates as datetime instance
from datetime import datetime
print('In-sample predictions: ', arima.predict(start=datetime(1981,1,28), end=datetime(1981,2,3)))
```

Output

```
UserWarning: No frequency information was provided, so inferred frequency 3D will be used.

In-sample predictions:  Date
1981-01-28    243.389069
1981-01-31    232.545345
1981-02-03    235.522806
Freq: 3D, dtype: float64
```

For example,

```
# perform out-sample prediction with start and end both are dates as datetime instance
from datetime import datetime
print('Out-sample predictions: ', arima.predict(start=datetime(1981,2,24), end=datetime(1981,2,24)))
```

Output

```
UserWarning: No frequency information was provided, so inferred frequency 3D will be used.

Out-sample predictions:  1981-02-24    259.295505
Freq: 3D, dtype: float64
```

When endog is a dataframe having a supported index (numeric index such as RangeIndex), then both `start` and `end` can only be integer values in order to perform in-sample and out-sample predictions.

Here, the start or end provided as integers will refer here as positional indices.

For example,

```
# loading a pandas dataframe having a supported index
import pandas as pd
index = pd.RangeIndex(start=10, stop=46, step=2)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# perform in-sample prediction with start and end as integers
print('In-sample predictions: ', arima.predict(start=11, end=12))
```

Output

```
In-sample predictions:  32    192.400565
```

```
34    255.586914
dtype: float64
```

For example,

```
# perform out-sample prediction with start and end as integers
print('Out-sample predictions: ', arima.predict(start=22, end=24))
```

Output

```
Out-sample predictions:  54    294.413814
56    297.292898
58    303.964832
dtype: float64
```

When endog is a dataframe having an unsupported index (datetime index having no associated frequency), then only integer is used to perform in-sample and out-sample predictions. For using dates as string and datetime instance to perform predictions, then only in-sample prediction can be done. No out-sample prediction can be done when using dates as string or datetime instance.

Also, the start or end provided as integers will refer here as positional indices.

For example,

```
# loading a pandas dataframe having an unsupported index
import pandas as pd
index = ["01-01-1981", "01-05-1981", "01-07-1981", "01-10-1981", "01-12-1981", "01-16-1981",
         "01-19-1981", "01-21-1981", "01-25-1981", "01-28-1981", "01-31-1981", "02-03-1981",
         "02-06-1981", "02-10-1981", "02-12-1981", "02-16-1981", "02-18-1981", "02-22-1981"]
index = pd.DatetimeIndex(data = index, name="Date", freq=None)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# perform in-sample prediction with start and end as integers
print('In-sample predictions: ', arima.predict(start=11, end=12))
```

Output

```
UserWarning: A date index has been provided, but it has no associated frequency information and
so will be ignored when e.g. forecasting.

In-sample predictions:  Date
1981-02-03    192.400565
1981-02-06    255.586914
dtype: float64
```

Note: When integer are provided as start or end, then these are referred here as positional indices.

For example,

```
# perform out-sample prediction with start and end as integers
print('Out-sample predictions: ', arima.predict(start=22, end=24))
```

Output

```
UserWarning: A date index has been provided, but it has no associated frequency information and
so will be ignored when e.g. forecasting.

Out-sample predictions:  22    294.413814
23    297.292898
24    303.964832
dtype: float64
```

When start and end are both dates as string to perform predictions:

For example,

```
# perform in-sample prediction with start and end both are dates as strings
print('In-sample predictions: ', arima.predict(start='01-07-1981', end='01-16-1981'))
```

Output

```
UserWarning: A date index has been provided, but it has no associated frequency information and
so will be ignored when e.g. forecasting.

In-sample predictions:  Date
1981-01-07      0.000000
1981-01-10      7.183812
1981-01-12     50.686614
1981-01-16    188.205611
dtype: float64
```

When start and end are both datetime instance to perform predictions:

For example,

```
# perform in-sample prediction with start and end both are dates as datetime instance
from datetime import datetime
print('In-sample predictions: ', arima.predict(start=datetime(1981,1,28), end=datetime(1981,2,3)))
```

Output

```
UserWarning: A date index has been provided, but it has no associated frequency information and
so will be ignored when e.g. forecasting.

In-sample predictions:  Date
1981-01-28    243.389069
1981-01-31    232.545345
1981-02-03    235.522806
dtype: float64
```

When endog is a dataframe having an unsupported index (Index instance having indices being without 0), then only integer is used to perform in-sample and out-sample predictions.

Also, the start or end provided as integers will refer here as positional indices.

For example,

```
# loading a pandas dataframe having an unsupported index
import pandas as pd
index = pd.Index([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18])
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)
```

```
# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# perform in-sample prediction with start and end as integers
print('In-sample predictions: ', arima.predict(start=11, end=12))
```

Output

```
UserWarning: An unsupported index was provided and will be ignored when e.g. forecasting.

In-sample predictions:  12    192.400565
13    255.586914
dtype: float64
```

For example,

```
# perform out-sample prediction with start and end as integers
print('Out-sample predictions: ', arima.predict(start=22, end=24))
```

Output

```
UserWarning: An unsupported index was provided and will be ignored when e.g. forecasting.

Out-sample predictions:  22    294.413814
23    297.292898
24    303.964832
dtype: float64
```

Return Value
When endog is array-like:
- It returns a numpy array of shape (n_predictions,)

When endog is a pandas Series or DataFrame:
- It returns a pandas Series having an index and data column. Number of samples in the pandas Series are equal to the number of number of predictions.

4.3.1.4 4. forecast(steps = 1, exog = None, alpha = 0.05)

Parameters
steps: This parameter can be a positive integer, string or datetime instance. It specifies the number of out of sample values to be predicted. (Default: 1)
NOTE: When endog is array-like, steps must be a positive integer and is greater than or equal to 1. If endog is a dataframe having date indices with associated frequency (known as supported index), then steps can also be dates as string or datetime instance during out-of-sample forecasting.

When endog has unsupported indices, only integer must be used for out-of-sample forecasting. Also, a warning will be given when such indices are used for forecasting. For other types provided, it will raise an exception.
exog: An unused parameter. (Default: None)
alpha: An unused parameter. (Default: 0.05)

Purpose
It is used to perform out of sample forecasting.

Below mentioned examples show frovedis ARIMA to be used to perform forecasting.

When endog is array-like input, then steps must only be an integer to perform forecasting.

For example,

```
# loading an array-like data
import numpy as np
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5,
                 231.8, 224.5, 192.8, 122.9, 336.5, 185.9,
                 194.3, 149.5, 210.1, 273.3, 191.4, 287,
                 226, 303.6, 289.9, 421.6, 264.5, 342.3,
                 339.7, 440.4, 315.9, 439.3, 401.3, 437.4,
                 575.5, 407.6, 682, 475.3, 581.3, 646.9])

# fitting input array-like data on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(data, order=(2,1,2)).fit()

# perform forecasting with steps as an integer
print('forecast(): ', arima.forecast(steps=2))
```

Output

```
forecast():  [578.26315696 654.88723312]
```

When endog is a dataframe having a supported index (datetime index which is monotonically increasing and associated frequency), then `steps` can be an integer, dates as string or datetime instance to perform forecasting.

For example,

```
# loading a pandas dataframe having a supported index
import pandas as pd
index = ["01-01-1981", "01-04-1981", "01-07-1981", "01-10-1981", "01-13-1981", "01-16-1981",
         "01-19-1981", "01-22-1981", "01-25-1981", "01-28-1981", "01-31-1981", "02-03-1981",
         "02-06-1981", "02-09-1981", "02-12-1981", "02-15-1981", "02-18-1981", "02-21-1981"]

index = pd.DatetimeIndex(data = index, name="Date", freq=None)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# perform forecasting with steps as an integer
print('forecast(): ', arima.forecast(steps=2))
```

Output

```
UserWarning: No frequency information was provided, so inferred frequency 3D will be used.

forecast():  1981-02-24    259.295505
1981-02-27    280.647172
Freq: 3D, dtype: float64
```

When steps is a date as a string:

For example,

```
# perform forecasting where steps is a date as a string
print('forecast(): ', arima.forecast(steps='02-24-1981'))
```

Output

```
forecast():  1981-02-24    259.295505
Freq: 3D, dtype: float64
```

When steps is a date as a datetime instance:

For example,

```
# perform forecasting where steps is a dates as a datetime instance
from datetime import datetime
print('forecast(): ', arima.forecast(steps=datetime(1981,2,24)))
```

Output

```
forecast():  1981-02-24    259.295505
Freq: 3D, dtype: float64
```

When endog is a dataframe having a supported index (numeric index such as RangeIndex), then `steps` can only be an integer in order to perform forecasting.

For example,

```
# loading a pandas dataframe having a supported index
import pandas as pd
index = pd.RangeIndex(start=10, stop=46, step=2)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# perform forecasting with steps as an integer
print('forecast(): ', arima.forecast(steps=2))
```

Output

```
forecast():  46    259.295505
48    280.647172
```

dtype: float64

When endog is a dataframe having an unsupported index (datetime index having no associated frequency), then only integer is used to perform forecasting. No forecasting can be done when using dates as string or datetime instance.

For example,

```
# loading a pandas dataframe having a supported index
import pandas as pd
index = ["01-01-1981", "01-05-1981", "01-07-1981", "01-10-1981", "01-12-1981", "01-16-1981",
         "01-19-1981", "01-21-1981", "01-25-1981", "01-28-1981", "01-31-1981", "02-03-1981",
         "02-06-1981", "02-10-1981", "02-12-1981", "02-16-1981", "02-18-1981", "02-22-1981"]

index = pd.DatetimeIndex(data = index, name="Date", freq=None)
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
```

```
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# perform forecasting with steps as an integer
print('forecast(): ', arima.forecast(steps=2))
```

Output

UserWarning: A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. forecasting.

forecast(): 18 259.295505
19 280.647172
dtype: float64

When endog is a dataframe having an unsupported index (datetime index having no associated frequency), then only integer is used to perform forecasting.

For example,

```
# loading a pandas dataframe having an unsupported index
import pandas as pd
index = pd.Index([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18])
data = np.array([266, 145.9, 183.1, 119.3, 180.3, 168.5, 231.8,
                 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5,
                 210.1, 273.3, 191.4, 287])
df = pd.DataFrame({'Temp': data}, index = index)

# fitting input dataframe on ARIMA object
from frovedis.mllib.tsa.arima.model import ARIMA
arima = ARIMA(df, order=(2,1,2)).fit()

# perform forecasting with steps as an integer
print('forecast(): ', arima.forecast(steps=2))
```

Output

```
UserWarning: An unsupported index was provided and will be ignored when e.g. forecasting.

forecast():  18    259.295505
19    280.647172
dtype: float64
```

Retur Value
When endog is array-like:
- It returns a numpy array of shape (steps,)

When endog is a pandas Series or DataFrame:
- It returns a pandas Series having an index and data column.

4.3.1.5 5. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by ARIMA. It is used to get parameters and their values of ARIMA class.

For example,

```
print(arima.get_params())
```

Output

{'auto_arima': False, 'concentrate_scale': False, 'dates': None, 'endog': array([266. , 145.9, 183.1, 119.3, 180.3, 168.5, 231.8, 224.5, 192.8, 122.9, 336.5, 185.9, 194.3, 149.5, 210.1, 273.3, 191.4, 287. , 226. , 303.6, 289.9, 421.6, 264.5, 342.3, 339.7, 440.4, 315.9, 439.3, 401.3, 437.4, 575.5, 407.6, 682. , 475.3, 581.3, 646.9]), 'enforce_invertibility': True, 'enforce_stationarity': True, 'exog': None, 'freq': None, 'missing': 'none', 'order': (2, 1, 2), 'seasonal': 0, 'seasonal_order': (0, 0, 0, 0), 'solver': 'lapack', 'trend': None, 'trend_offset': 1, 'validate_specification': True, 'verbose': 0}

Return Value
A dictionary of parameter names mapped to their values.

4.3.1.6 6. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by ARIMA, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(arima.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
arima.set_params(order=(1,1,1), solver='lbfgs')
print("get parameters after setting:")
print(arima.get_params())
```

Output

```
get parameters before setting:
{'auto_arima': False, 'concentrate_scale': False, 'dates': None,
 'endog': array([266. , 145.9, 183.1, 119.3, 180.3, 168.5, 231.8, 224.5, 192.8,
                 122.9, 336.5, 185.9, 194.3, 149.5, 210.1, 273.3, 191.4, 287. ,
                 226. , 303.6, 289.9, 421.6, 264.5, 342.3, 339.7, 440.4, 315.9,
                 439.3, 401.3, 437.4, 575.5, 407.6, 682. , 475.3, 581.3, 646.9]),
 'enforce_invertibility': True, 'enforce_stationarity': True, 'exog': None, 'freq': None,
 'missing': 'none', 'order': (2, 1, 2), 'seasonal': 0, 'seasonal_order': (0, 0, 0, 0),
 'solver': 'lapack', 'trend': None, 'trend_offset': 1, 'validate_specification': True,
 'verbose': 0}

get parameters after setting:
{'auto_arima': False, 'concentrate_scale': False, 'dates': None,
 'endog': array([266. , 145.9, 183.1, 119.3, 180.3, 168.5, 231.8, 224.5, 192.8,
                 122.9, 336.5, 185.9, 194.3, 149.5, 210.1, 273.3, 191.4, 287. ,
                 226. , 303.6, 289.9, 421.6, 264.5, 342.3, 339.7, 440.4, 315.9,
                 439.3, 401.3, 437.4, 575.5, 407.6, 682. , 475.3, 581.3, 646.9]),
```

```
'enforce_invertibility': True, 'enforce_stationarity': True, 'exog': None, 'freq': None,
'missing': 'none', 'order': (1, 1, 1), 'seasonal': 0, 'seasonal_order': (0, 0, 0, 0),
'solver': 'lbfgs', 'trend': None, 'trend_offset': 1, 'validate_specification': True,
'verbose': 0}
```

Return Value
It simply returns "self" reference.

4.3.1.7 7. release()

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

For example,

```
arima.release()
```

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

Return Value
It returns nothing.

4.3.1.8 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 ARIMA model first.

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

4.4 SEE ALSO

- **Introduction to FrovedisDvector**
- **DataFrame - Introduction**

Chapter 5

Bernoulli Naive Bayes

5.1 NAME

BernoulliNB - One of the variations of Naive Bayes algorithm. It is a classification algorithm to predict only binary output.

5.2 SYNOPSIS

```
frovedis.mllib.naive_bayes.BernoulliNB(alpha=1.0, fit_prior=True,
                                       class_prior=None, binarize=0.0,
                                       verbose=0)
```

5.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
predict_proba(X)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
debug_print()
load(fname, dtype = None)
save(fname)
release()
is_fitted()

5.3 DESCRIPTION

Naive Bayes classifier for bernoulli models.

The Bernoulli Naive Bayes classifier is suitable for classification with binary/boolean features.

This model is popular for document classification tasks, where binary term occurrence features (i.e a word occurs in a document or not) are used rather than finding the frequency of a word in document.

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 BernoulliNB 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 BernoulliNB 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.

5.3.1 Detailed Description

5.3.1.1 1. BernoulliNB()

Parameters
alpha: A positive double (float64) smoothing parameter (0 for no smoothing). It must be greater than or equal to 1. (Default: 1.0)
fit_prior: A boolean parameter specifying whether to learn class prior probabilities or not. If False, a uniform prior will be used. (Default: True)
class_prior: A numpy ndarray of double (float64) type values and must be of shape **(n_classes,)**. It gives prior probabilities of the classes. (Default: None)
When it is None (not specified explicitly), the priors are adjusted according to the data.
binarize: A double (float64) parameter specifying the threshold for binarizing sample features. (Default: 0.0)
verbose: An integer parameter specifying the log level to use. Its value is 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.

Attributes
class_log_prior_: A python ndarray of double (float64) type values and has shape **(n_classes,)**. It contains log probability of each class (smoothed).
feature_log_prob_: A python ndarray of double (float64) type values and has shape **(n_classes, n_features)**. It contains empirical log probability of features given a class, P(x_i|y).
class_count_: A python ndarray of double (float64) type values and has shape **(n_classes,)**. It contains the number of samples encountered for each class during fitting. This value is weighted by the sample weight when provided.
classes_: A python ndarray of double (float64) type values and has shape **(n_classes,)**. It contains the of unique labels given to the classifier during training.
feature_count_: A python ndarray of double (float64) type values and has shape **(n_classes, n_features)**. It contains the number of samples encountered for each (class, feature) during fitting. This value is weighted by the sample weight when provided.
coef_: A python ndarray of double (float64) type values.
- If 'classess_' is 2, then its shape **(1, n_features)**.
- If 'classess_' is more than 2, then its shape is **(n_classes, n_features)**.

It mirrors 'feature_log_prob_' for interpreting BernoulliNB as a linear model.
intercept_: A python ndarray of double (float64) type values.
- If 'classes_' is 2, the its shape **(1,)**.
- If 'classes_' is more than 2, then its shape is **(n_classes,)**.

It mirrors 'class_log_prior_' for interpreting BernoulliNB as a linear model.

Purpose
It initializes a BernoulliNB object with the given parameters.

Return Value
It simply returns "self" reference.

5.3.1.2 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 FrovedisRowmajorMatrix for dense data.
y: Any python array-like object or an instance of FrovedisDvector. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

Purpose
It accepts the training matrix (X) with labels (y) and trains a BernoulliNB model.

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 BernoulliNB object
from frovedis.mllib.linear_model import BernoulliNB
bnb = BernoulliNB(alpha = 1.0).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, especially 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 FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisRowmajorMatrix(mat)
dlbl = FrovedisDvector(lbl)

# BernoulliNB with pre-constructed frovedis-like inputs
from frovedis.mllib.linear_model import BernoulliNB
bnb = BernoulliNB(alpha = 1.0).fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

5.3.1.3 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 BernoulliNB model
bnb.predict(mat)
```

Output

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

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# predicting on BernoulliNB model using pre-constructed input
bnb.predict(rmat)
```

Output

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

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

5.3.1.4 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. Unlike Scikit-learn, it performs the classification on an array and returns the probability estimates for the test feature matrix (X).

For example,

```
# finds the probability sample for each class in the model
bnb.predict_proba(mat)
```

Output

```
 [[0.35939685 0.64060315]
  [0.35939685 0.64060315]
  [0.35939685 0.64060315]
  ...
```

```
  [0.35939685 0.64060315]
  [0.35939685 0.64060315]
  [0.35939685 0.64060315]]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# finds the probability sample for each class in the model
bnb.predict_proba(rmat)
```

Output

```
[[0.35939685 0.64060315]
 [0.35939685 0.64060315]
 [0.35939685 0.64060315]
 ...
 [0.35939685 0.64060315]
 [0.35939685 0.64060315]
 [0.35939685 0.64060315]]
```

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.

5.3.1.5 5. score(X, y)

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 containing true labels for X. It has shape **(n_samples,)**.

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
bnb.score(mat,lbl)
```

Output

```
0.6274
```

Return Value
It returns an accuracy score of float type.

5.3.1.6 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,

```
bnb.debug_print()
```

Output

```
model_type: bernoulli
binarize: -1
feature_count: 212 212 212 212 212  ... 357 357 357 357 357
theta: node = 0, local_num_row = 2, local_num_col = 30, val = -0.00468385 -0.00468385
-0.00468385 -0.00468385 -0.00468385 ... -0.0027894 -0.0027894 -0.0027894 -0.0027894 -0.0027894
pi: -0.987294 -0.466145
label: 0 1
class count: 212 357
theta_minus_negtheta: node = 0, local_num_row = 2, local_num_col = 30, val = 5.36129
5.36129 5.36129 5.36129 5.36129 ... 5.88053 5.88053 5.88053 5.88053 5.88053
negtheta_sum: -160.979 -176.5
```

This output will be visible on server side. It displays the target model information like model_type, binarize, feature_count, theta, pi, etc. values on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

5.3.1.7 7. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by BernoulliNB. It is used to get parameters and their values of BernoulliNB class.

For example,

```
print(bnb.get_params())
```

Output

```
{'alpha': 1.0, 'binarize': -1.0, 'class_prior': None, 'fit_prior': True, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

5.3.1.8 8. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by BernoulliNB, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(bnb.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
bnb.set_params(n_clusters = 4)
print("get parameters after setting:")
print(bnb.get_params())
```

Output

```
get parameters before setting:
{'alpha': 1.0, 'binarize': -1.0, 'class_prior': None, 'fit_prior': True, 'verbose': 0}
get parameters after setting:
{'alpha': 1.0, 'binarize': 0.5, 'class_prior': None, 'fit_prior': True, 'verbose': 0}
```

Return Value
It simply returns "self" reference.

5.3.1.9 9. load(fname, dtype=None)

Parameters
fname: A string object containing the name of the file having model information such as theta, cls_count, feature_count, label, pi, type 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,

```
bnb.load("./out/BNBModel")
```

Return Value
It simply returns "self" reference.

5.3.1.10 10. 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 (label_map, metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the bernoulli naive bayes model
bnb.save("./out/BNBModel")
```

This will save the naive bayes model on the path '/out/BNBModel'. It would raise exception if the directory already exists with same name.

The 'BNBModel' directory has

BNBModel
|—-label_map
|—-metadata
|—-**model**

|——cls_count
|——feature_count
|——label
|——pi
|——**theta**
|——type

'label_map' contains information about labels mapped with their encoded value.
The metadata file contains the model kind, input datatype used for trained model.
Here, the **model** directory contains information about class count, feature count, labels, pi, **theta** and thier datatype.

Return Value
It returns nothing

5.3.1.11 11. release()

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

For example,

```
bnb.release()
```

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

Return Value
It returns nothing.

5.3.1.12 12. 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 naive bayes model first.

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

5.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Multinomial Naive Bayes in Frovedis**

Chapter 6

DBSCAN

6.1 NAME

DBSCAN clustering - Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm commonly used in EDA (exploratory data analysis). It is a base algorithm for density-based clustering, which can discover clusters of different shapes and sizes from a large amount of data, containing noise and outliers.

6.2 SYNOPSIS

```
class frovedis.mllib.cluster.DBSCAN(eps=0.5, min_samples=5, metric='euclidean',
                                    metric_params=None, algorithm='auto',
                                    leaf_size=30, p=None, n_jobs=None,
                                    batch_fraction=None, verbose=0)
```

6.2.1 Public Member Functions

fit(X, y = None, sample_weight = None)
fit_predict(X, sample_weight = None)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
release()
is_fitted()

6.3 DESCRIPTION

DBSCAN is a Density-Based Clustering unsupervised learning algorithm that identifies distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point-density, separated from other such clusters by contiguous regions of low point-density.

It is able to find arbitrary shaped clusters and clusters with noise (i.e. outliers). The main idea behind DBSCAN is that a point belongs to a cluster if it is close to many points from that cluster.
DBSCAN algorithm requires two key parameters: **eps** and **min_samples**.

Based on these two parameters, the points are classified as core point, border point, or outlier:
- **Core point**: A point is a core point if there are at least minPts number of points (including the point itself) in its surrounding area with radius eps.
- **Border point**: A point is a border point if it is reachable from a core point and there are less than minPts number of points within its surrounding area.
- **Outlier**: A point is an outlier if it is not a core point and not reachable from any core points.

The main advantage of using DBSCAN is that it is able to find arbitrarily size and arbitrarily shaped clusters and identify noise data while clustering.

This module provides a client-server implementation, where the client application is a normal python program. Frovedis is almost same as Scikit-learn clustering module providing DBSCAN 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 DBSCAN 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, the 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.

6.3.1 Detailed Description

6.3.1.1 1. DBSCAN()

Parameters

eps: A positive double (float64) parameter containing the epsilon value or distance that specifies the neighborhoods. (Default: 0.5)
Two points are considered to be neighbors if the distance between them is less than or equal to eps.
min_samples: A positive integer parameter which specifies the number of samples in a neighborhood for a point to be considered as a core point.
This includes the point itself. (Default: 5)
metric: A string object parameter used when calculating distance between instances in a feature array. (Default: 'euclidean')
It only supports 'euclidean' distance.
metric_params: It is unused parameter which is an additional keyword arguments for the metric function. (Default: None)
algorithm: A string object parameter, used to compute pointwise distances and find nearest neighbors. (Default: 'auto')
When it is 'auto', it will be set as 'brute'(brute-force search approach). Unlike Scikit-learn, currently it supports only 'brute'.
leaf_size: An unused parameter, which is used to affect the speed of the construction and query. (Default: 30)
p: An unused parameter specifying the power of the Minkowski metric to be used to calculate distance between points. (Default: None)
n_jobs: An unused parameter specifying the number of parallel jobs to run. (Default: None)
batch_fraction: A positive double (float64) parameter used to calculate the batches of specific size. These batches are used to construct the distance matrix. (Default: None)
It must be within the range of 0.0 to 1.0. When it is None (not specified explicitly), it will be set as np.finfo(np.float64).max value.
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.

Attributes
labels_: It is a python ndarray, containing int64 typed values and has shape **(n_samples,)**. These are the coordinates of cluster centers.
core_sample_indices_: It is a python ndarray, containing int32 or int64 typed values and has shape **(n_samples,)**. These are the core samples indices.
components_: It is a numpy array or FrovedisRowmajorMatrix, containing float or double (float64) typed values and has shape **(n_samples, n_features)**.
These are the copy of each core sample found by training.

Purpose
It initializes a DBSCAN object with the given parameters.

The parameters: "metric_params", "leaf_size", "p" and "n_jobs" are simply kept to make the interface uniform to Scikit-learn DBSCAN clustering module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

6.3.1.2 2. fit(X, y = None, 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample.

Purpose

It clusters the given data points (X).

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# fitting input matrix on DBSCAN object
from frovedis.mllib.cluster import DBSCAN
dbm = DBSCAN(eps = 5, min_samples = 2).fit(train_mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
```

```
rmat = FrovedisRowmajorMatrix(train_mat)

# DBSCAN with pre-constructed Frovedis-like inputs
from frovedis.mllib.cluster import DBSCAN
dbm = DBSCAN(eps = 5, min_samples = 2).fit(rmat)
```

Return Value
It simply returns "self" reference.

6.3.1.3 3. fit_predict(X, 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
sample_weight: Python array-like containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample.

Purpose

It computes the clusters from the given data points (X) or distance matrix and predict labels.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# fitting input matrix on DBSCAN object
from frovedis.mllib.cluster import DBSCAN
dbm = DBSCAN(eps = 5, min_samples = 2)
print(dbm.fit_predict(train_mat))
```

Output

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

Like in fit() frovedis-like input can be used to speed-up training at server side.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# using pre-constructed input matrix
from frovedis.mllib.cluster import DBSCAN
dbm = DBSCAN(eps = 5, min_samples = 2)
print(dbm.fit_predict(rmat))
```

Output

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

Return Value
It returns a numpy array of int64 type containing the cluster labels, with shape **(n_samples,)**.

6.3.1.4 4. score(X, y, sample_weight = None)

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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: A python ndarray or an instance of FrovedisVector containing the true labels for X. It has shape **(n_samples,)**.
sample_weight: An unused parameter whose default value is None. It is simply ignored in frovedis implementation, like in Scikit-learn as well.

Purpose
It calculates homogeneity score on given test data and labels i.e homogeneity score of self.predict(X, y) wrt. y.

For example,

```
dbm.score(train_mat, [0 0 1 1 1])
```

Output

```
1.0
```

Return Value
It returns a score of float type.

6.3.1.5 5. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by DBSCAN. It is used to get parameters and their values of DBSCAN class.

For example,

```
print(dbm.get_params())
```

Output

```
{'algorithm': 'auto', 'batch_fraction': None, 'eps': 5, 'leaf_size': 30,
'metric': 'euclidean', 'metric_params': None, 'min_samples': 2, 'n_jobs': None,
'p': None, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

6.3.1.6 6. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by DBSCAN, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(kmeans.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
kmeans.set_params(n_clusters=4, n_init=5)
print("Get parameters after setting:")
print(kmeans.get_params())
```

Output

```
Get parameters before setting:
{'algorithm': 'auto', 'copy_x': True, 'init':'random', 'max_iter': 300,
'n_clusters': 2, 'n_init': 1, 'n_jobs': 1, 'precompute_distances': 'auto',
'random_state': None, 'tol': 0.0001, 'use_shrink': False, 'verbose': 0}
Get parameters after setting:
{'algorithm': 'auto', 'copy_x': True, 'init': 'random', 'max_iter': 300,
'n_clusters': 4, 'n_init': 5, 'n_jobs': 1, 'precompute_distances': 'auto',
'random_state': None, 'tol': 0.0001, 'use_shrink': False, 'verbose': 0}
```

Return Value
It simply returns "self" reference.

6.3.1.7 7. release()

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

For example,

```
dbm.release()
```

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

Return Value
It returns nothing.

6.3.1.8 8. is_fitted()

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

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

6.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Agglomerative Clustering in Frovedis**
- **Spectral Clustering in Frovedis**

- **KMeans in Frovedis**

Chapter 7

DecisionTreeClassifier

7.1 NAME

DecisionTreeClassifier - A classification algorithm that predicts the binary and multi-class output using conditional control statements. A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance, event, outcomes, resource costs, and utility.

7.2 SYNOPSIS

```
class frovedis.mllib.tree.DecisionTreeClassifier(criterion='gini', splitter='best',
                                                 max_depth=None, min_samples_split=2,
                                                 min_samples_leaf=1,
                                                 min_weight_fraction_leaf=0.0,
                                                 max_features=None,
                                                 random_state=None, max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 class_weight=None, presort='deprecated',
                                                 ccp_alpha=0.0, max_bins=32,
                                                 verbose=0, categorical_info={})
```

7.2.1 Public Member Functions

fit(X, y)
predict(X)
predict_proba(X)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
score(X, y, sample_weight = None)
debug_print()
release()
is_fitted()

7.3 DESCRIPTION

Decision Tree Classifier is a supervised learning method used for classification problems. The aim is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Like any other tree representation, it has a root node, internal nodes, and leaf nodes. The internal node represents condition on attributes, the branches represent the results of the condition and the leaf node represents the class label. **Frovedis supports both binary and multinomial decision tree classification algorithms.**

During training, the input **X** is the training data and **y** is the corresponding label values (Frovedis supports any values for labels, but internally it encodes the input binary labels to 0 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.

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 DecisionTreeClassifier 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 DecisionTreeClassifier 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.

7.3.1 Detailed Description

7.3.1.1 1. DecisionTreeClassifier()

Parameters
criterion: A string object parameter that specifies the function to measure the quality of a split. Supported criteria are 'gini' and 'entropy'. (Default: 'gini')
- **'gini' impurity**: calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly.
- **'entropy' (information gain)**: it is applied to quantify which feature provides maximal information about the classification based on the notion of entropy.

splitter: An unused parameter. (Default: 'best')
max_depth: A positive integer parameter that specifies the maximum depth of the tree. (Default: None)
If it is None (not specified explicitly), then 'max_depth' is set to 5.
min_samples_split: An unused parameter. (Default: 2)
min_samples_leaf: A positive integer or float value that specifies the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered, if it leaves, at least 'min_samples_leaf' training samples in each of the left and right branches. (Default: 1)
- If it is an integer, then 'min_samples_leaf' should be greater than or equal to 1.
- If it is float, then 'min_samples_leaf' should be in range **(0,0.5]**.

min_weight_fraction_leaf: An unused parameter. (Default: 0.0)
max_features: An unused parameter. (Default: None)
random_state: An unused parameter. (Default: None)
max_leaf_nodes: An unused parameter. (Default: None)
min_impurity_decrease: A positive double (float64) parameter. A node will be split if this split induces

a decrease of the impurity greater than or equal to this value. (Default: 0.0)
class_weight: An unused parameter. (Default: None)
presort: An unused parameter. (Default: 'deprecated')
ccp_alpha: An unused parameter. (Default: 0.0)
max_bins: A positive integer parameter that specifies the maximum number of bins created by ordered splits. (Default: 32)
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.
categorical_info: A dictionary that specifies categorical features information. Here, it gives column indices of categorical features and the number of categories for those features.
For example, a dictionary { {0, 2}, {4, 5} } means that the feature[0] takes values 0 or 1 (binary) and the feature[4] has five categories (values 0, 1, 2, 3 or 4). Note that feature indices and category assignments are 0-based. (Default: {})

Attributes
classes_: It is a python ndarray (any type) of unique labels given to the classifier during training. It has shape **(n_classes,)**.
n_features_: An integer value specifying the number of features when fitting the estimator.

Purpose
It initializes a DecisionTreeClassifier object with the given parameters.

The parameters: "splitter", "min_samples_split", "min_weight_fraction_leaf", "max_features", "random_state", "max_leaf_nodes", "class_weight", "presort" and "ccp_alpha" are simply kept in to make the interface uniform to the Scikit-learn DecisionTreeClassifier module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

7.3.1.2 2. fit(X, y)

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 containing the class labels. It has shape **(n_samples,)**.

Purpose
It builds a decision tree classifier from the training data X and labels y.

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 DecisionTreeClassifier object
from frovedis.mllib.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(max_depth = 5)
dtc.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, especially 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)

# DecisionTreeClassifier with pre-constructed frovedis-like inputs
from frovedis.mllib.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(max_depth = 5)
dtc.fit(cmat,dlbl)
```

Return Value
It simply returns "self" reference.

7.3.1.3 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
Predict class for X.

For a classification model, the predicted class value for each sample in X is returned.

For example,

```
# predicting on decision tree classifier model
dtc.predict(mat)
```

Output

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

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# predicting on decision tree classifier model using pre-constructed input
dtc.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 classes.

7.3.1.4 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. It has shape **(n_samples, n_features)**.

Purpose
Predict class log-probabilities of the input samples X.

Currently, this method is not supported for multinomial classification problems.

For example,

```
# finds the probability sample for each class in the model
dtc.predict_proba(mat)
```

Output

```
[[1.       0.      ]
 [1.       0.      ]
 [1.       0.      ]
 ...
 [1.       0.      ]
 [1.       0.      ]
 [0.009375 0.990625]]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# finds the probability sample for each class in the model
dtc.predict_proba(cmat.to_frovedis_rowmatrix())
```

Output

```
[[1.       0.      ]
 [1.       0.      ]
 [1.       0.      ]
 ...
 [1.       0.      ]
 [1.       0.      ]
 [0.009375 0.990625]]
```

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. Since only binary classification problems (**n_classes = 2**) are supported for this method, so currenlty its shape will be **(n_samples, 2)**.

7.3.1.5 5. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by DecisionTreeClassifier. It is used to get parameters and their values of DecisionTreeClassifier class.

For example,

```
print(dtc.get_params())
```

Output

```
{'categorical_info': {}, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'GINI',
'max_bins': 32, 'max_depth': 5, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': None,
'splitter': 'best', 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

7.3.1.6 6. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by DecisionTreeClassifier, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(dtc.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
dtc.set_params(criterion = 'entropy', max_depth = 5)
print("get parameters after setting:")
print(dtc.get_params())
```

Output

```
get parameters before setting:
{'categorical_info': {}, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'GINI',
'max_bins': 32, 'max_depth': 5, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': None,
'splitter': 'best', 'verbose': 0}
get parameters after setting:
{'categorical_info': {}, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy',
'max_bins': 32, 'max_depth': 5, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': None,
'splitter': 'best', 'verbose': 0}
```

Return Value
It simply returns “self” reference.

7.3.1.7 7. 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,

```
dtc.load("./out/MyDecisionTreeClassifierModel")
```

Return Value
It simply returns "self" reference.

7.3.1.8 8. 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 (label_map, metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the decision tree classifier model
dtc.save("./out/MyDecisionTreeClassifierModel")
```

This will save the decision tree classifier model on the path "/out/MyDecisionTreeClassifierModel". It would raise exception if the directory already exists with same name.

The 'MyDecisionTreeClassifierModel' directory has

MyDecisionTreeClassifierModel
|—label_map
|—metadata
|—model

'label_map' file contains information about labels mapped with their encoded value.
The 'metadata' file contains the number of classes, model kind and input datatype used for trained model.
The 'model' file contains the decision tree model saved in binary format.

Return Value
It returns nothing.

7.3.1.9 9. 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 containing the true labels for X. It has shape **(n_samples,)**.
sample_weight: Python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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
dtc.score(mat,lbl)
```

Output

```
0.9895
```

Return Value
It returns an accuracy score of float type.

7.3.1.10 10. debug_print()

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

For example,

```
dtc.debug_print()
```

Output

```
    -------- Classification Tree:: --------
# of nodes: 35, height: 5
<1> Split: feature[22] < 113.157, IG: 0.319406
 \_ <2> Split: feature[27] < 0.144844, IG: 0.0626618
 |  \_ <4> Split: feature[22] < 107.223, IG: 0.00955662
 |  |  \_ <8> Split: feature[10] < 0.904737, IG: 0.00582386
 |  |  |  \_ <16> Split: feature[27] < 0.133781, IG: 0.00558482
 |  |  |  |  \_ (32) Predict: 1 (99.0625%)
 |  |  |  |  \_ (33) Predict: 1 (66.6667%)
 |  |  |  \_ (17) Predict: 0 (100%)
 |  |  \_ <9> Split: feature[14] < 0.00570775, IG: 0.184704
 |  |     \_ <18> Split: feature[0] < 14.0963, IG: 0.165289
 |  |     |  \_ (36) Predict: 0 (100%)
 |  |     |  \_ (37) Predict: 1 (100%)
 |  |     \_ <19> Split: feature[21] < 21.9025, IG: 0.408163
 |  |        \_ (38) Predict: 1 (100%)
 |  |        \_ (39) Predict: 0 (100%)
 |  \_ <5> Split: feature[21] < 24.13, IG: 0.189372
 |     \_ <10> Split: feature[4] < 0.109085, IG: 0.32
 |     |  \_ (20) Predict: 1 (100%)
 |     |  \_ (21) Predict: 0 (100%)
 |     \_ <11> Split: feature[9] < 0.0609525, IG: 0.144027
 |        \_ (22) Predict: 1 (100%)
 |        \_ <23> Split: feature[7] < 0.0404469, IG: 0.0907029
 |           \_ (46) Predict: 1 (100%)
 |           \_ (47) Predict: 0 (100%)
 \_ <3> Split: feature[7] < 0.0509028, IG: 0.0364914
    \_ <6> Split: feature[17] < 0.00995862, IG: 0.259286
    |  \_ <12> Split: feature[1] < 15.855, IG: 0.336735
    |  |  \_ (24) Predict: 1 (100%)
    |  |  \_ (25) Predict: 0 (100%)
    |  \_ (13) Predict: 1 (100%)
    \_ <7> Split: feature[1] < 13.9925, IG: 0.0116213
       \_ <14> Split: feature[1] < 12.18, IG: 0.5
       |  \_ (28) Predict: 0 (100%)
```

```
|  \_ (29) Predict: 1 (100%)
\_ (15) Predict: 0 (100%)
```

This output will be visible on server side. It displays the decision tree having maximum depth of 5 and total 35 nodes in the tree.

No such output will be visible on client side.

Return Value
It returns nothing.

7.3.1.11 11. release()

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

For example,

```
dtc.release()
```

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

Return Value
It returns nothing.

7.3.1.12 12. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

7.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Decision Tree Regressor in Frovedis**

Chapter 8

DecisionTreeRegressor

8.1 NAME

DecisionTreeRegressor - A regression algorithm used to predict the output using conditional control statements. A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance, event, outcomes, resource costs, and utility.

8.2 SYNOPSIS

```
class frovedis.mllib.tree.DecisionTreeRegressor(criterion='mse', splitter='best',
                                                max_depth=None, min_samples_split=2,
                                                min_samples_leaf=1,
                                                min_weight_fraction_leaf=0.0,
                                                max_features=None, random_state=None,
                                                max_leaf_nodes=None,
                                                min_impurity_decrease=0.0,
                                                presort='deprecated', ccp_alpha=0.0,
                                                max_bins=32, verbose=0,
                                                categorical_info={})
```

8.2.1 Public Member Functions

fit(X, y)
predict(X)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
score(X, y, sample_weight = None)
debug_print()
release()
is_fitted()

8.3 DESCRIPTION

Decision Tree Regressor is a supervised learning method used for regression problems. The aim is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Like any other tree representation, it has a root node, internal nodes, and leaf nodes. The internal node represents condition on attributes, the branches represent the results of the condition and the leaf node represents the class label.

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 DecisionTreeRegressor 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 DecisionTreeRegressor 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.

8.3.1 Detailed Description

8.3.1.1 1. DecisionTreeRegressor()

Parameters
criterion: A string object parameter that specifies the function to measure the quality of a split.
Currently, supported criteria is 'mse'. The mean squared error (mse), is equal to variance reduction as a feature selection criterion and minimizes the L2 loss using the mean of each terminal node. (Default: 'mse')
splitter: An unused parameter. (Default: 'best')
max_depth: A positive integer parameter that specifies the maximum depth of the tree. (Default: None)
When it is None (not specified explicitly), then 'max_depth' is set to 5.
min_samples_split: An unused parameter. (Default: 2)
min_samples_leaf: A positive integer or float value that specifies the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least 'min_samples_leaf' training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression. (Default: 1)
- If it is an integer, then 'min_samples_leaf' should be greater than or equal to 1.
- If it is float, then 'min_samples_leaf' should be in range **(0,0.5]**.

min_weight_fraction_leaf: An unused parameter. (Default: 0.0)
max_features: An unused parameter. (Default: None)
random_state: An unused parameter. (Default: None)
max_leaf_nodes: An unused parameter. (Default: None)
min_impurity_decrease: A positive float parameter. A node will be split if this split induces a decrease of the impurity greater than or equal to this value. (Default: 0.0)
presort: An unused parameter. (Default: 'deprecated')
ccp_alpha: An unused parameter. (Default: 0.0)
max_bins: A positive integer parameter that specifies the maximum number of bins created by ordered splits. (Default: 32)
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.

categorical_info: A dictionary that specifies categorical features information. Here, it gives column indices of categorical features and the number of categories for those features.
For example, a dictionary { {0, 2}, {4, 5} } means that the feature[0] takes values 0 or 1 (binary) and the feature[4] has five categories (values 0, 1, 2, 3 or 4). Note that feature indices and category assignments are 0-based. (Default: {})

Attributes
n_features_: An integer value specifying the number of features when fitting the estimator.

Purpose
It initializes a DecisionTreeRegressor object with the given parameters.

The parameters: "splitter", "min_samples_split", "min_weight_fraction_leaf", "max_features", "random_state", "max_leaf_nodes", "presort" and "ccp_alpha" are simply kept in to make the interface uniform to the Scikit-learn DecisionTreeRegressor module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

8.3.1.2 2. fit(X, y)

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 containing the target values for X. It has shape **(n_samples,)**.

Purpose
It builds a decision tree regressor from the training data X and labels y.

For example,

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

# fitting input matrix and label on DecisionTreeRegressor object
from frovedis.mllib.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(max_depth = 5)
dtr.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, especially 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, lbl = 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)
```

```
# DecisionTreeRegressor with pre-constructed frovedis-like inputs
from frovedis.mllib.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(max_depth = 5)
dtr.fit(cmat,dlbl)
```

Return Value
It simply returns "self" reference.

8.3.1.3 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
Predict regression value for X.

For a regression model, the predicted value based on X is returned.

For example,

```
# predicting on decision tree regressor model
dtr.predict(mat)
```

Output

```
[24.64166667 24.64166667 34.80666667 ... 27.83636364 27.83636364 20.98461538]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# predicting on decision tree regressor model using pre-constructed input
dtr.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[24.64166667 24.64166667 34.80666667 ... 27.83636364 27.83636364 20.98461538]
```

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

8.3.1.4 4. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by DecisionTreeRegressor. It is used to get parameters and their values of DecisionTreeRegressor class.

For example,

```
print(dtr.get_params())
```

Output

```
{'categorical_info': {}, 'ccp_alpha': 0.0, 'criterion': 'MSE', 'max_bins': 32,
'max_depth': 5, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'presort': 'deprecated', 'random_state': None, 'splitter': 'best', 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

8.3.1.5 5. set_params(**params)

Parameters
***params**: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by DecisionTreeRegressor, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(dtr.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
dtr.set_params(max_depth = 4)
print("get parameters after setting:")
print(dtr.get_params())
```

Output

```
get parameters before setting:
{'categorical_info': {}, 'ccp_alpha': 0.0, 'criterion': 'MSE', 'max_bins': 32,
'max_depth': 5, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'presort': 'deprecated', 'random_state': None, 'splitter': 'best', 'verbose': 0}
get parameters after setting:
{'categorical_info': {}, 'ccp_alpha': 0.0, 'criterion': 'MSE', 'max_bins': 32,
'max_depth': 4, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'presort': 'deprecated', 'random_state': None, 'splitter': 'best', 'verbose': 0}
```

Return Value
It simply returns “self” reference.

8.3.1.6 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,

```
dtr.load("./out/MyDecisionTreeRegressorModel")
```

Return Value
It simply returns "self" reference.

8.3.1.7 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the decision tree regressor model
dtr.save("./out/MyDecisionTreeRegressorModel")
```

This will save the decision tree regressor model on the path "/out/MyDecisionTreeRegressorModel". It would raise exception if the directory already exists with same name.

The 'MyDecisionTreeRegressorModel' directory has

MyDecisionTreeRegressorModel
|——metadata
|——model

The 'metadata' file contains the information on model kind and input datatype used for trained model.
The 'model' file contains the decision tree model saved in binary format.

Return Value
It returns nothing.

8.3.1.8 8. 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 containing the true values for X. It has shape **(n_samples,)**.
sample_weight: Python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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
dtr.score(mat,lbl)
```

Output

```
0.9109
```

Return Value
It returns an R2 score of float type.

8.3.1.9 9. debug_print()

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

For example,

```
dtr.debug_print()
```

Output

```
-------- Regression Tree:: --------
# of nodes: 51, height: 5
<1> Split: feature[12] < 9.6575, IG: 37.2288
 \_ <2> Split: feature[5] < 7.14906, IG: 37.6417
 |  \_ <4> Split: feature[7] < 1.47325, IG: 17.8515
 |  |  \_ (8) Predict: 50
 |  |  \_ <9> Split: feature[5] < 6.69962, IG: 9.19518
 |  |     \_ <18> Split: feature[5] < 6.10819, IG: 2.41396
 |  |     |  \_ (36) Predict: 20.9846
 |  |     |  \_ (37) Predict: 24.6417
 |  |     \_ <19> Split: feature[12] < 5.27, IG: 6.57304
 |  |        \_ (38) Predict: 32.9652
 |  |        \_ (39) Predict: 27.8364
 |  \_ <5> Split: feature[5] < 7.45969, IG: 25.6196
 |     \_ <10> Split: feature[5] < 7.43531, IG: 4.71838
 |     |  \_ <20> Split: feature[8] < 7.125, IG: 1.42815
 |     |  |  \_ (40) Predict: 34.8067
 |     |  |  \_ (41) Predict: 31.6
 |     |  \_ (21) Predict: 44
 |     \_ <11> Split: feature[0] < 2.1766, IG: 19.28
 |        \_ <22> Split: feature[10] < 17.9563, IG: 5.68288
 |        |  \_ (44) Predict: 46.9375
 |        |  \_ (45) Predict: 40.125
 |        \_ (23) Predict: 21.9
 \_ <3> Split: feature[12] < 14.9763, IG: 9.38095
    \_ <6> Split: feature[5] < 6.56925, IG: 1.74534
    |  \_ <12> Split: feature[11] < 101.175, IG: 0.921292
    |  |  \_ <24> Split: feature[4] < 0.5995, IG: 6.02083
    |  |  |  \_ (48) Predict: 16.5667
    |  |  |  \_ (49) Predict: 10.9
    |  |  \_ <25> Split: feature[2] < 2.57187, IG: 0.586878
    |  |     \_ (50) Predict: 25.3667
    |  |     \_ (51) Predict: 20.4598
    |  \_ <13> Split: feature[0] < 4.94925, IG: 10.4783
    |     \_ <26> Split: feature[4] < 0.59225, IG: 7.70469
```

```
|     |  \_ (52) Predict: 24.4429
|     |  \_ (53) Predict: 30.5
|     \_ (27) Predict: 15
\_ <7> Split: feature[0] < 5.64819, IG: 5.7942
   \_ <14> Split: feature[4] < 0.52725, IG: 2.64736
   |  \_ <28> Split: feature[12] < 29.8562, IG: 2.20522
   |  |  \_ (56) Predict: 20.4
   |  |  \_ (57) Predict: 15.45
   |  \_ <29> Split: feature[12] < 18.0497, IG: 1.97256
   |     \_ (58) Predict: 17.5667
   |     \_ (59) Predict: 14.7563
   \_ <15> Split: feature[12] < 20.0091, IG: 3.36396
      \_ <30> Split: feature[5] < 5.92387, IG: 2.19683
      |  \_ (60) Predict: 11.4714
      |  \_ (61) Predict: 15.137
      \_ <31> Split: feature[4] < 0.67175, IG: 4.84222
         \_ (62) Predict: 13.88
         \_ (63) Predict: 9.15938
```

This output will be visible on server side. It displays the decision tree having maximum depth of 5 and total 51 nodes in the tree.

No such output will be visible on client side.

Return Value
It returns nothing.

8.3.1.10 10. release()

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

For example,

```
dtr.release()
```

This will remove the trained model, model-id present on server, along with releasing server side memory.

Return Value
It returns nothing.

8.3.1.11 11. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

8.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**

- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Decision Tree Classifier in Frovedis**

Chapter 9

FactorizationMachineClassifier

9.1 NAME

FactorizationMachineClassifier - A factorization machine is a general-purpose supervised learning algorithm that can be used for classification tasks. It is an extension of a linear model that is designed to capture interactions between features within high dimensional sparse datasets.

9.2 SYNOPSIS

```
class frovedis.mllib.fm.FactorizationMachineClassifier(iteration=100, init_stdev=0.1,
                                                       init_learn_rate=0.01, optimizer="SGD",
                                                       dim=(True,True, 8), reg=(0, 0, 0),
                                                       batch_size_pernode=100, verbose=0)
```

9.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
load(fname, dtype = None)
score(X, y, sample_weight = None)
save(fname) get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

9.3 DESCRIPTION

The FactorizationMachineClassifier (fmc) is a general predictor like SVMs but is also able to estimate reliable parameters under very high sparsity. The factorization machine models all nested variable interactions (comparable to a polynomial kernel in SVM), but uses a factorized parameterization instead of a dense parametrization like in SVMs. We show that the model equation of fmcs can be computed in linear time and that it depends only on a linear number of parameters. This allows direct optimization and storage of model

parameters without the need of storing any training data (e.g. support vectors) for prediction. **Frovedis supports both binary and multinomial labels.**

During training, the input **X** is the training data and **y** are their corresponding label values (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.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis interface is almost same as libFM. libFM is a software implementation for factorization machines that features stochastic gradient descent (SGD) and alternating least squares (ALS) optimization as well as Bayesian inference using Markov Chain Monte Carlo (MCMC). 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 FactorizationMachineClassifier 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 predict-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 1. FactorizationMachineClassifier()

Parameters
iteration: A positive integer parameter, specifying the maximum number of iteration count. (Default: 100)
init_stdev: A positive double parameter specifying the standard deviation which is used to initialize the model parameter of 2-way factors. (Default: 0.1)
init_learn_rate: A double parameter containing the learning rate for SGD optimizer. (Default: 0.01)
It should be in range from 0.00001 to 1.0.
optimizer: A string object parameter that specifies which algorithms minimize or maximize a Loss function E(x) using its gradient values with respect to the parameters. (Default: 'SGD')
Only 'SGD' is supported.
dim: A tuple that specifies three important parameters with default values- (True, True,8):
- ***global_bias***: A boolean value that represents a switch to use bias. Currently, this parameter is not used in Frovedis implementation.
- ***dim_one_interactions*** : A boolean value represents a switch to use 1-way interaction.
- ***dim_factors_no*** : A positive integer that represents the dimension of 2-way interaction or number of factors that are used for pairwise interactions.

When any of the three is None (not specified explicitly), then user experiences an Error.
reg: A tuple that specifies three important parameters with default values- (Default: (0, 0, 0))
- ***regularization_intercept*** : A positive integer that represents the regularization parameters of intercept or bias regularization.
- ***regularization_one_interactions*** : A positive integer that represents the switch to use 1-way regularization.
- ***regularization_factors_no*** : A positive integer that represents the dimension of 2-way interaction or number of factors that are used for pairwise regularization.

When any of the three is None (not specified explicitly), then user experiences an Error.
batch_size_pernode: A positive integer parameter specifies the size of minibatch processed by one node. (Default: 100)
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.

Attributes
classes_: A numpy array of long (int64) type value that specifies unique labels given to the classifier during training. It has shape **(n_classes,)**, where n_classes is the unique number of classes.

Purpose
It initializes a FactorizationMachineClassifier object with the given parameters.

Return Value
It simply returns "self" reference.

9.3.1.2 2. fit(X, y, sample_weight=None)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: Any python array-like object or an instance of FrovedisDvector containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample.

Purpose
It accepts the training matrix (X) with labels (y) and trains a FactorizationMachineClassifier model.

For example,

```
# loading sample data
row = np.array([0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5])
col = np.array([0, 1, 4, 0, 1, 2, 4, 1, 2, 3, 2,3,4,5, 0, 1, 3, 4, 3, 5])
data = np.asarray([ 2.0, 1.0,1.0, 1.0, 3.0,1.0, 1.0, 1.0, 2.0,1.0, 1.0, 3.0,
                    1.0,1.0,1.0, 1.0, 1.0, 3.0,1.0, 1.0])
csr = csr_matrix((data, (row, col)), shape = (6, 6))
lbl = [10.0, 10.0, 10.0, 20.0, 10.0, 20.0]

# fitting input data on FactorizationMachineClassifier object
from frovedis.mllib.fm import FactorizationMachineClassifier
fmc = FactorizationMachineClassifier()
fmc.fit(csr, 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, especially when same data would be used for multiple executions.

For example,

```
# loading sample data
row = np.array([0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5])
col = np.array([0, 1, 4, 0, 1, 2, 4, 1, 2, 3, 2,3,4,5, 0, 1, 3, 4, 3, 5])
data = np.asarray([ 2.0, 1.0,1.0, 1.0, 3.0,1.0, 1.0, 1.0, 2.0,1.0, 1.0, 3.0,
                    1.0,1.0,1.0, 1.0, 1.0, 3.0,1.0, 1.0])
csr = csr_matrix((data, (row, col)), shape = (6, 6))
lbl = [10.0, 10.0, 10.0, 20.0, 10.0, 20.0]

# Since "csr" is scipy sparse data, we have created FrovedisCRSMatrix.
from frovedis.matrix.crs import FrovedisCRSMatrix
```

```
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisCRSMatrix(csr)
dlbl = FrovedisDvector(lbl)

# fitting input data on FactorizationMachineClassifier object
from frovedis.mllib.fm import FactorizationMachineClassifier
fmc = FactorizationMachineClassifier()
fmc.fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

9.3.1.3 3. predict(X)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix of float or double (float64) type. 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,

```
# predicting on FactorizationMachineClassifier model
fmc.predict(csr)
```

Output

```
[20. 20. 20. 20. 20. 20.]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

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

# predicting on FactorizationMachineClassifier model using pre-constructed input
fmc.predict(cmat)
```

Output

```
[20. 20. 20. 20. 20. 20.]
```

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

9.3.1.4 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
Currently, this method is not supported for FactorizationMachineClassifier. It is simply kept in FactorizationMachineClassifier module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

9.3.1.5 5. score(X, y, sample_weight = None)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: Any python array-like object containing class labels. It is of shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), 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,

```
fmc.score(csr, lbl)
```

Output

```
0.83
```

Return Value
It returns an accuracy score of double (float64) type.

9.3.1.6 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(label_map, metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the FactorizationMachineClassifier model
fmc.save("./out/FMCModel")
```

The FMCModel contains below directory structure:
FMCModel
|——label_map
|——metadata
|——model

'label_map' contains information about labels mapped with their encoded value.
'metadata' represents the detail about model_kind and datatype of training vector.
Here, the model file contains information about trained model in binary format.

This will save the FactorizationMachineClassifier model on the path '/out/FMCModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

9.3.1.7 7. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by FactorizationMachineClassifier. It is used to get parameters and their values of FactorizationMachineClassifier class.

For example,

```
print(fmc.get_params())
```

Output

```
{'batch_size_pernode': 100, 'dim': (True, True, 8), 'init_learn_rate': 0.01,
'init_stdev': 0.1, 'iteration': 100, 'optimizer': 'SGD', 'reg': (0, 0, 0),
'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

9.3.1.8 8. set_params(**params)

Parameters
****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by FactorizationMachineClassifier, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(fmc.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
fmc.set_params(iteration = 200)
print("Get parameters after setting:")
print(fmc.get_params())
```

Output

```
Get parameters before setting:
{'batch_size_pernode': 100, 'dim': (True, True, 8), 'init_learn_rate': 0.01,
'init_stdev': 0.1, 'iteration': 100, 'optimizer': 'SGD', 'reg': (0, 0, 0),
'verbose': 0}
Get parameters after setting:
{'batch_size_pernode': 100, 'dim': (True, True, 8), 'init_learn_rate': 0.01,
'init_stdev': 0.1, 'iteration': 200, 'optimizer': 'SGD', 'reg': (0, 0, 0),
'verbose': 0}
```

Return Value
It simply returns “self” reference.

9.3.1.9 9. debug_print()

Purpose
Currently, this method is not supported for FactorizationMachineClassifier. It is simply kept in FactorizationMachineClassifier module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

9.3.1.10 10. release()

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

For example,

```
fmc.release()
```

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

Return Value
It returns nothing.

9.3.1.11 11. 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'.

9.4 SEE ALSO

- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Factorization Machine Regressor in Frovedis**

Chapter 10

FactorizationMachineRegressor

10.1 NAME

FactorizationMachineRegressor - A factorization machine is a general-purpose supervised learning algorithm that can be used for regression tasks. It is an extension of a linear model that is designed to capture interactions between features within high dimensional sparse datasets.

10.2 SYNOPSIS

```
class frovedis.mllib.fm.FactorizationMachineRegressor(iteration=100, init_stdev=0.1,
                                                      init_learn_rate=0.01, optimizer="SGD",
                                                      dim=(True,True, 8), reg=(0, 0, 0),
                                                      batch_size_pernode=100, verbose=0)
```

10.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
load(fname, dtype = None)
save(fname)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

10.3 DESCRIPTION

The FactorizationMachineRegressor (fmr) is a general predictor like SVMs but is also able to estimate reliable parameters under very high sparsity. The factorization machine models all nested variable interactions (comparable to a polynomial kernel in SVM), but uses a factorized parameterization instead of a dense parametrization like in SVMs. We show that the model equation of fmrs can be computed in linear time and

that it depends only on a linear number of parameters. This allows direct optimization and storage of model parameters without the need of storing any training data (e.g. support vectors) for prediction.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis interface is almost same as libFM. libFM is a software implementation for factorization machines that features stochastic gradient descent (SGD) and alternating least squares (ALS) optimization as well as Bayesian inference using Markov Chain Monte Carlo (MCMC). 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 FactorizationMachineRegressor 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 predict-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.

10.3.1 Detailed Description

10.3.1.1 1. FactorizationMachineRegressor()

Parameters
iteration: A positive integer parameter, specifying the maximum number of iteration count. (Default: 100)
init_stdev: A positive double parameter specifying the standard deviation which is used to initialize the model parameter of 2-way factors (Default: 0.1)
init_learn_rate: A double parameter containing the learning rate for SGD optimizer. (Default: 0.01)
It should be in range from 0.00001 to 1.0.
optimizer: A string object parameter that specifies which algorithms minimize or maximize a Loss function E(x) using its Gradient values with respect to the parameters. (Default: 'SGD')
Only 'SGD' is supported.
dim: A tuple that specifies three important parameters with default values- (True, True,8):
- ***global_bias***: A boolean value that represents a switch to use bias. Currently, this parameter is not used in Frovedis implementation.
- ***dim_one_interactions*** : A boolean value that represents a switch to use 1-way interaction.
- ***dim_factors_no*** : A positive integer that represents the dimension of 2-way interaction or number of factors that are used for pairwise interactions.

When any of the three is None (not specified explicitly), then user experiences an Error.
reg: An tuple of values that specifies three important parameters with default values- (Default: (0, 0, 0))
- ***regularization_intercept*** : A positive integer that represents the regularization parameters of intercept or bias regularization.
- ***regularization_one_interactions*** : A positive integer that represents the switch to use 1-way regularization.
- ***regularization_factors_no*** : A positive integer that represents the dimension of 2-way interaction or number of factors that are used for pairwise regularization.

When any of the three is None (not specified explicitly), then user experiences an Error.
batch_size_pernode: A positive integer parameter specifies the size of minibatch processed by one node. (Default: 100)
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.

Purpose
It initializes a FactorizationMachineRegressor object with the given parameters.
Return Value
It simply returns "self" reference.

10.3.1.2 2. fit(X, y, sample_weight=None)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: Any python array-like object or an instance of FrovedisDvector containing the target values. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample.

Purpose
It accepts the training matrix (X) with labels (y) and trains a FactorizationMachineRegressor model.

For example,

```
# loading sample data
row = np.array([0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5])
col = np.array([0, 1, 4, 0, 1, 2, 4, 1, 2, 3, 2,3,4,5, 0, 1, 3, 4, 3, 5])
data = np.asarray([ 2.0, 1.0,1.0, 1.0, 3.0,1.0, 1.0, 1.0, 2.0,1.0, 1.0, 3.0,
                    1.0,1.0,1.0, 1.0, 1.0, 3.0,1.0, 1.0])
csr = csr_matrix((data, (row, col)), shape = (6, 6))
lbl = [10.0, 10.0, 10.0, 20.0, 10.0, 20.0]

# fitting input data on FactorizationMachineRegressor object
from frovedis.mllib.fm import FactorizationMachineRegressor
fmr = FactorizationMachineRegressor()
fmr.fit(csr, 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, especially when same data would be used for multiple executions.

For example,

```
# loading sample data
row = np.array([0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5])
col = np.array([0, 1, 4, 0, 1, 2, 4, 1, 2, 3, 2,3,4,5, 0, 1, 3, 4, 3, 5])
data = np.asarray([ 2.0, 1.0,1.0, 1.0, 3.0,1.0, 1.0, 1.0, 2.0,1.0, 1.0, 3.0,
                    1.0,1.0,1.0, 1.0, 1.0, 3.0,1.0, 1.0])
csr = csr_matrix((data, (row, col)), shape = (6, 6))
lbl = [10.0, 10.0, 10.0, 20.0, 10.0, 20.0]

# Since "csr" is scipy sparse data, we have created FrovedisCRSMatrix.
from frovedis.matrix.crs import FrovedisCRSMatrix
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisCRSMatrix(csr)
dlbl = FrovedisDvector(lbl)

# fitting input data on FactorizationMachineRegressor object
from frovedis.mllib.fm import FactorizationMachineRegressor
```

```
fmr = FactorizationMachineRegressor()
fmr.fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

10.3.1.3 3. predict(X)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix of float or double (float64) type. 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,

```
# predicting on FactorizationMachineRegressor model
fmr.predict(csr)
```

Output

```
[10.00277618 20.6134937  13.81870647 26.75876098 33.13530746  3.03828379]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

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

# predicting on FactorizationMachineRegressor model using pre-constructed input
fmr.predict(cmat)
```

Output

```
[10.00277618 20.6134937  13.81870647 26.75876098 33.13530746  3.03828379]
```

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

10.3.1.4 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
Currently, this method is not supported for FactorizationMachineRegressor. It is simply kept in FactorizationMachineRegressor module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

10.3.1.5 5. score(X, y, sample_weight = None)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: Any python array-like object containing the target values. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), 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,

```
fmr.score(csr, lbl)
```

Output

```
-0.10
```

Return Value
It returns an accuracy score of double (float64) type.

10.3.1.6 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(metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the FactorizationMachineRegressor model
fmr.save("./out/FMRModel")
```

The FMRModel contains below directory structure:
FMRModel
|——metadata
|——model

'metadata' represents the detail about model_kind and datatype of training vector.
Here, the model file contains information about trained model in binary format.

This will save the FactorizationMachineRegressor model on the path '/out/FMRModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

10.3.1.7 7. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by FactorizationMachineRegressor. It is used to get parameters and their values of FactorizationMachineRegressor class.

For example,

```
print(fmr.get_params())
```

Output

```
{'batch_size_pernode': 100, 'dim': (True, True, 8), 'init_learn_rate': 0.01,
'init_stdev': 0.1, 'iteration': 100, 'optimizer': 'SGD', 'reg': (False, False, 0),
'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

10.3.1.8 8. set_params(**params)

Parameters
****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by FactorizationMachineRegressor, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(fmr.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
fmr.set_params(iteration=200)
print("Get parameters after setting:")
print(fmr.get_params())
```

Output

```
Get parameters before setting: {'batch_size_pernode': 100, 'dim': (True, True, 8),
'init_learn_rate': 0.01, 'init_stdev': 0.1, 'iteration': 100, 'optimizer': 'SGD',
'reg': (False, False, 0), 'verbose': 0}
Get parameters before setting: {'batch_size_pernode': 100, 'dim': (True, True, 8),
'init_learn_rate': 0.01, 'init_stdev': 0.1, 'iteration': 200, 'optimizer': 'SGD',
'reg': (0, 0, 0), 'verbose': 0}
```

Return Value
It simply returns “self” reference.

10.3.1.9 9. debug_print()

Purpose
Currently, this method is not supported for FactorizationMachineRegressor. It is simply kept in Factoriza-

tionMachineRegressor module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

10.3.1.10 10. release()

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

For example,

```
fmr.release()
```

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

Return Value
It returns nothing.

10.3.1.11 11. 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'.

10.4 SEE ALSO

- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Factorization Machine Classifier in Frovedis**

Chapter 11

FPGrowth

11.1 NAME

FPGrowth - A frequent pattern mining algorithm supported by Frovedis.

11.2 SYNOPSIS

```
class frovedis.mllib.fpm.FPGrowth(minSupport=0.3, minConfidence=0.8, itemsCol='items',
                                  predictionCol='prediction', numPartitions=None,
                                  tree_depth=None, compression_point=4, mem_opt_level=0,
                                  verbose=0, encode_string_input=False)
```

11.2.1 Public Member Functions

fit(data)
generate_rules(confidence = None)
transform(data)
load(fname)
save(fname)
debug_print()
release()
is_fitted()

11.3 DESCRIPTION

FPGrowth is an algorithm for discovering frequent itemsets in a transaction database. The input of FPGrowth is a set of transactions called transaction database. Each transaction is a set of items. **Frovedis supports numeric and non-numeric values for transaction data.**

For example, consider the following transaction database. It contains 4 transactions (t1, t2, t3, t4) and 4 items (1, 2, 3, 4). The first transaction represents the set of items 1, 2 , 3 and 4.

```
Transaction id      Items
t1                  {1, 2, 3, 4}
t2                  {1, 2, 3}
```

```
t3                  {1, 2}
t4                  {1}
```

It is important to note that repetition of item in any given transaction needs to be avoided. It would raise exception in that case.

Now, if FPGrowth is run on the above transaction database with a minSupport of 40% and a tree_depth of 5 levels,

FPGrowth produces the following result:

```
items               freq
[1]                 4
[2]                 3
[3]                 2
[2, 1]              3
[3, 1]              2
[3, 2]              2
[3, 2, 1]           2
```

In the results, each itemset is annotated with its corresponding frequency.

This module provides a client-server implementation, where the client application is a normal python program. 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 FPGrowth 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 1. FPGrowth()

Parameters
minSupport: A positive double (float64) type value that specifies the minimum support level of frequent itemsets. Its value must be within 0 to 1. (Default: 0.3)
minConfidence: A positive double (float64) type value that specifies the minimal confidence for generating association rules. It will not affect the mining for frequent itemsets, but will affect the association rules generation. Its value must be within 0 to 1. (Default: 0.8)
itemsCol: An unsed parameter. (Default: 'items')
predictionCol: An unsed parameter. (Default: 'prediction')
numPartitions: An unsed parameter. (Default: None)
tree_depth: A positive integer parameter specifying the maximum number of levels for tree construction. Its value must be greater than 1. (Default: None)
When it is None (not specified explicitly), the tree is constructed to its maximum depth according to the data. Since transaction databases tend to be very large, there may be a scenario wherein entire FP tree cannot be contained in memory. In those cases, the size of FP tree may be limited by using this parameter.
compression_point: A positive integer parameter. This is an internal memory optimisation strategy which helps when working with large transaction databases. Its value must be greater than or equal to 2. No compression will be performed till the level specified by this parameter is reached. (Default: 4)
mem_opt_level: An integer value which must be either 0 (memory optimisation OFF) or 1 (memory optimisation ON). If switched On, it will lead to removal of redundant tree data residing in memory at server

side. It should only be used where systems have memory constraints. By default, it is 0, but in case of memory constraints, execution should be attempted keeping this value as 1, it will help in reducing memory footprint. However, when it is 1, it might still cause memory issue in case data is too big. In this case, data may be spilled onto disk if this environment variable has been set. This will degrade performance (execution time).
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.
encode_string_input: A boolean parameter when set to True, encodes the non-numeric (like strings) itemset values. It first internally encodes the named-items to an encoded numbered-items and the encoded dataframe is used for further training at frovedis server. It helps to train the encoded fpgrowth model faster with non-numeric itemsets since data present with server is two column dataframe, operations like join(), etc are little on slower on non-numeric columns than the numeric columns. (Default: False)

For example,

```
# let the data be some non-numeric transaction database
data = [['banana','apple','mango','cake'],
        ['cake','banana','apple'],
        ['bread','banana'],
        ['banana']]
# Using FPGrowth object with memory optimization parameters for training
# Here, disabling the parameter encode_string_input = False by default
from frovedis.mllib.fpm import FPGrowth
fpm = FPGrowth(minSupport = 0.01, minConfidence = 0.5, compression_point = 4,
               mem_opt_level = 1)
fpm.fit(data)
```

Frequent itemsets generation time: 0.0361 sec

And, when enabling 'encode_string_input' with the same non-numeric data,

```
# Using FPGrowth object with memory optimization parameters for training
# Here, parameter encode_string_input = True
from frovedis.mllib.fpm import FPGrowth
fpm = FPGrowth(minSupport = 0.01, minConfidence = 0.5, compression_point = 4,
               mem_opt_level = 1, encode_string_input = True)
fpm.fit(data)
```

Frequent itemsets generation time: 0.0315 sec

Attributes
freqItemsets: A pandas dataframe having two fields, 'items' and 'freq', where 'items' is an array whereas 'freq' is double (float64) type value. It contains the itemsets along with their frequency values. Here, the frequency of an itemset signifies as to how many times the itemset appears in the transaction database.
associationRules: A pandas dataframe having six fields, 'antecedent', 'consequent', 'confidence', 'lift', 'support' and 'conviction'.
Every association rule is composed of two parts: an antecedent (if) and a consequent (then).
An 'antecedent' is an item found within the data. A 'consequent' is an item found in combination with the 'antecedent'. Both are itemsets (arrays).
For measuring the effectiveness of association rule, 'confidence', 'lift', 'support' and 'conviction' are used. All are double (float64) type values.
'confidence' refers to the amount of times a given rule turns out to be true in practice.
'support' is an indication of how frequently the itemset appears in the dataset.
'lift' is the ratio of confidence to support. If the rule has a lift of 1, it would imply that the probability of occurrence of the 'antecedent' and that of the 'consequent' are independent of each other. When two events are independent of each other, no rule can be drawn involving those two events. If the lift is greater than 1, that lets us know the degree to which those two occurrences are dependent on one another, and makes those

rules potentially useful for predicting the consequent in future data sets. If the lift is less than 1, that lets us know the items are substitute to each other. This means that presence of one item has negative effect on presence of other item and vice versa.
'conviction' compares the probability that X appears without Y if they were dependent with the actual frequency of the appearance of X without Y. If it equals 1, then they are completely unrelated.

count: A positive integer value which specifies the frequent itemsets count.
encode_logic: A python dictionary having transaction items with a corresponding encoded number as key-value pairs. It is only available when transaction items are have string values and 'encode_string_input' parameter is set to True. Otherwise it is None. This is used internally to perform auto decoding of items during 'freqItemsets' construction.

For example,

```
# let data be some non-numeric transaction database
data = [['banana','apple','mango','cake'],
        ['cake','banana','apple'],
        ['bread','banana'],
        ['banana']]

# creating a pandas dataframe
import pandas as pd
dataDF = pd.DataFrame(data)

# Using FPGrowth object with memory optimization parameters and enabling encoding on non-numeric
# inputs and training
from frovedis.mllib.fpm import FPGrowth
fpm = FPGrowth(minSupport = 0.01, minConfidence = 0.5, compression_point = 4,
               mem_opt_level = 1, encode_string_input = True)
fpm.fit(data)
print("logic: ", fpm.encode_logic)
```

Output,

```
logic:  {1: 'apple', 2: 'banana', 3: 'bread', 4: 'cake', 5: 'mango'}
```

Purpose
It initializes an FPGrowth object with the given parameters.

The parameters: "itemsCol", "predictionCol" and "numPartitions" are simply kept in to make the interface uniform to the PySpark FPGrowth module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

11.3.1.2 2. fit(data)

Parameters
data: A python iterable or a pandas dataframe (manually constructed or loaded from file) or frovedis-two column dataframe containing the transaction data. Frovedis supports numeric and non-numeric values in transaction dataset.

Purpose
It accepts the training data and trains the fpgrowth model with specified minimum support value and tree depth value for construction of tree.

For pandas dataframe, if it has 2 columns, the second column would be treated as items and needs to be an array-like input.

For example,

```
# creating a pandas dataframe
import pandas as pd
data = [['banana','apple','mango','cake'],
        ['cake','banana','apple'],
        ['bread','banana'],
        ['banana']]
dataDF = pd.DataFrame(data)

# fitting input dataframe on FPGrowth object
fpm = FPGrowth(minSupport = 0.01, minConfidence = 0.5, compression_point = 4,
               mem_opt_level = 1).fit(dataDF)
# Now, to print the frequent itemsets table after training is completed
print(fpm.freqItemsets)
# to print table with all the association rules
print(fpm.associationRules)
```

Output,

```
frequent itemsets:
                           items  freq
0                       [banana]   4.0
1                        [apple]   2.0
2                         [cake]   2.0
3                        [bread]   1.0
4                        [mango]   1.0
5                  [cake, apple]   2.0
6                 [mango, apple]   1.0
7                [apple, banana]   2.0
8                [bread, banana]   1.0
9                 [cake, banana]   2.0
10               [mango, banana]   1.0
...
16  [mango, cake, apple, banana]   1.0
association rules:
                antecedent consequent  confidence  lift  support  conviction
0                 [banana]    [apple]         0.5   1.0     0.50         1.0
1                  [apple]     [cake]         1.0   2.0     0.50         NaN
2                 [banana]     [cake]         0.5   1.0     0.50         1.0
3                  [apple]    [mango]         0.5   2.0     0.25         1.5
4                   [cake]    [mango]         0.5   2.0     0.25         1.5
5                  [apple]   [banana]         1.0   1.0     0.50         NaN
6                  [bread]   [banana]         1.0   1.0     0.25         NaN
7                   [cake]    [apple]         1.0   2.0     0.50         NaN
8                   [cake]   [banana]         1.0   1.0     0.50         NaN
9                  [mango]    [apple]         1.0   2.0     0.25         NaN
10                 [mango]   [banana]         1.0   1.0     0.25         NaN
...
27    [mango, cake, apple]   [banana]         1.0   1.0     0.25         NaN
```

If it has more than 2 columns, all columns would be treated as individual item and missing items in any given transaction needs to be NaN.

For example,

FILE: groceries.csv

Item(s),Item 1,Item 2,Item 3,Item 4,Item 5,Item 6,Item 7, ... , Item 30,Item 31,Item 32
4,citrus fruit,semi-finished bread,margarine,ready soups,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
3,tropical fruit,yogurt,coffee,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
1,whole milk,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
4,pip fruit,yogurt,cream cheese,meat spreads,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
...
10,chicken,citrus fruit,other vegetables,butter,yogurt, ..., cling film/bags,,,,,,,,,,,,,,,,,,,,,,,
4,semi-finished bread,bottled water,soda,bottled beer,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
5,chicken,tropical fruit,other vegetables,vinegar,shopping bags,,,,,,,,,,,,,,,,,,,,,,,,,,,,

```
# reading data from csv file
import pandas as pd
dataDF = pd.read_csv("./input/groceries.csv", dtype=str).drop(['Item(s)'], axis=1)

# fitting input dataframe on FPGrowth object
fpm = FPGrowth(minSupport = 0.01, minConfidence = 0.5, compression_point = 4,
               mem_opt_level = 1).fit(dataDF)
# Now, to print the frequent itemsets table after training is completed
print(fpm.freqItemsets)
# to print table with all the association rules
print(fpm.associationRules)
```

Output

```
frequent itemsets:
                                        items    freq
0                                [whole milk]  2513.0
1                          [other vegetables]  1903.0
2                                [rolls/buns]  1809.0
3                                      [soda]  1715.0
4                                    [yogurt]  1372.0
..                                        ...     ...
328  [whipped/sour cream, yogurt, whole milk]   107.0
329    [yogurt, other vegetables, whole milk]   219.0
330    [yogurt, rolls/buns, other vegetables]   113.0
331          [yogurt, rolls/buns, whole milk]   153.0
332                [yogurt, soda, whole milk]   103.0
association rules:
                           antecedent         consequent    confidence    lift   support  conviction
0             [yogurt, other vegetables]     [whole milk]     0.512881  2.007235 0.022267 1.528340
1           [whipped/sour cream, yogurt]     [whole milk]     0.524510  2.052747 0.010880 1.565719
2 [whipped/sour cream, other vegetables]     [whole milk]     0.507042  1.984385 0.014642 1.510239
3                [tropical fruit, yogurt]     [whole milk]     0.517361  2.024770 0.015150 1.542528
4      [tropical fruit, root vegetables]     [whole milk]     0.570048  2.230969 0.011998 1.731553
5               [root vegetables, yogurt]     [whole milk]     0.562992  2.203354 0.014540 1.703594
6           [root vegetables, rolls/buns]     [whole milk]     0.523013  2.046888 0.012710 1.560804
7           [pip fruit, other vegetables]     [whole milk]     0.517510  2.025351 0.013523 1.543003
8      [domestic eggs, other vegetables]     [whole milk]     0.552511  2.162336 0.012303 1.663694
9                          [curd, yogurt]     [whole milk]     0.582353  2.279125 0.010066 1.782567
10            [butter, other vegetables]     [whole milk]     0.573604  2.244885 0.011490 1.745992
11     [tropical fruit, root vegetables] [other vegetables]   0.584541  3.020999 0.012303 1.941244
12              [root vegetables, yogurt] [other vegetables]   0.500000  2.584078 0.012913 1.613015
13          [root vegetables, rolls/buns] [other vegetables]   0.502092  2.594890 0.012201 1.619792
14       [citrus fruit, root vegetables] [other vegetables]   0.586207  3.029608 0.010371 1.949059
```

When native python iterable or a pandas dataframe is provided, it is converted to frovedis dataframe 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, especially when same data would be used for multiple executions. The storage representation when creating frovedis dataframe is slightly different than pandas dataframe as shown below :

```
# creating a pandas dataframe
import pandas as pd
data = [['banana','apple','mango','cake'],
        ['cake','banana','apple'],
        ['bread','banana'],
        ['banana']]
dataDF = pd.DataFrame(data)

# Creating a two column frovedis dataframe
import frovedis.dataframe as fpd
frovDF = fpd.DataFrame(dataDF)

# fitting frovedis-like input dataframe on FPGrowth object
fpm = FPGrowth(minSupport = 0.01, minConfidence = 0.5, compression_point = 4,
               mem_opt_level = 1).fit(frovDF)
# Now, to print the frequent itemsets table after training is completed
print(fpm.freqItemsets)
# to print table with all the association rules
print(fpm.associationRules)
```

Output,

```
frequent itemsets:
                          items  freq
0                      [banana]   4.0
1                       [apple]   2.0
2                        [cake]   2.0
3                       [bread]   1.0
4                       [mango]   1.0
5                 [cake, apple]   2.0
6                [mango, apple]   1.0
7               [apple, banana]   2.0
8               [bread, banana]   1.0
9                [cake, banana]   2.0
10              [mango, banana]   1.0
...
16  [mango, cake, apple, banana]   1.0
association rules:
               antecedent consequent  confidence  lift  support  conviction
0                [banana]    [apple]         0.5   1.0     0.50         1.0
1                 [apple]     [cake]         1.0   2.0     0.50         NaN
2                [banana]     [cake]         0.5   1.0     0.50         1.0
3                 [apple]    [mango]         0.5   2.0     0.25         1.5
4                  [cake]    [mango]         0.5   2.0     0.25         1.5
5                 [apple]   [banana]         1.0   1.0     0.50         NaN
6                 [bread]   [banana]         1.0   1.0     0.25         NaN
7                  [cake]    [apple]         1.0   2.0     0.50         NaN
8                  [cake]   [banana]         1.0   1.0     0.50         NaN
9                 [mango]    [apple]         1.0   2.0     0.25         NaN
10                [mango]   [banana]         1.0   1.0     0.25         NaN
...
```

```
27    [mango, cake, apple]   [banana]         1.0   1.0     0.25         NaN
```

Another way to work with frovedis dataframe is by loading it from a file. Since in real world, large transaction database will be used for rule mining, such files needs to have two columns named "trans_id" and "item", where "trans_id" column will have respective transaction id and "item" column will have the individual items in that transaction. If other names are present for the columns in the file, then it would raise exception.

For example:

FILE: trans.csv

trans_id,item 1,1
1,2
1,3
1,4
2,1
2,2
2,3
3,1
3,2
4,1

The above data can be loaded and passed to frovedis FPGrowth as follows:

```
# loadinf frovedis dataframe from a csv file
import frovedis.dataframe as fdf
frovDF = fdf.read_csv("trans.csv")

# fitting frovedis-like input dataframe on FPGrowth object
fpm = FPGrowth(minSupport = 0.01, minConfidence = 0.5, compression_point = 4,
               mem_opt_level = 1).fit(frovDF)
```

Return Value
It simply returns "self" reference.

11.3.1.3 3. generate_rules(confidence = None)

Parameters
confidence: A double (float64) type parameter indicating the minimum confidence value. (Default: None) When it is None (not specified explicitly), then it will use confidence value used during FPGrowth object creation.

Purpose
It accepts the minimum confidence value to trim the rules during the generation of association rules.

For example,

```
# generating rules with minimum confidence value of 0.2
fp_rules = fpm.generate_rules(0.2)
```

This will generate tables containing rules at server side.
To print these generated rules, debug_print() may be used as show below:

```
fp_rules.debug_print()
```

This will show all tables of different antecedent length at server side. Here, encoding was disabled during rule generation.

Output,

```
--- rule[0] ---
antecedent1   consequent   confidence   lift   support  conviction
      apple   cake         1            2      0.5      NULL
      apple   mango        0.5          2      0.25     1.5
     banana   apple        0.5          1      0.5      1
     banana   cake         0.5          1      0.5      1
       cake   mango        0.5          2      0.25     1.5

--- rule[1] ---
antecedent1   consequent   confidence   lift   support  conviction
       cake   apple        1            2      0.5      NULL
      mango   apple        1            2      0.25     NULL
      apple   banana       1            1      0.5      NULL
      bread   banana       1            1      0.25     NULL
       cake   banana       1            1      0.5      NULL
      mango   banana       1            1      0.25     NULL
      mango   cake         1            2      0.25     NULL

--- rule[2] ---
antecedent1   antecedent2  consequent  confidence  lift  support  conviction
      apple   banana       cake        1           2     0.5      NULL
       cake   banana       mango       0.5         2     0.25     1.5
      apple   banana       mango       0.5         2     0.25     1.5
       cake   apple        mango       0.5         2     0.25     1.5

--- rule[3] ---
antecedent1   antecedent2  consequent  confidence  lift  support  conviction
      mango   banana       apple       1           2     0.25     NULL
       cake   banana       apple       1           2     0.5      NULL
      mango   banana       cake        1           2     0.25     NULL
      mango   apple        cake        1           2     0.25     NULL

--- rule[4] ---
antecedent1   antecedent2  consequent  confidence  lift  support  conviction
      mango   cake         banana      1           1     0.25     NULL
      mango   apple        banana      1           1     0.25     NULL
       cake   apple        banana      1           1     0.5      NULL
      mango   cake         apple       1           2     0.25     NULL

--- rule[5] ---
antecedent1   antecedent2  antecedent3 consequent  confidence  lift  support  conviction
       cake   apple        banana           mango  0.5         2     0.25     1.5

--- rule[6] ---
antecedent1   antecedent2  antecedent3 consequent  confidence  lift  support  conviction
      mango   apple        banana          cake    1           2     0.25     NULL

--- rule[7] ---
antecedent1   antecedent2  antecedent3 consequent  confidence  lift  support  conviction
      mango   cake         banana          apple   1           2     0.25     NULL

--- rule[8] ---
antecedent1   antecedent2  antecedent3 consequent  confidence  lift  support  conviction
      mango   cake         apple           banana  1           1     0.25     NULL
```

This output will be visible on server side. No such output will be visible on client side.

These above generated rules can also be saved and loaded separately as shown below:

```
rule.save("./out/FPRule")
```

It saves the rules in 'FPRule' directory.
It would raise exception if the directory already exists with same name.

The 'FPRule' directory has

FPRule
|——— encode_logic
|——— **rule_0**
|——— **rule_1**
|——— **rule_2**
|——— **rule_3**
|——— **rule_4**
|——— **rule_5**
|——— **rule_6**
|——— **rule_7**
|——— **rule_8**

The 'encode_logic' file is created only when 'encode_string_input = True' while training.
Other directories are created according to number of rules created which were constructed during training. Each rule based directory contains information about antecedent, consequent, confidence, lift, support, conviction.

For loading the already saved rules, following should be done:

```
rule.load("./out/FPRule")
```

Return Value
It returns an FPRules object.

11.3.1.4 4. transform(data)

Parameters
data: A python iterable or a pandas dataframe or frovedis-like dataframe containing the transaction data. Frovedis supports numeric and non-numeric values as for transaction data.

Purpose
It gives the prediction of list of items for each of the corresponding transaction items.

For example,

```
print(fpm.transform(data))
```

Output

```
                          items     prediction
0  [banana, apple, mango, cake]             []
1         [cake, banana, apple]        [mango]
2               [bread, banana]  [apple, cake]
3                      [banana]  [apple, cake]
```

In case no prediction is made for any list of items, then it will give an empty list, just like in pandas dataframe as well.

Return Value
It returns a pandas dataframe having two fields 'items' and 'prediction'. Here 'items' is a list of transaction items and 'prediction' is a corresponding list of predictions for these transaction items.

11.3.1.5 5. load(fname)

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

Purpose
It loads the model from the specified file path.

For example,

```
fpm.load("./out/FPModel")
```

Return Value
It simply returns "self" reference.

11.3.1.6 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 in the specified file path. Otherwise, it throws an exception.

For example,

```
# To save the FPGrowth model
fpm.save("./out/FPModel")
```

This will save the fp-growth model on the path '/out/FPModel'.
It would raise exception if the directory already exists with same name.

The 'FPModel' directory has

FPModel
|——— encode_logic
|——— metadata
|——— **tree_0**
|——— **tree_1**
|——— **tree_2**
|——— **tree_3**

The 'encode_logic' file is created only when 'encode_string_input = True' while training.
The metadata file contains the number of transactions and rule frequent itemsets count.
Rest of the directories are created according to tree levels which were constructed during training.
Each tree based directory contains information about items and their frequency count on each tree level.

Return Value
It returns nothing

11.3.1.7 7. debug_print()

Purpose
It shows the target model information (frequent itemsets, generated fprules, frequent itemsets count) on the server side user terminal. It is mainly used for debugging purpose.

For example,

```
fpm.debug_print()
```

Output

```
--- item_support ---
item    item_support
banana  1
apple   0.5
cake    0.5
bread   0.25
mango   0.25

--- tree[0] ---
item    count
banana  4
apple   2
cake    2
bread   1
mango   1

--- tree_info[0] ---

--- tree[1] ---
item    item1   count
cake    apple   2
mango   apple   1
apple   banana  2
bread   banana  1
cake    banana  2
mango   banana  1
mango   cake    1

--- tree_info[1] ---

--- tree[2] ---
item    item1   item2   count
mango   cake    apple   1
cake    apple   banana  2
mango   apple   banana  1
mango   cake    banana  1

--- tree_info[2] ---

--- tree[3] ---
item    item1   item2   item3   count
mango   cake    apple   banana  1

--- tree_info[3] ---

total #FIS: 17
```

This output will be visible on server side. It displays the in memory frequent itemsets, generated fprules, frequent itemsets count which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing

11.3.1.8 8. release()

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

For example,

```
fpm.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 9. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

11.4 SEE ALSO

- **Introduction to frovedis DataFrame**

Chapter 12

GaussianMixture

12.1 NAME

GaussianMixture - It is a representation of a Gaussian mixture model probability distribution. This class allows to estimate the parameters of a Gaussian mixture distribution.

12.2 SYNOPSIS

```
class frovedis.mllib.mixture.gmm.GaussianMixture(n_components=1, covariance_type='full',
                                                  tol=1e-3, reg_covar=1e-6, max_iter=100,
                                                  n_init=1, init_params='kmeans',
                                                  weights_init=None, means_init=None,
                                                  precisions_init=None, random_state=None,
                                                  warm_start=False, verbose=0,
                                                  verbose_interval=10)
```

12.2.1 Public Member Functions

fit(X, y = None)
fit_predict(X, y = None)
predict(X)
predict_proba(X)
sample(n_samples = 1)
score(X, y = None)
score_samples(X)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
bic(X)
aic(X)
debug_print()
release()
is_fitted()

12.3 DESCRIPTION

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

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 GaussianMixture 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 nternally 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 GaussianMixture 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, the 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 1. GaussianMixture()

Parameters
n_components: A positive integer parameter specifying the number of mixture components. (Default: 1)
covariance_type: A string object parameter specifying the type of covariance parameters to use.
Currently, **'full'** is supported in frovedis. In 'full' covariance type, each component has its own general covariance matrix. (Default: 'full')
tol: Zero or a positive double (float64) parameter specifying the convergence tolerance. EM (expectation-maximization) iterations will stop when the lower bound average gain is below this threshold. (Default: 1e-3)
reg_covar: An unused parameter. (Default: 1e-6)
max_iter: A positive integer parameter specifying the number of EM (expectation-maximization) iterations to perform. (Default: 100)
n_init: A positive integer parameter specifying the number of initializations to perform. (Default: 1)
If it is None (not specified explicitly), it will be set as 1.
init_params: A string object parameter specifying the method used to initialize the weights, the means and the precisions. (Default: 'kmeans')
Must be one of the following:
- **'kmeans'** : responsibilities are initialized using kmeans.
- **'random'** : responsibilities are initialized randomly.

weights_init: An unused parameter. (Default: None)
means_init: An unused parameter. (Default: None)
precisions_init: An unused parameter. (Default: None)
random_state: An integer, float parameter that controls the random seed given to the method chosen to initialize the parameters. It does not support RandomState instance. (Default: None)
If it is None (not specified explicitly) or a RandomState instance, it will be set as 0.
warm_start: An unused parameter. (Default: False)
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.
verbose_interval: An unused parameter. (Default: 10)

Attribute
weights_: It is a python ndarray, containing double (float64) typed values and has shape **(n_components,)**. It stores the weights of each mixture components.
covariances_: It is a python ndarray, containing double (float64) typed values. It stores the covariance of each mixture component. The shape depends on **covariance_type**:
- if **'full'**, then the shape is **(n_components, n_features, n_features)**.

means_: It is a python ndarray, containing double (float64) typed values and has shape **(n_components, n_features)**. It stores the mean of each mixture component.
converged_: A boolean value. If True, then convergence was reached in fit(), otherwise it will be False.
n_iter_int: An integer value specifying the number of step used by the best fit of EM to reach the convergence.
lower_bound_: A float value specifying the lower bound value on the log-likelihood (of the training data with respect to the model) of the best fit of EM.

Purpose
It initializes a GaussianMixture object with the given parameters.

The parameters: "reg_covar", "weights_init,"means_init", "precisions_init", "warm_start", and "verbose_interval" are simply kept to make the interface uniform to Scikit-learn GaussianMixture module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

12.3.1.2 2. fit(X, y = 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)**. During training, **n_samples >= n_components**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It estimates the model parameters with the EM algorithm.

The method fits the model **'n_init'** times and sets the parameters with which the model has the largest likelihood or lower bound. Within each trial, the method iterates between E-step and M-step for **'max_iter'** times until the change of likelihood or lower bound is less than **'tol'**.

For example,

```
# loading sample matrix dense data
# train_mat = np.array([[1., 2.],
                        [1., 4.],
                        [1., 0.],
                        [10., 2.],
                        [10., 4.],
                        [10., 0.]])

# fitting input matrix on GaussianMixture object
from frovedis.mllib.mixture import GaussianMixture
gmm_model = GaussianMixture(n_components = 2)
gmm_model.fit(train_mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading sample matrix dense data
# train_mat = np.array([[1., 2.],
                        [1., 4.],
                        [1., 0.],
                        [10., 2.],
                        [10., 4.],
                        [10., 0.]])

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# fitting GaussianMixture object with pre-constructed input
from frovedis.mllib.mixture import GaussianMixture
gmm_model = GaussianMixture(n_components = 2)
gmm_model.fit(rmat)
```

Return Value
It simply returns "self" reference.

12.3.1.3 3. fit_predict(X, y = 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)**. During training, **n_samples >= n_components**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It estimates the model parameters using X and predict the labels for X.

The method fits the model **'n_init'** times and sets the parameters with which the model has the largest likelihood or lower bound. Within each trial, the method iterates between E-step and M-step for **'max_iter'** times until the change of likelihood or lower bound is less than **'tol'**. After fitting, it predicts the most probable label for the input data points.

For example,

```
# loading sample matrix dense data
# train_mat = np.array([[1., 2.],
                        [1., 4.],
                        [1., 0.],
                        [10., 2.],
                        [10., 4.],
                        [10., 0.]])

# fitting input matrix on GaussianMixture object and perform predictions
from frovedis.mllib.mixture import GaussianMixture
gmm_model = GaussianMixture(n_components = 2)
print(gmm_model.fit_predict(train_mat))
```

Output

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

Like in fit() frovedis-like input can be used to speed-up training at server side.

For example,

```
# loading sample matrix dense data
# train_mat = np.array([[1., 2.],
                        [1., 4.],
                        [1., 0.],
                        [10., 2.],
                        [10., 4.],
                        [10., 0.]])

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# fitting GaussianMixture object with pre-constructed input and perform predictions
from frovedis.mllib.mixture import GaussianMixture
gmm_model = GaussianMixture(n_components = 2)
print(gmm_model.fit_predict(rmat))
```

Output

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

Return Value
It returns a numpy array of int64 type values containing the components labels. It has a shape **(n_samples,)**.

12.3.1.4 4. 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 predict the labels for the data samples in X using trained model.

For example,

```
# loading sample matrix dense data
# train_mat = np.array([[1., 2.],
                        [1., 4.],
                        [1., 0.],
                        [10., 2.],
                        [10., 4.],
                        [10., 0.]])

# predicting on GaussianMixture model
from frovedis.mllib.mixture import GaussianMixture
gmm_model = GaussianMixture(n_components = 2).fit(train_mat)
print(gmm_model.predict(train_mat))
```

Output

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

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# loading sample matrix dense data
# train_mat = np.array([[1., 2.],
                        [1., 4.],
                        [1., 0.],
                        [10., 2.],
                        [10., 4.],
                        [10., 0.]])

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# predicting on GaussianMixture model using pre-constructed input
from frovedis.mllib.mixture import GaussianMixture
gmm_model = GaussianMixture(n_components = 2).fit(rmat)
print(gmm_model.predict(rmat))
```

Output

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

Return Value
It returns a numpy array of int64 type values containing the components labels. It has a shape **(n_samples,)**.

12.3.1.5 5. 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. It has shape **(n_samples, n_features)**.

Purpose

It predict the labels for the data samples in X using trained model.

For example,

```
# loading sample matrix dense data
# train_mat = np.array([[1., 2.],
                        [1., 4.],
                        [1., 0.],
                        [10., 2.],
                        [10., 4.],
                        [10., 0.]])

# predicting on GaussianMixture model
from frovedis.mllib.mixture import GaussianMixture
gmm_model = GaussianMixture(n_components = 2).fit(train_mat)
print(gmm_model.predict_proba(train_mat))
```

Output

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

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

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# loading sample matrix dense data
# train_mat = np.array([[1., 2.],
                        [1., 4.],
                        [1., 0.],
                        [10., 2.],
                        [10., 4.],
                        [10., 0.]])

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# predicting on GaussianMixture model using pre-constructed input
from frovedis.mllib.mixture import GaussianMixture
gmm_model = GaussianMixture(n_components = 2).fit(rmat)
print(gmm_model.predict_proba(rmat))
```

Output

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

Return Value
It returns a numpy array of double (float64) type values containing the density of each gaussian component for each sample in X. It has a shape **(n_samples, n_components)**.

12.3.1.6 6. sample(n_samples = 1)

Parameters
n_samples: A positive integer value that specifies the number of samples to generate. (Default: 1)

Purpose
Currently this method is not supported for GaussianMixture in frovedis.

Return Value
It simply raises a NotImplementedError.

12.3.1.7 7. score(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 computes the per-sample average log-likelihood of the given data X.

For example,

```
# calculate log-likelihood on given test data X
gmm_model.score(train_mat)
```

Output

```
3.386
```

Return Value
It returns a score of double (float64) type.

12.3.1.8 8. score_samples(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 computes the log-likelihood of each sample.

For example,

```
gmm_model.score_samples(train_mat)
```

Output

```
[3.88631622 3.1363165  3.1363165  3.88631622 3.1363165  3.1363165 ]
```

Return Value
It returns a numpy array of double (float64) type containing log-likelihood of each sample in 'X' under the current model. It has a shape **(n_samples,)**.

12.3.1.9 9. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by GaussianMixture. It is used to get parameters and their values of GaussianMixture class.

For example,

```
print(gmm_model.get_params())
```

Output

```
{'covariance_type': 'full', 'init_params': 'kmeans', 'max_iter': 100, 'means_init': None,
'n_components': 2, 'n_init': 1, 'precisions_init': None, 'random_state': None,
'reg_covar': None, 'tol': 0.001, 'verbose': 0, 'verbose_interval': None, 'warm_start': None,
'weights_init': None}
```

Return Value
A dictionary of parameter names mapped to their values.

12.3.1.10 10. set_params(**params)

Parameters
***params**: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by GaussianMixture, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(gmm_model.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
gmm_model.set_params(n_components = 4)
print("get parameters after setting:")
print(gmm_model.get_params())
```

Output

```
get parameters before setting:
{'covariance_type': 'full', 'init_params': 'kmeans', 'max_iter': 100, 'means_init': None,
'n_components': 2, 'n_init': 1, 'precisions_init': None, 'random_state': None,
'reg_covar': None, 'tol': 0.001, 'verbose': 0, 'verbose_interval': None, 'warm_start': None,
'weights_init': None}
get parameters after setting:
{'covariance_type': 'full', 'init_params': 'kmeans', 'max_iter': 100, 'means_init': None,
'n_components': 4, 'n_init': 1, 'precisions_init': None, 'random_state': None,
'reg_covar': None, 'tol': 0.001, 'verbose': 0, 'verbose_interval': None, 'warm_start': None,
'weights_init': None}
```

Return Value
It simply returns “self” reference.

12.3.1.11 11. 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
gmm_model.load("./out/MyGmmModel",dtype = np.float64)
```

Return Value
It simply returns “self” reference.

12.3.1.12 12. 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
gmm_model.save("./out/MyGmmModel")
```

This will save the random forest classifier model on the path "/out/MyGmmModel". It would raise exception if the directory already exists with same name.

The MyGmmModel contains below directory structure:

MyGmmModel
|——metadata
|——model

'metadata' represents the detail about n_components, n_features, converged_, n_iter_, lower_bound_, model_kind and datatype of training vector.
Here, the 'model' file contains information about gaussian mixture model in binary format.

Return Value
It returns nothing.

12.3.1.13 13. bic(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 is the Bayesian information criterion for the current model on the input X.

For example,

```
gmm_model.bic(train_mat)
```

Output

```
-20.926
```

Return Value
It returns a bayesian information criterion of double (float64) type.

12.3.1.14 14. aic(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 is the Akaike information criterion for the current model on the input X.

For example,

```
gmm_model.aic(train_mat)
```

Output

```
-18.636
```

Return Value
It returns an akaike information criterion of double (float64) type.

12.3.1.15 15. debug_print()

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

For example,

```
gmm_model.debug_print()
```

Output

```
Gaussian Mixture Model:
Means:
node = 0, local_num_row = 2, local_num_col = 2, val = 1 2 10 2

Weights:
node = 0, local_num_row = 2, local_num_col = 1, val = 0.5 0.5

Covariances:
node = 0, local_num_row = 2, local_num_col = 4, val = 0 0 0 2.66667 0 0 0 2.66667
```

This output will be visible on server side. It displays the information on the trained model such as means, weights, covariances which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

12.3.1.16 16. release()

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

For example,

```
gmm_model.release()
```

This will reset the after-fit populated attributes (like weights_, means_, etc.) to None, along with releasing server side memory.

Return Value
It returns nothing.

12.3.1.17 17. 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'.

12.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **KMeans in Frovedis**

Chapter 13

GradientBoostingClassifier

13.1 NAME

GradientBoostingClassifier - It is a machine learning algorithm, used for binary classification problems. It works on the principle that many weak learners (multiple decision trees) can together make a more accurate predictor.

13.2 SYNOPSIS

```
class frovedis.mllib.ensemble.gbtree.GradientBoostingClassifier(loss="deviance",
                                                                learning_rate=0.1,
                                                                n_estimators=100,
                                                                subsample=1.0,
                                                                criterion="friedman_mse",
                                                                min_samples_split=2,
                                                                min_samples_leaf=1,
                                                                min_weight_fraction_leaf=0.,
                                                                max_depth=3,
                                                                min_impurity_decrease=0.,
                                                                min_impurity_split=None,
                                                                init=None, random_state=None,
                                                                max_features=None, verbose=0,
                                                                max_leaf_nodes=None,
                                                                warm_start=False,
                                                                presort="deprecated",
                                                                validation_fraction=0.1,
                                                                n_iter_no_change=None,
                                                                tol=1e-4, ccp_alpha=0.0,
                                                                max_bins=32)
```

13.2.1 Public Member Functions

fit(X, y)
predict(X)
get_params(deep = True)

set_params(**params)
load(fname, dtype = None)
score(X, y, sample_weight = None)
save(fname)
debug_print()
release()
is_fitted()

13.3 DESCRIPTION

GradientBoostingClassifier is a supervised machine learning algorithm used for classification using decision trees. In gradient boosting decision trees, we combine many weak learners to come up with one strong learner. The weak learners here are the individual decision trees. All the trees are connected in series and each tree tries to minimise the error of the previous tree. Due to this sequential connection, boosting algorithms are usually slow to learn, but also highly accurate. **Currently, frovedis supports only binary gradient boosting classification algorithm.**

During training, the input **X** is the training data and **y** is the corresponding label values (Frovedis supports any values for labels, but internally it encodes the input binary labels to -1 and 1) before training at Frovedis server) which we want to predict.

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 GradientBoostingClassifier 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 GradientBoostingClassifier 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 1. GradientBoostingClassifier()

Parameters
loss: A string object parameter that specifies the loss function to be optimized. Currently, it supports only 'deviance' loss (logloss as in logistic regression) for classification. (Default: 'deviance')
learning_rate: A positive double (float64) parameter that shrinks the contribution of each tree by 'learning_rate' value provided. (Default: 0.1)
n_estimators: A positive integer parameter that specifies the number of boosting stages to perform. (Default: 100)
subsample: A positive double (float64) parameter that specifies the fraction of samples to be used for fitting the individual base learners. It must be in range **(0, 1.0]**. (Default: 1.0)
criterion: A string object parameter that specifies the function to measure the quality of a split. (Default: 'friedman_mse')
Currently, supported criteria are 'friedman_mse', 'mse' and 'mae'.
- **'friedman_mse'**: the mean squared error with improvement score by Friedman.

- **'mse'**: the mean squared error, which is equal to variance reduction as feature selection criterion.
- **'mae'**: the mean absolute error uses reduction in Poisson deviance to find splits.

min_samples_split: An unused parameter. (Default: 2)
min_samples_leaf: An unused parameter. (Default: 1)
min_weight_fraction_leaf: An unused parameter. (Default: 0.0)
max_depth: A positive integer parameter that specifies the maximum depth of the individual estimators. (Default: 3)
min_impurity_decrease: A positive double (float64) parameter. A node will be split if this split induces a decrease of the impurity greater than or equal to this value. (Default: 0.0)
min_impurity_split: An unused parameter. (Default: None)
init: An unused parameter. (Default: None)
random_state: An integer parameter that controls the random seed given to each tree estimator at each boosting iteration. In addition, it controls the random permutation of the features at each split. (Default: None)
If it is None (not specified explicitly), then 'random_state' is set as -1.
max_features: A string object parameter that specifies the number of features to consider when looking for the best split:
- If it is an integer, then it will be set as **(max_features * 1.0) / n_features_**.
- If it is float, then it will be **'max_features'** number of features at each split.
- If it is 'auto', then it will be set as **sqrt(n_features_)**.
- If 'sqrt', then it will be set as **sqrt(n_features_)** (same as 'auto').
- If 'log2', then it will be set as **log2(n_features_)**.
- If None, then it will be set as **n_features_**. (Default: 'None')

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.
max_leaf_nodes: An unused parameter. (Default: None)
warm_start: An unused parameter. (Default: False)
presort: An unused parameter. (Default: 'deprecated')
validation_fraction: An unused parameter. (Default: 0.1)
n_iter_no_change: An unused parameter. (Default: None)
tol: A double (float64) parameter that specifies the tolerance value for the early stopping. (Default: 1e-4)
ccp_alpha: An unused parameter. (Default: 0.0)
max_bins: A positive integer parameter that specifies the maximum number of bins created by ordered splits. (Default: 32)

Attributes
classes_: It is a python ndarray (any type) of unique labels given to the classifier during training. It has shape **(n_classes,)**.
n_estimators_: An integer value specifying the 'n_estimators' value.
n_features_: An integer value specifying the number of features when fitting the estimator.

Purpose
It initializes a GradientBoostingClassifier object with the given parameters.

The parameters: "min_samples_split", "min_samples_leaf", "min_weight_fraction_leaf", "min_impurity_split", "init", "max_leaf_nodes", "warm_start", "presort", "validation_fraction", "n_iter_no_change" and "ccp_alpha" are simply kept in to make the interface uniform to the Scikit-learn GradientBoostingClassifier module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

13.3.1.2 2. fit(X, y)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisColmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.
y: Any python array-like object or an instance of FrovedisDvector containing the target values. It has shape **(n_samples,)**.

Purpose
It fits the gradient boosting model.

For example,

```
# loading a sample matrix and labels dense data
import numpy as np
mat = np.array([[10, 0, 1, 0, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 0, 1],
                [1, 0, 0, 1, 0, 1, 0]])
lbl = np.array([1, 0, 1, 0])

# fitting input matrix and label on GradientBoostingClassifier object
from frovedis.mllib.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier(n_estimators = 2)
gbc.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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels dense data
import numpy as np
mat = np.array([[10, 0, 1, 0, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 0, 1],
                [1, 0, 0, 1, 0, 1, 0]])
lbl = np.array([1, 0, 1, 0])

# Since "mat" is numpy dense data, we have created FrovedisColmajorMatrix.
from frovedis.matrix.dense import FrovedisColmajorMatrix
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisColmajorMatrix(mat)
dlbl = FrovedisDvector(lbl)

# fitting input matrix and label on GradientBoostingClassifier object
from frovedis.mllib.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier(n_estimators = 2)
gbc.fit(cmat, dlbl)
```

Return Value
It simply returns “self” reference.

13.3.1.3 3. predict(X)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.

Purpose
Predict class for X.

For example,

```
# predicting on gradient boosting classifier model
gbc.predict(mat)
```

Output

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

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# predicting on gradient boosting classifier model
gbc.predict(cmat.to_frovedis_rowmatrix())
```

Output

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

Return Value
It returns a numpy array of int64 type and of shape **(n_samples,)** containing the predicted classes.

13.3.1.4 4. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by GradientBoostingClassifier. It is used to get parameters and their values of GradientBoostingClassifier class.

For example,

```
print(gbc.get_params())
```

Output

```
{'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1,
'loss': 'deviance', 'max_bins': 32, 'max_depth': 3, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 2, 'n_iter_no_change': None, 'presort': 'deprecated', 'random_state': -1,
'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0,
'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

13.3.1.5 5. set_params(**params)

Parameters
***params**: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by GradientBoostingClassifier, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(gbc.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
gbc.set_params(n_estimators = 5)
print("get parameters after setting:")
print(gbc.get_params())
```

Output

```
get parameters before setting:
{'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1,
'loss': 'deviance', 'max_bins': 32, 'max_depth': 3, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 2, 'n_iter_no_change': None, 'presort': 'deprecated', 'random_state': -1,
'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0,
'warm_start': False}
get parameters after setting:
{'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1,
'loss': 'deviance', 'max_bins': 32, 'max_depth': 3, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 5, 'n_iter_no_change': None, 'presort': 'deprecated', 'random_state': -1,
'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0,
'warm_start': False}
```

Return Value
It simply returns “self” reference.

13.3.1.6 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,

```
gbc.load("./out/gbt_classifier_model")
```

Return Value
It simply returns “self” reference.

13.3.1.7 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the gradient boosting classifier model
gbc.save("./out/gbt_classifier_model")
```

This will save the gradient boosted classifier model on the path "/out/gbt_classifier_model". It would raise exception if the directory already exists with same name.

The 'gbt_classifier_model' directory has

gbt_classifier_model
|—label_map
|—metadata
|—model

'label_map' file contains information about labels mapped with their encoded value.
The 'metadata' file contains the number of classes, model kind and input datatype used for trained model.
The 'model' file contains the gradient boosting model saved in binary format.

Return Value
It returns nothing.

13.3.1.8 8. score(X, y, sample_weight = None)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.
y: Any python array-like object containing the true labels for X. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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
gbc.score(mat,lbl)
```

Output

```
1.00
```

Return Value
It returns an accuracy score of float type.

13.3.1.9 9. debug_print()

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

For example,

```
gbc.debug_print()
```

Output

```
-------- Classification Trees (GBTs):: --------
# of trees: 2
---- [0] ----
  # of nodes: 3, height: 1
  <1> Split: feature[3] < 0.25, IG: 1
   \_ (2) Predict: 1
   \_ (3) Predict: -1
---- [1] ----
  # of nodes: 3, height: 1
  <1> Split: feature[3] < 0.25, IG: 0.227349
   \_ (2) Predict: 0.476812
   \_ (3) Predict: -0.476812
```

This output will be visible on server side. It displays the gradient boosting tree having maximum depth of 4 and total 2 decision trees.

No such output will be visible on client side.

Return Value
It returns nothing.

13.3.1.10 10. release()

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

For example,

```
gbc.release()
```

This will reset the after-fit populated attributes (like classes_, n_features_) to None, along with releasing server side memory.

Return Value
It returns nothing.

13.3.1.11 11. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

13.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Gradient Boosting Regressor in Frovedis**
- **Decision Tree Classifier in Frovedis**

Chapter 14

GradientBoostingRegressor

14.1 NAME

GradientBoostingRegressor - It is a machine learning algorithm, used for regression. It works on the principle that many weak learners (multiple decision trees) can together make a more accurate predictor.

14.2 SYNOPSIS

```
class frovedis.mllib.ensemble.gbtree.GradientBoostingRegressor(loss="ls",
                                                               learning_rate=0.1,
                                                               n_estimators=100,
                                                               subsample=1.0,
                                                               criterion="friedman_mse",
                                                               min_samples_split=2,
                                                               min_samples_leaf=1,
                                                               min_weight_fraction_leaf=0.,
                                                               max_depth=3,
                                                               min_impurity_decrease=0.,
                                                               min_impurity_split=None,
                                                               init=None, random_state=None,
                                                               max_features=None, alpha=0.9,
                                                               verbose=0,
                                                               max_leaf_nodes=None,
                                                               warm_start=False,
                                                               presort="deprecated",
                                                               validation_fraction=0.1,
                                                               n_iter_no_change=None,
                                                               tol=1e-4, ccp_alpha=0.0,
                                                               max_bins=32)
```

14.2.1 Public Member Functions

fit(X, y)
predict(X)
get_params(deep = True)

set_params(**params)
load(fname, dtype = None)
score(X, y, sample_weight = None)
save(fname)
debug_print()
release()
is_fitted()

14.3 DESCRIPTION

GradientBoostingRegressor is a supervised machine learning algorithm used for regression using decision trees. In gradient boosting decision trees, we combine many weak learners to come up with one strong learner. The weak learners here are the individual decision trees. All the trees are connected in series and each tree tries to minimise the error of the previous tree. Due to this sequential connection, boosting algorithms are usually slow to learn, but also highly accurate.

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 GradientBoostingRegressor 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 GradientBoostingRegressor 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.

14.3.1 Detailed Description

14.3.1.1 1. GradientBoostingRegressor()

Parameters
loss: A string object parameter that specifies the loss function to be optimized. Currently, it supports 'ls' and 'lad' loss. (Default: 'ls')
- **'ls'**: refers to least squares regression.
- **'lad' (least absolute deviation)**: it is a highly robust loss function solely based on order information of the input variables.

learning_rate: A positive double (float64) parameter that shrinks the contribution of each tree by 'learning_rate' value provided. (Default: 0.1)
n_estimators: A positive integer parameter that specifies the number of boosting stages to perform. (Default: 100)
subsample: A positive double (float64) parameter that specifies the fraction of samples to be used for fitting the individual base learners. It must be in range **(0, 1.0]**. (Default: 1.0)
criterion: A string object parameter that specifies the function to measure the quality of a split. (Default: 'friedman_mse')
Currently, supported criteria are 'friedman_mse', 'mse' and 'mae'.
- **'friedman_mse'**: the mean squared error with improvement score by Friedman.

- **'mse'**: the mean squared error, which is equal to variance reduction as feature selection criterion.
- **'mae'**: the mean absolute error uses reduction in Poisson deviance to find splits.

min_samples_split: An unused parameter. (Default: 2)
min_samples_leaf: An unused parameter. (Default: 1)
min_weight_fraction_leaf: An unused parameter. (Default: 0.0)
max_depth: A positive integer parameter that specifies the maximum depth of the individual regression estimators. It limits the number of nodes in the tree. (Default: 3)
min_impurity_decrease: A positive double (float64) parameter. A node will be split if this split induces a decrease of the impurity greater than or equal to this value. (Default: 0.0)
min_impurity_split: An unused parameter. (Default: None)
init: An unused parameter. (Default: None)
random_state: An integer parameter that controls the random seed given to each tree estimator at each boosting iteration. In addition, it controls the random permutation of the features at each split. (Default: None)
If it is None (not specified explicitly), then 'random_state' is set as -1.
max_features: A string object parameter that specifies the number of features to consider when looking for the best split:
- If it is an integer, then it will be set as **(max_features * 1.0) / n_features_**.
- If it is float, then it will be **'max_features'** number of features at each split.
- If it is 'auto', then it will be set as **sqrt(n_features_)**.
- If 'sqrt', then it will be set as **sqrt(n_features_)** (same as 'auto').
- If 'log2', then it will be set as **log2(n_features_)**.
- If None, then it will be set as **n_features_**. (Default: 'None')

alpha: An unused parameter. (Default: 0.9)
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.
max_leaf_nodes: An unused parameter. (Default: None)
warm_start: An unused parameter. (Default: False)
presort: An unused parameter. (Default: 'deprecated')
validation_fraction: An unused parameter. (Default: 0.1)
n_iter_no_change: An unused parameter. (Default: None)
tol: A double (float64) parameter that specifies the tolerance value for the early stopping. (Default: 1e-4)
ccp_alpha: An unused parameter. (Default: 0.0)
max_bins: A positive integer parameter that specifies the maximum number of bins created by ordered splits. (Default: 32)

Attributes
n_estimators_: An integer value specifying the 'n_estimators' value.
n_features_: An integer value specifying the number of features when fitting the estimator.

Purpose
It initializes a GradientBoostingRegressor object with the given parameters.

The parameters: "min_samples_split", "min_samples_leaf", "min_weight_fraction_leaf", "min_impurity_split", "init", "alpha", "max_leaf_nodes", "warm_start", "presort", "validation_fraction", "n_iter_no_change" and "ccp_alpha" are simply kept in to make the interface uniform to the Scikit-learn GradientBoostingRegressor module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

14.3.1.2 2. fit(X, y)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisColmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.
y: Any python array-like object or an instance of FrovedisDvector containing the target values. It has shape **(n_samples,)**.

Purpose
It fits the gradient boosting model.

For example,

```
# loading a sample matrix and labels dense data
import numpy as np
mat = np.array([[10, 0, 1, 0, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 0, 1],
                [1, 0, 0, 1, 0, 1, 0]])
lbl = np.array([1.2,0.3,1.1,1.9])

# fitting input matrix and label on GradientBoostingRegressor object
from frovedis.mllib.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor(n_estimators = 2)
gbr.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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels dense data
import numpy as np
mat = np.array([[10, 0, 1, 0, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 0, 1],
                [1, 0, 0, 1, 0, 1, 0]])
lbl = np.array([1.2,0.3,1.1,1.9])

# Since "mat" is numpy dense data, we have created FrovedisColmajorMatrix.
from frovedis.matrix.dense import FrovedisColmajorMatrix
from frovedis.matrix.dvector import FrovedisDvector
cmat = FrovedisColmajorMatrix(mat)
dlbl = FrovedisDvector(lbl)

# fitting input matrix and label on GradientBoostingRegressor object
from frovedis.mllib.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor(n_estimators = 2)
gbr.fit(cmat, dlbl)
```

Return Value
It simply returns “self” reference.

14.3.1.3 3. predict(X)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.

Purpose
Predict regression target for X.

For example,

```
# predicting on gradient boosting regressor model
gbr.predict(mat)
```

Output

```
[1.2 0.3 1.1 1.9]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# predicting on gradient boosting regressor model
gbr.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[1.2 0.3 1.1 1.9]
```

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

14.3.1.4 4. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by GradientBoostingRegressor. It is used to get parameters and their values of GradientBoostingRegressor class.

For example,

```
print(gbr.get_params())
```

Output

```
{'alpha': 0.9, 'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None,
'learning_rate': 0.1, 'loss': 'ls', 'max_bins': 32, 'max_depth': 3, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 2, 'n_iter_no_change': None, 'presort': 'deprecated', 'random_state': -1,
'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0,
'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

14.3.1.5 5. set_params(**params)

Parameters
***params**: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by GradientBoostingRegressor, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(gbr.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
gbr.set_params(n_estimators = 5)
print("get parameters after setting:")
print(gbr.get_params())
```

Output

```
get parameters before setting:
{'alpha': 0.9, 'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None,
'learning_rate': 0.1, 'loss': 'ls', 'max_bins': 32, 'max_depth': 3, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 2, 'n_iter_no_change': None, 'presort': 'deprecated', 'random_state': -1,
'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0,
'warm_start': False}
get parameters after setting:
{'alpha': 0.9, 'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None,
'learning_rate': 0.1, 'loss': 'ls', 'max_bins': 32, 'max_depth': 3, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 5, 'n_iter_no_change': None, 'presort': 'deprecated', 'random_state': -1,
'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0,
'warm_start': False}
```

Return Value
It simply returns “self” reference.

14.3.1.6 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,

```
gbr.load("./out/gbt_regressor_model")
```

Return Value
It simply returns “self” reference.

14.3.1.7 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the gradient boosting regressor model
gbr.save("./out/gbt_regressor_model")
```

This will save the gradient boosted regressor model on the path "/out/gbt_regressor_model". It would raise exception if the directory already exists with same name.

The 'gbt_regressor_model' directory has

gbt_regressor_model
|—metadata
|—model

The 'metadata' file contains the model kind and input datatype used for trained model.
The 'model' file contains the gradient boosting model saved in binary format.

Return Value
It returns nothing.

14.3.1.8 8. score(X, y, sample_weight = None)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.
y: Any python array-like object containing the true values for X. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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
gbr.score(mat,lbl)
```

Output

```
1.00
```

Return Value
It returns an R2 score of float type.

14.3.1.9 9. debug_print()

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

For example,

```
gbr.debug_print()
```

Output

```
-------- Regression Trees (GBTs):: --------
# of trees: 2
---- [0] ----
    # of nodes: 7, height: 2
    <1> Split: feature[1] < 0.25, IG: 0.180625
     \_ <2> Split: feature[0] < 5.5, IG: 0.1225
     |  \_ (4) Predict: 1.9
     |  \_ (5) Predict: 1.2
     \_ <3> Split: feature[3] < 0.5, IG: 0.16
         \_ (6) Predict: 1.1
         \_ (7) Predict: 0.3
---- [1] ----
    # of nodes: 1, height: 0
     (1) Predict: 0
```

This output will be visible on server side. It displays the gradient boosting tree having maximum depth of 4 and total 2 decision trees.

No such output will be visible on client side.

Return Value
It returns nothing.

14.3.1.10 10. release()

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

For example,

```
gbr.release()
```

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

Return Value
It returns nothing.

14.3.1.11 11. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

14.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Gradient Boosting Classifier in Frovedis**
- **Decision Tree Regressor in Frovedis**

Chapter 15

KMeans Clustering

15.1 NAME

KMeans - is a clustering algorithm commonly used in EDA (exploratory data analysis).

15.2 SYNOPSIS

```
class frovedis.mllib.cluster.KMeans(n_clusters=8, init='random', 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', use_shrink=False)
```

15.2.1 Public Member Functions

fit(X, y = None, sample_weight = None)
fit_predict(X, y = None, sample_weight = None)
fit_transform(X, y = None, sample_weight = None)
transform(X)
predict(X, sample_weight = None)
score(X, y = None, sample_weight = None)
load(fname, dtype = None)
save(fname)
get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

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. Kmeans is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters.

Under unsupervised learning, there are two clustering methods- 'k-means' and 'k-means++'. The main difference between these two lies in the selection of the centroids (we assume centroid is the center of the cluster) around which the clustering takes place.

Frovedis supports only k-means clustering method (i.e. init ='random') which will randomly initialize the data points called centroid. Further, each data point is clustered to its nearest centroid and after every iteration the centroid is updated for each cluster. This cycle continues for a given number of repetitions and after that we have our final clusters.

This module provides a client-server implementation, where the client application is a normal python program. Frovedis is almost same as Scikit-learn clustering module providing kmeans 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 Kmeans 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, the 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.

15.3.1 Detailed Description

15.3.1.1 1. KMeans()

Parameters
n_clusters: An integer parameter specifying the number of clusters. The number of clusters should be greater than zero and less than n_samples. (Default: 8)
When it is None (specified explicitly), then it will be set as min(8, n_samples).
init: A string object parameter specifies the method of initialization. (Default: 'random')
Unlike Scikit-learn, currently it only supports 'random' initialization.
n_init: A positive integer specifying the number of times the kmeans algorithm will be run with different centroid seeds. (Default: 10)
When it is None (specified explicitly), then it will be set as default 10.
max_iter: A positive integer parameter specifying the maximum iteration count. (Default: 300)
tol: Zero or a positive double (float64) parameter specifying the convergence tolerance. (Default: 1e-4)
precompute_distances: A string object parameter. (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: A zero or positive integer parameter. When it is None (not specified explicitly), it will be set as 0. (unused)
copy_x: A boolean parameter. (unused)
n_jobs: An integer parameter. (unused)
algorithm: A string object parameter, specifies the kmeans algorithm to use. (Default: auto)
When it is 'auto', it will be set as 'full'. Unlike Scikit-learn, currently it supports only 'full'.
use_shrink: A boolean parameter applicable only for "sparse" input (X). When set to True for sparse input, it can improve training performance by reducing communication overhead across participating processes. (Default: False)

Attribute
cluster_centers_: It is a python ndarray, containing float or double (float64) typed values and has shape **(n_clusters, n_features)**. These are the coordinates of cluster centers.

labels__: A python ndarray of int64 values and has shape **(n__clusters,)**. It contains predicted cluster labels for each point.
inertia__: A float parameter specifies the sum of squared distances of samples to their closest cluster center, weighted by the sample weights if provided.
n__iter__: An integer parameter specifies the number of iterations to run.

Purpose
It initializes a Kmeans object with the given parameters.

The parameters: "precompute_distances", "random_state", "copy_x" and "n_jobs" are simply kept to make the interface uniform to Scikit-learn cluster module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

15.3.1.2 2. fit(X, y = None, 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 of float or double (float64) type. It has shape **(n__samples, n__features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.
sample__weight: An unused parameter whose default value is None. It is simply ignored in frovedis implementation, like in Scikit-learn.

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

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# fitting input matrix on kmeans object
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1).fit(train_mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# KMeans with pre-constructed frovedis-like inputs
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1).fit(rmat)
```

Return Value
It simply returns "self" reference.

15.3.1.3 3. fit_predict(X, y = None, 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.
sample_weight: An unused parameter whose default value is None. It is simply ignored in frovedis implementation.

Purpose
It clusters the given data points (X) into a predefined number of clusters (n_clusters) and predicts the cluster index for each sample.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# fitting input matrix on KMeans object
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1)
print(kmeans.fit_predict(train_mat))
```

Output

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

Like in fit() frovedis-like input can be used to speed-up training at server side.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# using pre-constructed input matrix
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1)
print(kmeans.fit_predict(rmat))
```

Output

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

Return Value
It returns a numpy array of int64 type containing the cluster labels. It has a shape **(n_samples,)**.

15.3.1.4 4. fit_transform(X, y = None, 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.
sample_weight: An unused parameter whose default value is None and simply ignored in frovedis implementation.

Purpose
It computes clustering and transforms training data (X) to cluster-distance space.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# fitting input matrix on KMeans object
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1)
print(kmeans.fit_transform(train_mat))
```

Output

```
[[ 0.08660254 15.58845727]
[ 0.08660254 15.41525219]
[15.32864965  0.17320508]
[15.50185473  0.        ]
[15.67505981  0.17320508]]
```

If training data (X) is a numpy array or a scipy sparse matrix, it will return a new numpy dense array.

Like in fit() frovedis-like input can be used to speed-up training at server side.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("sample_data.txt")

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# using pre-constructed input matrix
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1)
# it returns a FrovedisRowmajorMatrix object
kmeans.fit_transform(rmat)
```

If training data (X) is a frovedis-like input, it will return a FrovedisRowmajorMatrix object.

Return Value
- **When training data is native-python data**:
Then the output will be a numpy array containing the transformed matrix.
- **When training data is frovedis-like data**:
Then the output will be a FrovedisRowmajorMatrix.

In both cases output would be of float or double (float64) type (depending upon input dtype) and of shape **(n_samples, n_clusters)**.

Note that even if training data (X) is sparse, the output would typically be dense.

15.3.1.5 5. transform(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 of float or double (float64) type. It has shape **(n_samples, n_features)**.

Purpose
It transforms the test data (X) to a cluster-distance space.

For example,

```
# loading sample matrix data
test_mat = np.loadtxt("sample_data.txt")

# fitting input matrix on kmeans object
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1).fit(test_mat)
print(kmeans.transform(test_mat))
```

Output

```
[[ 0.08660254 15.58845727]
[ 0.08660254 15.41525219]
[15.32864965  0.17320508]
[15.50185473  0.        ]
[15.67505981  0.17320508]]
```

If test data (X) is a numpy array or a scipy sparse matrix, it will return a new numpy dense array.

Like in fit() frovedis-like input can be used to speed-up training the test data at server side.

For example,

```
# loading sample matrix data
test_mat = np.loadtxt("sample_data.txt")

# Since "test_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
tr_mat = FrovedisRowmajorMatrix(test_mat)

# using pre-constructed input matrix
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1).fit(tr_mat)
kmeans.transform(tr_mat)
```

If test data (X) is a frovedis-like input, it will return FrovedisRowmajorMatrix object.

Return Value
- **If native-python data is input:**
Then it returns a numpy array containing the transformed matrix.
- **If frovedis-like data is input:**
Then it returns a FrovedisRowmajorMatrix.

In both cases output would be of float or double (float64) type (depending upon input dtype) and of shape **(n_samples, n_clusters)**.

Note that even if test data (X) is sparse, the output would typically be dense.

15.3.1.6 6. predict(X, 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
sample_weight: None or any python array-like object containing the intended weights for each input samples. It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It accepts the test data points (X) and returns the closest cluster each sample in X belongs to.

For example,

```
# loading sample matrix data
test_mat = np.loadtxt("sample_data.txt")

# fitting input matrix on KMeans object
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1).fit(test_mat)
print(kmeans.predict(test_mat))
```

Output

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

Like in fit() frovedis-like input can be used to speed-up prediction on the test data at server side.

For example,

```
# loading sample matrix data
test_mat = np.loadtxt("sample_data.txt")

# Since "test_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
tr_mat = FrovedisRowmajorMatrix(test_mat)

# using pre-constructed input matrix
from frovedis.mllib.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 1).fit(tr_mat)
print(kmeans.predict(tr_mat))
```

Output

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

Return Value
It returns a numpy array of int32 type containing the centroid values. It has a shape **(n_samples,)**.

15.3.1.7 7. score(X, y = None, 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.
sample_weight: None or any python array-like object containing the intended weights for each input samples. It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It is calculated as "-1.0 * inertia", which an indication of how far the points are from the centroids. Bad scores will return a large negative number, whereas good scores return a value close to zero.

For example,

```
kmeans.score(test_mat)
```

Output

```
-0.07499999552965164
```

Return Value
It returns a score of float type.

15.3.1.8 8. 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
kmeans.load("./out/MyKMeansModel",dtype=np.float64)
```

Return Value
It simply returns "self" instance.

15.3.1.9 9. 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(metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
kmeans.save("./out/MyKMeansModel")
```

The MyKMeansModel contains below directory structure:
MyKMeansModel
|——metadata
|——**model**

'metadata' represents the detail about model_id, model_kind and datatype of training vector.
Here, the **model** directory contains information about n_clusters_, n_features, model_kind and datatype of training vector.

This will save the Kmeans model on the path '/out/MyKMeansModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

15.3.1.10 10. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by Kmeans. It is used to get parameters and their values of Kmeans class.

For example,

```
print(kmeans.get_params())
```

Output

```
{'algorithm': 'auto', 'copy_x': True, 'init': 'random', 'max_iter': 300, 'n_clusters': 2,
'n_init': 1, 'n_jobs': 1, 'precompute_distances': 'auto', 'random_state': None,
'tol': 0.0001, 'use_shrink': False, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

15.3.1.11 11. set_params(**params)

Parameters
****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by Kmeans, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(kmeans.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
kmeans.set_params(n_clusters = 4, n_init = 5)
print("Get parameters after setting:")
print(kmeans.get_params())
```

Output

```
Get parameters before setting:
{'algorithm': 'auto', 'copy_x': True, 'init': 'random', 'max_iter': 300,
'n_clusters': 2, 'n_init': 1, 'n_jobs': 1,'precompute_distances': 'auto',
'random_state': None,'tol': 0.0001,'use_shrink': False, 'verbose': 0}
Get parameters after setting: {'algorithm': 'auto', 'copy_x': True,
```

```
'init': 'random', 'max_iter': 300, 'n_clusters': 4, 'n_init': 5,
'n_jobs': 1, 'precompute_distances': 'auto', 'random_state': None,
'tol': 0.0001,'use_shrink': False, 'verbose': 0}
```

Return Value
It simply returns "self" reference.

15.3.1.12 12. debug_print()

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

For example,

```
kmeans.debug_print()
```

Output

```
centroid:
node = 0, local_num_row = 2, local_num_col = 3, val = 0.15 0.15 0.15 9.1 9.1 9.1
```

This output will be visible on server side. It displays the centroid information on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

15.3.1.13 13. release()

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

For example,

```
kmeans.release()
```

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

Return Value
It returns nothing.

15.3.1.14 14. 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 clustering model first.

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

15.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**

- **Introduction to FrovedisCRSMatrix**
- **Agglomerative Clustering in Frovedis**
- **Spectral Clustering in Frovedis**
- **DBSCAN in Frovedis**

Chapter 16

KNeighborsClassifier

16.1 NAME

KNeighborsClassifier - Classifier implementing the k-nearest neighbors vote.

16.2 SYNOPSIS

```
class frovedis.mllib.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform',
                                                    algorithm='auto', leaf_size=30, p=2,
                                                    metric='euclidean', metric_params=None,
                                                    n_jobs=None, verbose=0, chunk_size=1.0,
                                                    batch_fraction=None)
```

16.2.1 Public Member Functions

fit(X, y)
kneighbors(X = None, n_neighbors = None, return_distance = True)
kneighbors_graph(X = None, n_neighbors = None, mode = 'connectivity')
save(fname)
load(fname)
predict(X, save_proba = False)
predict_proba(X)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

16.3 DESCRIPTION

Neighbors-based classification is a type of instance-based learning or non-generalizing learning. It does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point- a query point is assigned the

data class which has the most representatives within the nearest neighbors of the point. **Frovedis supports both binary and multinomial labels.**

The k-neighbors classification in KNeighborsClassifier is a commonly used technique. The optimal choice of the value is highly data-dependent. In general, a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.

During training, the input **X** is the training data and **y** are their corresponding label values (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.

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 KNeighborsClassifier 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 KNeighborsClassifier 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 kneighbors-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.

16.3.1 Detailed Description

16.3.1.1 1. KNeighborsClassifier()

Parameters
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for 'kneighbors' queries. It must be within the range of 0 and n_samples. (Default: 5)
weights: An unused parameter. (Default: uniform)
algorithm: A string object parameter, specifying the algorithm used to compute the nearest neighbors. (Default: auto)
When it is 'auto', it will be set as 'brute'(brute-force search approach). Unlike Scikit-learn, currently it supports only 'brute'.
leaf_size: An unused parameter. (Default: 30)
p: An unused parameter. (Default: 2)
metric: A string object parameter specifying the distance metric to use for the tree. (Default: 'euclidean')
Currenlty it only supports 'euclidean', 'seuclidean' and 'cosine' distance.
metric_params: An unused parameter. (Default: None)
n_jobs: An unused parameter. (Default: None)
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.
chunk_size: A positive float parameter specifying the amount of data (in megabytes) to be processed in one time. (Default: 1.0)
batch_fraction: A positive double (float64) parameter used to calculate the batches of specific size. These batches are used to construct the distance matrix. It must be within the range of 0.0 to 1.0. (Default: None)
When it is None (not specified explicitly), it will be set as np.finfo(np.float64).max value.

Attributes
classes_: A numpy array of int64 type value that specifies unique labels given to the classifier during training. It has shape **(n_samples,)**.

Purpose
It initializes a KNeighborsClassifier object with the given parameters.

The parameters: “weights”,“leaf_size”, “p”,“metric_params” and “n_jobs” are simply kept in to make the interface uniform to the Scikit-learn KNeighborsClassifier module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns “self” reference.

16.3.1.2 2. fit(X, y)

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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: Any python array-like object or an instance of FrovedisDvector containing the target labels. It has shape **(n_samples,)**.

Purpose
It accepts the training matrix (X) with labels (y) and trains a KNeighborsClassifier model.

For example,

```
# loading sample data
samples = np.loadtxt("./input/knc_data.txt", dtype = np.float64)
lbl = [10, 10, 10, 20, 10, 20]

# fitting input data on KNeighborsClassifier object
from frovedis.mllib.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors = 3)
knc.fit(samples, 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, especially when same data would be used for multiple executions.

For example,

```
# loading sample data
samples = np.loadtxt("./input/knc_data.txt", dtype = np.float64)
lbl = [10, 10, 10, 20, 10, 20]

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

# fitting input data on KNeighborsClassifier object
from frovedis.mllib.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors = 3)
knc.fit(rmat, dlbl)
```

Return Value
It simply returns “self” reference.

16.3.1.3 3. kneighbors(X = None, n_neighbors = None, return_distance = True)

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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where 'n_queries' is the number of rows in the test data. (Default: None)
When it is None (not specified explicitly), it will be training data (X) used as input in fit().
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for 'kneighbors' queries. It must be within the range of 0 and n_samples. (Default: None)
When it is None (not specified explicitly), it will be 'n_neighbors' value used during KNeighborsClassifier object creation.
return_distance: A boolean parameter specifying whether or not to return the distances. (Default: True)
If set to False, it will not return distances. Then, only indices are returned by this method.

Purpose
Finds the k-Neighbors of a point and returns the indices of neighbors and distances to the neighbors of each point.

For example,

```
distances, indices = knc.kneighbors(samples)
print('distances')
print(distances)
print('indices')
print(indices)
```

Output

```
distances
[[0.         1.         2.23606798]
 [0.         1.         1.41421356]
 [0.         1.41421356 2.23606798]
 [0.         1.         2.23606798]
 [0.         1.         1.41421356]
 [0.         1.41421356 2.23606798]]
indices
[[0 1 2]
 [1 0 2]
 [2 1 0]
 [3 4 5]
 [4 3 5]
 [5 4 3]]
```

Like in fit(), frovedis-like input can be used to speed-up the computation of indices and distances at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# fitting input data on KNeighborsClassifier object
from frovedis.mllib.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors = 3)
distances, indices = knc.kneighbors(rmat)
```

```
# Here FrovedisRowmajorMatrix().debug_print() is used
print('distances')
distances.debug_print()
# Here FrovedisRowmajorMatrix().debug_print() is used
print('indices')
indices.debug_print()
```

Output

```
distances
matrix:
num_row = 6, num_col = 3
node 0
node = 0, local_num_row = 6, local_num_col = 3, val = 0 1 2.23607 0 1 1.41421 0 1.41421
2.23607 0 1 2.23607 0 1 1.41421 0 1.41421 2.23607
indices
matrix:
num_row = 6, num_col = 3
node 0
node = 0, local_num_row = 6, local_num_col = 3, val = 0 1 2 1 0 2 2 1 0 3 4 5 4 3 5 5 4 3
```

It returns distances and indices as FrovedisRowmajorMatrix objects.

Return Value
1. **When test data and training data used by fitted model are python native input:**
- ***distances***: A numpy array of float or double (float64) type values. It has shape **(n_queries, n_neighbors)**, where 'n_queries' is the number of rows in the test data. It is only returned by kneighbors(), if return_distance = True.
- ***indices***: A numpy array of int64 type values. It has shape **(n_queries, n_neighbors)**.

2. **When either test data or training data used by fitted model is frovedis-like input:**
   - ***distances***: A FrovedisRowmajorMatrix of float or double (float64) type values. It has shape **(n_queries, n_neighbors)**, where 'n_queries' is the number of rows in the test data. It is only returned by kneighbors(), if return_distance = True.

   - ***indices***: A FrovedisRowmajorMatrix of int64 typess values. It has shape **(n_queries, n_neighbors)**.

16.3.1.4 4. kneighbors_graph(X = None, n_neighbors = None, mode = 'connectivity')

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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where 'n_queries' is the number of rows in the test data. (Default: None)
When it is None (not specified explicitly), it will be training data (X) used as input in fit().
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for 'kneighbors' queries. It must be within the range of 0 and n_samples. (Default: None)
When it is None (not specified explicitly), it will be 'n_queries' value used during NearestNeighbors object creation.
mode: A string object parameter which can be either 'connectivity' or 'distance'. It specifies the type of returned matrix.
For 'connectivity', it will return the connectivity matrix with ones and zeros, whereas for 'distance', the edges are euclidean distance between points, type of distance depends on the selected 'metric' value in KNeighborsClassifier class. (Default: 'connectivity')

Purpose
It computes the (weighted) graph of k-Neighbors for points in X.

For example,

```
# Here 'mode = connectivity' by default
graph = knc.kneighbors_graph(samples)
print('kneighbors graph')
print(graph)
```

Output

```
kneighbors graph
(0, 0)        1.0
(0, 1)        1.0
(0, 2)        1.0
(1, 1)        1.0
(1, 0)        1.0
(1, 2)        1.0
(2, 2)        1.0
(2, 1)        1.0
(2, 0)        1.0
(3, 3)        1.0
(3, 4)        1.0
(3, 5)        1.0
(4, 4)        1.0
(4, 3)        1.0
(4, 5)        1.0
(5, 5)        1.0
(5, 4)        1.0
(5, 3)        1.0
```

For example, when mode = 'distance'

```
# Here 'mode = distance'
graph = knc.kneighbors_graph(samples, mode = 'distance')
print('kneighbors graph')
print(graph)
```

Output

```
(0, 0)        0.0
(0, 1)        1.0
(0, 2)        2.23606797749979
(1, 1)        0.0
(1, 0)        1.0
(1, 2)        1.4142135623730951
(2, 2)        0.0
(2, 1)        1.4142135623730951
(2, 0)        2.23606797749979
(3, 3)        0.0
(3, 4)        1.0
(3, 5)        2.23606797749979
(4, 4)        0.0
(4, 3)        1.0
(4, 5)        1.4142135623730951
(5, 5)        0.0
(5, 4)        1.4142135623730951
```

```
(5, 3)        2.23606797749979
```

Like in fit(), frovedis-like input can be used to speed-up the graph making at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# fitting input data on KNeighborsClassifier object
from frovedis.mllib.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors = 3)
# Here 'mode = connectivty' by default
graph = knc.kneighbors_graph(rmat)

# Here FrovedisCRSMatrix().debug_print() is used
print('graph')
graph.debug_print()
```

Output

```
matrix:
num_row = 6, num_col = 6
node 0
local_num_row = 6, local_num_col = 6
val : 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
idx : 0 1 2 1 0 2 2 1 0 3 4 5 4 3 5 5 4 3
off : 0 3 6 9 12 15 18
```

It returns a FrovedisCRSMatrix object.

Return Value
- **When test data and training data used by fitted model are python native input:**
It returns a scipy sparse matrix of float or double (float64) type values. It has shape **(n_queries, n_samples_fit)**, where 'n_queries' is the number of rows in the test data and 'n_samples_fit' is the number of samples in the fitted data.
- **When either test data or training data used by fitted model is frovedis-like input:**
It returns a FrovedisCRSMatrix of float or double (float64) type values. It has shape **(n_queries, n_samples_fit)**, where 'n_queries' is the number of rows in the test data and 'n_samples_fit' is the number of samples in the fitted data.

16.3.1.5 5. save(fname)

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

Purpose
Currently, this method is not supported for KNeighborsClassifier. It is simply kept in KNeighborsClassifier module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

16.3.1.6 6. load(fname)

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

Purpose
Currently, this method is not supported for KNeighborsClassifier. It is simply kept in KNeighborsClassifier module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

16.3.1.7 7. predict(X, save_proba = False)

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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where 'n_queries' is the number of rows in the test data.
save_proba: A boolean parameter specifies whether to save the predicted probability or not.
If it is set to 'True', a matrix with the name 'probability_matrix' would be generated in the current execution directory, which can be used for inspecting the probability of individual classes (just for debug purpose).

Purpose
Predict the class labels for the provided data.

For example,

```
# predicting on KNeighborsClassifier model
knc.predict(samples)
```

Output

```
[10 10 10 20 20 20]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# predicting on KNeighborsClassifier model using pre-constructed input
knc.predict(rmat)
```

Output

```
[10 10 10 20 20 20]
```

Return Value
It returns a numpy array of long (int64) type containing the predicted outputs. It is of shape **(n_queries,)**, where 'n_queries' is the number of rows in the test data.

16.3.1.8 8. 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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where 'n_queries' is the number of rows in the test data.

Purpose
Calculates probability for the test data X.

For example,

```
knc.predict_proba(samples)
```

Output

```
[[1.         0.        ]
 [1.         0.        ]
 [1.         0.        ]
 [0.33333333 0.66666667]
 [0.33333333 0.66666667]
 [0.33333333 0.66666667]]
```

Like in fit(), frovedis-like input can be used to speed-up the computation of probability at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# fitting input data on KNeighborsClassifier object
from frovedis.mllib.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors = 3)
proba = knc.predict_proba(rmat)

# Here FrovedisRowmajorMatrix().debug_print() is used
print(proba)
proba.debug_print()
```

Output

```
matrix:
num_row = 6, num_col = 2
node 0
node = 0, local_num_row = 6, local_num_col = 2, val = 1 0 1 0 1 0 0.333333 0.666667
0.333333 0.666667 0.333333 0.666667
```

It returns a FrovedisRowmajorMatrix object.

Return Value
- When test data and training data used by fitted model are python native input :
A numpy array of float or double (float64) type values. It has shape **(n_queries, n_neighbors)**.
- When either test data or training data used by fitted model is frovedis-like input:
A FrovedisRowmajorMatrix of float or double (float64) type values. It has shape **(n_queries, n_neighbors)**.

16.3.1.9 9. 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.

y: Any python array-like object containing the target labels. It has shape **(n_samples,)**.
sample_weight: An unused parameter whose default value is None. It is simply ignored in frovedis implementation, just like in Scikit-learn.

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

For example,

```
knc.score(samples, lbl)
```

Output

```
0.89
```

Return Value
It returns an accuracy score of double (float64) type.

16.3.1.10 9. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by KNeighborsClassifier. It is used to get parameters and their values of KNeighborsClassifier class.

For example,

```
print(knc.get_params())
```

Output

```
{'algorithm': 'auto', 'batch_fraction': None, 'chunk_size': 1.0, 'leaf_size': 30,
'metric': 'euclidean', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 3,
'p': 2, 'verbose': 0, 'weights': 'uniform'}
```

Return Value
A dictionary of parameter names mapped to their values.

16.3.1.11 10. set_params(**params)

Parameters
****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by KNeighborsClassifier, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(knc.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
knc.set_params(n_neighbors = 5)
print("Get parameters after setting:")
print(knc.get_params())
```

Output

```
Get parameters before setting:
{'algorithm': 'auto', 'batch_fraction': None, 'chunk_size': 1.0,
'leaf_size': 30, 'metric': 'euclidean', 'metric_params': None, 'n_jobs': None,
'n_neighbors': 3, 'p': 2, 'verbose': 0, 'weights': 'uniform'}
Get parameters after setting:
{'algorithm': 'auto', 'batch_fraction': None, 'chunk_size': 1.0, 'leaf_size': 30,
'metric': 'euclidean', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5,
'p': 2, 'verbose': 0, 'weights': 'uniform'}
```

Return Value
It simply returns "self" reference.

16.3.1.12 11. debug_print()

Purpose
Currently, this method is not supported for KNeighborsClassifier. It is simply kept in KNeighborsClassifier module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

16.3.1.13 12. release()

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

For example,

```
knc.release()
```

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

Return Value
It returns nothing.

16.3.1.14 13. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

16.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **KNeighbors Regressor in Frovedis**

Chapter 17

KNeighborsRegressor

17.1 NAME

KNeighborsRegressor - Regression based on k-nearest neighbors.

17.2 SYNOPSIS

```
class frovedis.mllib.neighbors.KNeighborsRegressor(n_neighbors=5, weights='uniform',
                                                   algorithm='auto', leaf_size=30, p=2,
                                                   metric='euclidean', metric_params=None,
                                                   n_jobs=None, verbose=0,
                                                   chunk_size=1.0, batch_fraction=None)
```

17.2.1 Public Member Functions

fit(X, y)
kneighbors(X = None, n_neighbors = None, return_distance = True)
kneighbors_graph(X = None, n_neighbors = None, mode = 'connectivity')
save(fname)
load(fname)
predict(X)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

17.3 DESCRIPTION

Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based on the mean of the labels of its nearest neighbors.

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 KNeighborsRegressor 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 KNeighborsRegressor 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 kneighbors-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.

17.3.1 Detailed Description

17.3.1.1 1. KNeighborsRegressor()

Parameters
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for 'kneighbors' queries. It must be within the range of 0 and n_samples. (Default: 5)
weights: An unused parameter. (Default: uniform)
algorithm: A string object parameter, specifying the algorithm used to compute the nearest neighbors. (Default: auto)
When it is 'auto', it will be set as 'brute'(brute-force search approach). Unlike Scikit-learn, currently it supports only 'brute'.
leaf_size: An unused parameter. (Default: 30)
p: An unused parameter. (Default: 2)
metric: A string object parameter specifying the distance metric to use for the tree. (Default: 'euclidean')
Currenlty it only supports 'euclidean', 'seuclidean' and 'cosine' distance.
metric_params: An unused parameter. (Default: None)
n_jobs: An unused parameter. (Default: None)
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.
chunk_size: A positive float parameter specifying the amount of data (in megabytes) to be processed in one time. (Default: 1.0)
batch_fraction: A positive double (float64) parameter used to calculate the batches of specific size. These batches are used to construct the distance matrix. It must be within the range of 0.0 to 1.0. (Default: None)
When it is None (not specified explicitly), it will be set as np.finfo(np.float64).max value.

Purpose
It initializes a KNeighborsRegressor object with the given parameters.

The parameters: "weights","leaf_size", "p","metric_params" and "n_jobs" are simply kept in to make the interface uniform to the Scikit-learn KNeighborsRegressor module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

17.3.1.2 2. fit(X, y)

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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: Any python array-like object or an instance of FrovedisDvector containing the target labels. It has shape **(n_samples,)**.

Purpose
It accepts the training matrix (X) with labels (y) and trains a KNeighborsRegressor model.

For example,

```
# loading sample data
samples = np.loadtxt("./input/knr_data.txt", dtype = np.float64)
lbl = [10, 10, 10, 20, 10, 20]

# fitting input data on KNeighborsRegressor object
from frovedis.mllib.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor(n_neighbors = 3)
knr.fit(samples, 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, especially when same data would be used for multiple executions.

For example,

```
# loading sample data
samples = np.loadtxt("./input/knr_data.txt", dtype = np.float64)
lbl = [10, 10, 10, 20, 10, 20]

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

# fitting input data on KNeighborsRegressor object
from frovedis.mllib.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor(n_neighbors = 3)
knr.fit(rmat, dlbl)
```

Return Value
It simply returns “self” reference.

17.3.1.3 3. kneighbors(X = None, n_neighbors = None, return_distance = True)

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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where ‘n_queries’ is the number of rows in the test data. (Default: None)
When it is None (not specified explicitly), it will be training data (X) used as input in fit().
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for ‘kneighbors’ queries. It must be within the range of 0 and n_samples. (Default: None)

When it is None (not specified explicitly), it will be 'n_neighbors' value used during KNeighborsRegressor object creation.
return_distance: A boolean parameter specifying whether or not to return the distances. (Default: True) If set to False, it will not return distances. Then, only indices are returned by this method.

Purpose
Finds the k-Neighbors of a point and returns the indices of neighbors and distances to the neighbors of each point.

For example,

```
distances, indices = knr.kneighbors(samples)
print('distances')
print(distances)
print('indices')
print(indices)
```

Output

```
distances
[[0.         1.         2.23606798]
 [0.         1.         1.41421356]
 [0.         1.41421356 2.23606798]
 [0.         1.         2.23606798]
 [0.         1.         1.41421356]
 [0.         1.41421356 2.23606798]]
indices
[[0 1 2]
 [1 0 2]
 [2 1 0]
 [3 4 5]
 [4 3 5]
 [5 4 3]]
```

Like in fit(), frovedis-like input can be used to speed-up the computation of indices and distances at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# fitting input data on KNeighborsRegressor object
from frovedis.mllib.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor(n_neighbors = 3)
distances, indices = knr.kneighbors(rmat)

# Here FrovedisRowmajorMatrix().debug_print() is used
print('distances')
distances.debug_print()

# Here FrovedisRowmajorMatrix().debug_print() is used
print('indices')
indices.debug_print()
```

Output

```
distances
matrix:
num_row = 6, num_col = 3
node 0
node = 0, local_num_row = 6, local_num_col = 3, val = 0 1 2.23607 0 1 1.41421 0 1.41421
2.23607 0 1 2.23607 0 1 1.41421 0 1.41421 2.23607
indices
matrix:
num_row = 6, num_col = 3
node 0
node = 0, local_num_row = 6, local_num_col = 3, val = 0 1 2 1 0 2 2 1 0 3 4 5 4 3 5 5 4 3
```

It returns distances and indices as FrovedisRowmajorMatrix objects.

Return Value
1. **When test data and training data used by fitted model are python native input :**
- ***distances***: A numpy array of float or double (float64) type values. It has shape **(n_queries, n_neighbors)**, where 'n_queries' is the number of rows in the test data. It is only returned by kneighbors(), if return_distance = True.
- ***indices***: A numpy array of int64 type values. It has shape **(n_queries, n_neighbors)**.

2. **When test data and training data used by fitted model is frovedis-like input::**
   - ***distances***: A FrovedisRowmajorMatrix of float or double (float64) type values. It has shape **(n_queries, n_neighbors)**, where 'n_queries' is the number of rows in the test data. It is only returned by kneighbors(), if return_distance = True.

   - ***indices***: A FrovedisRowmajorMatrix of int64 typess values. It has shape **(n_queries, n_neighbors)**.

17.3.1.4 4. kneighbors_graph(X = None, n_neighbors = None, mode = 'connectivity')

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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where 'n_queries' is the number of rows in the test data. (Default: None)
When it is None (not specified explicitly), it will be training data (X) used as input in fit().
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for 'kneighbors' queries. It must be within the range of 0 and n_samples. (Default: None)
When it is None (not specified explicitly), it will be 'n_neighbors' value used during KNeighborsRegressor object creation.
mode: A string object parameter which can be either 'connectivity' or 'distance'. It specifies the type of returned matrix.
For 'connectivity', it will return the connectivity matrix with ones and zeros, whereas for 'distance', the edges are euclidean distance between points, type of distance depends on the selected 'metric' value in KNeighborsRegressor class. (Default: 'connectivity')

Purpose
It computes the (weighted) graph of k-Neighbors for points in X.

For example,

```
# Here 'mode = connectivity' by default
graph = knr.kneighbors_graph(samples)
print('kneighbors graph')
print(graph)
```

Output

```
kneighbors graph
(0, 0)        1.0
(0, 1)        1.0
(0, 2)        1.0
(1, 1)        1.0
(1, 0)        1.0
(1, 2)        1.0
(2, 2)        1.0
(2, 1)        1.0
(2, 0)        1.0
(3, 3)        1.0
(3, 4)        1.0
(3, 5)        1.0
(4, 4)        1.0
(4, 3)        1.0
(4, 5)        1.0
(5, 5)        1.0
(5, 4)        1.0
(5, 3)        1.0
```

For example, when mode = 'distance'

```
# Here 'mode = distance'
graph = knr.kneighbors_graph(samples, mode = 'distance')
print('kneighbors graph')
print(graph)
```

Output

```
kneighbors graph
(0, 0)        0.0
(0, 1)        1.0
(0, 2)        2.23606797749979
(1, 1)        0.0
(1, 0)        1.0
(1, 2)        1.4142135623730951
(2, 2)        0.0
(2, 1)        1.4142135623730951
(2, 0)        2.23606797749979
(3, 3)        0.0
(3, 4)        1.0
(3, 5)        2.23606797749979
(4, 4)        0.0
(4, 3)        1.0
(4, 5)        1.4142135623730951
(5, 5)        0.0
(5, 4)        1.4142135623730951
(5, 3)        2.23606797749979
```

Like in fit(), frovedis-like input can be used to speed-up the graph making at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)
```

```
# fitting input data on KNeighborsRegressor object
from frovedis.mllib.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor(n_neighbors = 3)

# Here 'mode = connectivty' by default
graph = knr.kneighbors_graph(rmat)

# Here FrovedisCRSMatrix().debug_print() is used
print('graph')
graph.debug_print()
```

Output

```
graph
Active Elements:  18
matrix:
num_row = 6, num_col = 6
node 0
local_num_row = 6, local_num_col = 6
val : 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
idx : 0 1 2 1 0 2 2 1 0 3 4 5 4 3 5 5 4 3
off : 0 3 6 9 12 15 18
```

It returns a FrovedisCRSMatrix object for frovedis-like input.

Return Value
- **When test data and training data used by fitted model are python native input:**
It returns a scipy sparse matrix of float or double (float64) type values. It has shape **(n_queries, n_samples_fit)**, where 'n_queries' is the number of rows in the test data and 'n_samples_fit' is the number of samples in the fitted data.
- **When either test data or training data used by fitted model is frovedis-like input:**
It returns a FrovedisCRSMatrix of float or double (float64) type values. It has shape **(n_queries, n_samples_fit)**, where 'n_queries' is the number of rows in the test data and 'n_samples_fit' is the number of samples in the fitted data.

17.3.1.5 5. save(fname)

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

Purpose
Currently, this method is not supported for KNeighborsRegressor. It is simply kept in KNeighborsRegressor module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

17.3.1.6 6. load(fname)

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

Purpose
Currently, this method is not supported for KNeighborsRegressor. It is simply kept in KNeighborsRegressor module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

17.3.1.7 7. 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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where 'n_queries' is the number of rows in the test data.

Purpose
Predict the class labels for the provided data.

For example,

```
# predicting on KNeighborsRegressor model
knr.predict(samples)
```

Output

```
[10.         10.         10.         16.66666667 16.66666667 16.66666667]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# predicting on KNeighborsRegressor model using pre-constructed input
knr.predict(rmat)
```

Output

```
[10.         10.         10.         16.66666667 16.66666667 16.66666667]
```

Return Value
It returns a numpy array of long (int64) type containing the predicted outputs. It is of shape **(n_queries,)**, where 'n_queries' is the number of rows in the test data.

17.3.1.8 8. 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: Any python array-like object or an instance of FrovedisDvector containing the target values. It has shape **(n_samples,)**.
sample_weight: An unused parameter whose default value is None. It is simply ignored in frovedis implementation, like in Scikit-learn.

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,

```
knr.score(samples, lbl)
```

Output

```
0.5
```

Return Value
It returns an R2 score of float type.

17.3.1.9 9. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by KNeighborsRegressor. It is used to get parameters and their values of KNeighborsRegressor class.

For example,

```
print(knr.get_params())
```

Output

```
{'algorithm': 'auto', 'batch_fraction': None, 'chunk_size': 1.0, 'leaf_size': 30,
'metric': 'euclidean', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 3,
'p': 2, 'verbose': 0, 'weights': 'uniform '}
```

Return Value
A dictionary of parameter names mapped to their values.

17.3.1.10 10. set_params(**params)

Parameters
****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by KNeighborsRegressor, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(knr.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
knr.set_params(n_neighbors = 5)
print("Get parameters after setting:")
print(knr.get_params())
```

Output

```
Get parameters before setting:
{'algorithm': 'auto', 'batch_fraction': None, 'chunk_size': 1.0, 'leaf_size': 30,
'metric': 'euclidean', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 3,
'p': 2, 'verbose': 0, 'weights': 'uniform '}
Get parameters after setting:
{'algorithm': 'auto', 'batch_fraction': None, 'chunk_size': 1.0, 'leaf_size': 30,
'metric': 'euclidean', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5,
'p': 2, 'verbose': 0, 'weights': 'uniform '}
```

Return Value
It simply returns "self" reference.

17.3.1.11 11. debug_print()

Purpose
Currently, this method is not supported for KNeighborsRegressor. It is simply kept in KNeighborsRegressor module to maintain uniform interface like other estimators in frovedis.

Return Value
It simply raises an AttributeError.

17.3.1.12 12. release()

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

For example,

```
knr.release()
```

This will remove the trained model, model-id present on server, along with releasing server side memory.

Return Value
It returns nothing.

17.3.1.13 13. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

17.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **KNeighbors Classifier in Frovedis**

Chapter 18

Lasso Regression

18.1 NAME

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

18.2 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', lr_rate=1e-8,
                                        verbose=0, solver='sag')
```

18.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

18.3 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
```

During training, the input **X** is the training data and **y** is the corresponding label values 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
(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 Lasso 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 Lasso 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.

18.3.1 Detailed Description

18.3.1.1 1. Lasso()

Parameters
alpha: A constant that multiplies the L1 term. It must be a positive value of double (float64) type . (Default: 0.01)
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)
normalize: An unused parameter. (Default: False)
precompute: An unused parameter. (Default: False)
copy_X: An unused parameter. (Default: True)
max_iter: A positive integer value used to set the maximum number of iterations. (Default: 1000)
tol: Zero or a positive value of double (float64) type specifying the convergence tolerance value. (Default: 1e-4)
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)
positive: An unused parameter. (Default: False)
random_state: An unused parameter. (Default: None)

selection: An unused paremeter. (Default: cyclic)
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.
solver: A string object specifying the solver to use. (Default: 'sag')
It can be "sag" for frovedis side stochastic gradient descent or "lbfgs" for frovedis side LBFGS optimizer when optimizing the lasso regression model. **Both "sag" and "lbfgs" can handle L1 penalty.**

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 lasso regression problem. It has shape **(n_features,)**.
intercept_(bias): It is a python ndarray (containing 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 initializes a Lasso object with the given parameters.

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

Return Value
It simply returns "self" reference.

18.3.1.2 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 containing the target values. It has shape **(n_samples,)**.
sample_weight: Python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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

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 as well to speed up the training time, especially when same data would be used for multiple executions.

Return Value
It simply returns "self" reference.

18.3.1.3 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.

In case pre-constructed frovedis-like training data such as FrovedisColmajorMatrix (X) is provided during prediction, then "X.to_frovedis_rowmatrix()" will be used for prediction.

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

18.3.1.4 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 containing true values for X. It has shape **(n_samples,)**.
sample_weight: Python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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.

Return Value
It returns an R2 score of float type.

18.3.1.5 5. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by Lasso. It is used to get parameters and their values of Lasso class.

Return Value
A dictionary of parameter names mapped to their values.

18.3.1.6 6. set_params(**params)

Parameters
*****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by Lasso, used to set parameter values.

Return Value
It simply returns "self" reference.

18.3.1.7 7. 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).

Return Value
It simply returns "self" reference.

18.3.1.8 8. 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

Suppose 'LassoModel' directory is the model created, It will have

LassoModel
|——— metadata
|——— model

The metadata file contains the number of classes, model kind, input datatype used for trained model.
Here, the model file contains information about weights, intercept and threshold.

It would raise exception if the 'LassoModel' directory already existed with same name.

Return Value
It returns nothing.

18.3.1.9 9. debug_print()

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

No such output will be visible on client side.

Return Value
It returns nothing.

18.3.1.10 10. release()

Purpose
It can be used to release the in-memory model at frovedis server. With this, after-fit populated attributes are reset to None, along with releasing server side memory.

Return Value
It returns nothing.

18.3.1.11 11. 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 lasso regression model first.

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

18.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Linear Regression in Frovedis**
- **Ridge Regression in Frovedis**

Chapter 19

Latent Dirichlet Allocation

19.1 NAME

Latent Dirichlet Allocation(LDA) - It is a Natural Language Processing algorithm that allows a set of observations to be explained by unobserved groups. These are used to explain why some parts of the data are similar.

19.2 SYNOPSIS

```
class frovedis.mllib.decomposition.LatentDirichletAllocation(n_components=10,
                                                             doc_topic_prior=None,
                                                             topic_word_prior=None,
                                                             learning_method='batch',
                                                             learning_decay=.7,
                                                             learning_offset=10.,
                                                             max_iter=10, batch_size=128,
                                                             evaluate_every=-1,
                                                             total_samples=1e6,
                                                             perp_tol=1e-1,
                                                             mean_change_tol=1e-3,
                                                             max_doc_update_iter=100,
                                                             n_jobs=None, verbose=0,
                                                             random_state=None,
                                                             algorithm="original",
                                                             explore_iter=0)
```

19.2.1 Public Member Functions

fit(X, y = None)
transform(X)
perplexity(X, sub_sampling = False)
score(X, y = None)
get_params(deep = True)
set_params(**params)
fit_transform(X, y = None)

load(fname, dtype = None)
save(fname)
release()
is_fitted()

19.3 DESCRIPTION

Latent Dirichlet Allocation (LDA) is a widely used machine learning technique for topic modeling. The input is a corpus of documents and the output is per-document topic distribution and per-topic word distribution.

This implementation provides python wrapper interface to Vectorised LDA (VLDA), which accelerates LDA training by exploiting the data-level, and the thread-level parallelism using vector processors. The priority-aware scheduling approach is proposed to address the high memory requirement and workload imbalance issues with existing works.

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 LatentDirichletAllocation 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 LatentDirichletAllocation 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 transform-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.

19.3.1 Detailed Description

19.3.1.1 1. LatentDirichletAllocation()

Parameters
n_components: A positive integer parameter specifying the number of topics. (Default: 10)
doc_topic_prior: A float parameter which specifies the prior of document topic distribution theta. It is also called alpha. (Default: None)
If it is None (not specified explicitly), it will be set as (1 / n_components).
topic_word_prior: A float parameter which specifies the prior of topic word distribution beta. It is also called beta. (Default: None)
If it is None (not specified explicitly), it will be set as (1 / n_components).
learning_method: An unused parameter which specifies the method used to update 'components_'. Although, this parameter is unused in frovedis but it must be 'batch' or 'online' update. This is simply done to keep the behavior consistent with Scikit-learn. (Default: batch)
learning_decay: An unused parameter. (Default: 0.7)
learning_offset: An unused parameter. Although, this parameter is unused in frovedis but it must be zero or a positive float value. This is simply done to keep the behavior consistent with Scikit-learn. (Default: 10.0)
max_iter: An integer parameter which specifies the maximum number of passes over the training data. (Default: 10)
batch_size: An unused parameter. (Default: 128)
evaluate_every: An integer parameter that specifies how often to evaluate perplexity. Perplexity is not

evaluated in training by default. If this parameter is greater than 0, perplexity will be evaluated, which can help in checking convergence in training process, but total training time will be increased. (Default: -1)

For example, when evaluate_every < 0

```
# loading sample csr matrix data representing the input corpus-vocabulary count
corpus = [
'This is the first document.',
'This document is the second document.',
'And this is the third one.',
'Is this the first document?',
]

# feature transformers CountVectorizer can be used for converting text to word count vectors.
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
csr_mat = vectorizer.fit_transform(corpus)

# fitting input matrix on lda object
from frovedis.mllib.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components = 3, evaluate_every = -1)
lda.fit(csr_mat)
```

LDA training time: 0.0007402896881103516

For example, when evaluate_every > 0

```
# loading sample csr matrix data representing the input corpus-vocabulary count
corpus = [
'This is the first document.',
'This document is the second document.',
'And this is the third one.',
'Is this the first document?',
]

# feature transformers CountVectorizer can be used for converting text to word count vectors.
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
csr_mat = vectorizer.fit_transform(corpus)

# fitting input matrix on lda object
from frovedis.mllib.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components = 3, evaluate_every = 2)
lda.fit(csr_mat)
```

LDA training time: 0.0008301734924316406

total_samples: A positive integer parameter which is unused in frovedis. This is simply kept to make the behavior consistent with Scikit-learn. (Default: 1e6)
perp_tol: An unused parameter. (Default: 1e-1)
mean_change_tol: An unused parameter. (Default: 1e-3)
max_doc_update_iter: An unused parameter. (Default: 100)
n_jobs: An unused parameter. (Default: None)
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.
algorithm: A string object parameter which specifies the sampling technique to be used. (Default: 'original')

Frovedis LatentDirichletAllocation(LDA) supports either of two sampling techniques:
1. Collapsed Gibbs Sampling (CGS)
2. Metropolis Hastings
The default sampling algorithm is CGS (Default: 'original')
If Metropolis Hastings is to be used, then it is required to set the proposal types namely:
2a. document proposal: "dp" algo,
2b. word proposal: "wp" algo,
2c. cycle proposal: "cp" algo,
2d. sparse lda: "sparse"
explore_iter: An integer parameter that specifies the number of iterations to explore optimal hyperparams. (Default: 0)

Attributes
components_: A numpy ndarray of double (float64) type values and has shape **(n_components, n_features)**.

Purpose
It initializes a LatentDirichletAllocation object with the given parameters.

The parameters: "learning_method", "learning_decay", "learning_offset", "batch_size", "total_samples", "perp_tol", "mean_change_tol", "max_doc_update_iter" and "n_jobs" are simply kept in to make the interface uniform to the Scikit-learn LatentDirichletAllocation module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

19.3.1.2 2. fit(X, y = None)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix for sparse data of int, float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
Fit LatentDirichletAllocation model on training data X.

For example,

```
# loading sample csr matrix data representing the input corpus-vocabulary count
corpus = [
'This is the first document.',
'This document is the second document.',
'And this is the third one.',
'Is this the first document?',
]

# feature transformers CountVectorizer can be used for converting text to word count vectors.
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
csr_mat = vectorizer.fit_transform(corpus)

# fitting input matrix on lda object
from frovedis.mllib.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components = 3)
lda.fit(csr_mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading sample csr matrix data representing the input corpus-vocabulary count
corpus = [
'This is the first document.',
'This document is the second document.',
'And this is the third one.',
'Is this the first document?',
]

# feature transformers CountVectorizer can be used for converting text to word count vectors.
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
csr_mat = vectorizer.fit_transform(corpus)

# Since "csr_mat" is scipy sparse data, we have created FrovedisCRSMatrix
# Also it only supports int64 as itype for input sparse data during training
from frovedis.matrix.crs import FrovedisCRSMatrix
cmat = FrovedisCRSMatrix(csr_mat, dtype = np.int32, itype = np.int64)

# fitting input matrix on lda object
from frovedis.mllib.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components = 3)
lda.fit(cmat)
```

Return Value
It simply returns "self" reference.

19.3.1.3 3. transform(X)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix for sparse data of int, float or double (float64) type. It has shape **(n_samples, n_features)**.

Purpose
It transforms input matrix X according to the trained model.

For example,

```
# loading sample csr matrix data representing the input corpus-vocabulary count
test_corpus = [
'This is the first second third document.',
'This This one third document.'
]

# feature transformers CountVectorizer can be used for converting text to word count vectors.
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
csr_mat = vectorizer.fit_transform(test_corpus)

# transform input according to trained model
from frovedis.mllib.decomposition import LatentDirichletAllocation
```

```
lda = LatentDirichletAllocation(n_components = 3)
lda.fit(csr_mat)
print(lda.transform(csr_mat))
```

Output

```
[[0.47619048 0.19047619 0.47619048]
 [0.26666667 0.26666667 0.66666667]]
```

Like in fit(), we can also provide frovedis-like input in transform() for faster computation.

For example,

```
# loading sample csr matrix data representing the input corpus-vocabulary count
test_corpus = [
'This is the first second third document.',
'This This one third document.'
]

# feature transformers CountVectorizer can be used for converting text to word count vectors.
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
csr_mat = vectorizer.fit_transform(test_corpus)

# Since "csr_mat" is scipy sparse data, we have created FrovedisCRSMatrix
# Also it only supports int64 as itype for input sparse data during training
from frovedis.matrix.crs import FrovedisCRSMatrix
cmat = FrovedisCRSMatrix(csr_mat, dtype = np.int32, itype = np.int64)

# transform input according to trained model
from frovedis.mllib.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components = 3)
lda.fit(cmat)
lda.transform(cmat).debug_print()
```

Output

```
matrix:
num_row = 2, num_col = 3
node 0
node = 0, local_num_row = 2, local_num_col = 3, val = 0.47619 0.333333 0.333333 0.266667
0.0666667 0.866667
```

Return Value
- **When X is python native input:**
It returns a python ndarray of shape **(n_samples, n_components)** and double (float64) type values. It contains the document-wise topic distribution for input data X.

- **When X is frovedis-like input:**
  It returns a FrovedisRowmajorMatrix of shape **(n_samples, n_components)** and double (float64) type values, containing document-wise topic distribution for input data X.

19.3.1.4 4. perplexity(X, sub_sampling = False)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix for sparse data of int, float, double (float64) type. It has shape **(n_samples, n_features)**.

sub_sampling: A boolean parameter that specifies whether to do sub-sampling or not. It is simply ignored in frovedis implementation. (Default: False)

Purpose
Calculate approximate perplexity for data X.

Perplexity is defined as exp(-1. * log-likelihood per word).

Ideally, as the number of components increase, the perplexity of model should decrease.

For example,

```
perp = lda.perplexity(csr_mat)
print("perplexity: %.2f" % (perp))
```

Ouput

```
perplexity: 8.81
```

Return Value
It returns float type perplexity score.

19.3.1.5 5. score(X, y = None)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix for sparse data of int, float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, just like in Scikit-learn.

Purpose
Calculate approximate log-likelihood as score for data X.

Likelihood is a measure of how plausible model parameters are, given the data. Taking a logarithm makes calculations easier. All values are negative: when $x < 1$, $\log(x) < 0$.

The idea is to search for the largest log-likelihood (good score will be close to 0).

For example,

```
print("score: %.2f" % (lda.score(csr_mat)))
```

Output

```
score: -2.18
```

Return Value
It returns a float value as likelihood score

19.3.1.6 6. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by LatentDirichletAllocation. It is used to get parameters and their values of LatentDirichletAllocation class.

For example,

```
print(lda.get_params())
```

Output

```
{'algorithm': 'original', 'batch_size': 128, 'doc_topic_prior': 0.3333333333333333,
'evaluate_every': -1, 'explore_iter': 0, 'learning_decay': 0.7, 'learning_method': 'batch',
'learning_offset': 10.0, 'max_doc_update_iter': 100, 'max_iter': 10,
'mean_change_tol': 0.001, 'n_components': 3, 'n_jobs': None, 'perp_tol': 0.1,
'random_state': None, 'topic_word_prior': 0.3333333333333333, 'total_samples': 1000000.0,
'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

19.3.1.7 7. set_params(**params)

Parameters
***params**: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by LatentDirichletAllocation, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(lda.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
lda.set_params(n_components = 4, algorithm = 'dp')
print("get parameters after setting:")
print(lda.get_params())
```

Output

```
get parameters before setting:
{'algorithm': 'original', 'batch_size': 128, 'doc_topic_prior': 0.3333333333333333,
'evaluate_every': -1, 'explore_iter': 0, 'learning_decay': 0.7, 'learning_method': 'batch',
'learning_offset': 10.0, 'max_doc_update_iter': 100, 'max_iter': 10,
'mean_change_tol': 0.001, 'n_components': 3, 'n_jobs': None, 'perp_tol': 0.1,
'random_state': None, 'topic_word_prior': 0.3333333333333333, 'total_samples': 1000000.0,
'verbose': 0}
get parameters after setting:
{'algorithm': 'dp', 'batch_size': 128, 'doc_topic_prior': 0.3333333333333333,
'evaluate_every': -1, 'explore_iter': 0, 'learning_decay': 0.7, 'learning_method': 'batch',
'learning_offset': 10.0, 'max_doc_update_iter': 100, 'max_iter': 10,
'mean_change_tol': 0.001, 'n_components': 4, 'n_jobs': None, 'perp_tol': 0.1,
'random_state': None, 'topic_word_prior': 0.3333333333333333, 'total_samples': 1000000.0,
'verbose': 0}
```

Return Value
It simply returns "self" reference.

19.3.1.8 8. fit_transform(X, y = None)

Parameters
X: A scipy sparse matrix or an instance of FrovedisCRSMatrix for sparse data of int, float or double (float64) type. It has shape **(n_samples, n_features)**.

y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It performs fit() and transform() on X and y (unused), and returns a transformed version of X.

For example,

```
# loading sample csr matrix data representing the input corpus-vocabulary count
test_corpus = [
'This is the first second third document.',
'This This one third document.'
]

# feature transformers CountVectorizer can be used for converting text to word count vectors.
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
csr_mat = vectorizer.fit_transform(test_corpus)

# transform input after fitting
from frovedis.mllib.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components = 3)
print(lda.fit_transform(csr_mat))
```

Output

```
[[0.04761905 0.61904762 0.47619048]
 [0.06666667 0.66666667 0.46666667]]
```

Like in fit() and transform(), we can also provide frovedis-like input in fit_transform() for faster computation.

For example,

```
# loading sample csr matrix data representing the input corpus-vocabulary count
test_corpus = [
'This is the first second third document.',
'This This one third document.'
]

# feature transformers CountVectorizer can be used for converting text to word count vectors.
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
csr_mat = vectorizer.fit_transform(test_corpus)

# Since "csr_mat" is scipy sparse data, we have created FrovedisCRSMatrix
# Also it only supports int64 as itype for input sparse data during training
from frovedis.matrix.crs import FrovedisCRSMatrix
cmat = FrovedisCRSMatrix(csr_mat, dtype = np.int32, itype = np.int64)

# transform input according to trained model
from frovedis.mllib.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components = 3)
lda.fit_transform(cmat).debug_print()
```

Output

```
matrix:
num_row = 2, num_col = 3
node 0
```

```
node = 0, local_num_row = 2, local_num_col = 3, val = 0.333333 0.619048 0.190476
0.266667 0.666667 0.266667
```

Return Value
- **When X is python native input:**
It returns a numpy ndarray of shape **(n_samples, n_components)** and double (float64) type values. This is a transformed array X.

- **When X is frovedis-like input:**
It returns a FrovedisRowmajorMatrix of shape **(n_samples, n_components)** and double (float64) type values. This is a transformed array X.

19.3.1.9 9. 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,

```
lda.load("./out/trained_lda_model")
```

Return Value
It simply returns "self" reference.

19.3.1.10 10. 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 (corpus_topic, metadata and word_topic) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the lda model
lda.save("./out/trained_lda_model")
```

This will save the lda model on the path "/out/trained_lda_model".
It would raise exception if the directory already exists with same name.

The 'trained_lda_model' directory has

trained_lda_model
|—**corpus_topic**
|—metadata
|—**word_topic**

Here, the **corpus_topic** directory contains the corpus topic count as sparse matrix in binary format.
The metadata file contains the information of model kind, input datatype used for trained model.
Here, the **word_topic** directory contains the word topic count as sparse matrix in binary format.

Return Value
It returns nothing.

19.3.1.11 11. release()

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

For example,

```
lda.release()
```

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

Return Value
It returns nothing.

19.3.1.12 12. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

19.4 SEE ALSO

- **Introduction to FrovedisCRSMatrix**

Chapter 20

Linear Regression

20.1 NAME

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

20.2 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)
```

20.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

20.3 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
```

During training, the input **X** is the training data and **y** is the corresponding label values 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.

20.3.1 Detailed Description

20.3.1.1 1. LinearRegression()

Parameters
fit_intercept: A boolean parameter specifying whether a constant(intercept) should be added to the decision function. (Default: True)
normalize: An unused parameter. (Default: False)
copy_X: An unused parameter. (Default: True)
n_jobs: An unused parameter. (Default: None)
max_iter: A positive integer parameter 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).
When 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 has 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 the 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.

20.3.1.2 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 containing the target values. It has shape **(n_samples,)**.
sample_weight: Python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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 frovedis-like inputs can be used to speed up the training time, especially 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 frovedis-like inputs
from frovedis.mllib.linear_model import LinearRegression
lr = LinearRegression(solver = 'sag').fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

20.3.1.3 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 sag 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]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
```

```
# 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.

20.3.1.4 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 containing the true values for X. It has shape **(n_samples,)**.
sample_weight: Python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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.

20.3.1.5 5. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by LinearRegression. It is used to get parameters and their values of LinearRegression class.

For example,

```
print(lr.get_params())
```

Output

```
{'copy_X': True, 'fit_intercept': True, 'lr_rate': 1e-08, 'max_iter': None, 'n_jobs': None,
'normalize': False, 'solver': 'lapack', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

20.3.1.6 6. set_params(**params)

Parameters
***params**: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by LinearRegression, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(lr.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
lr.set_params(solver='lbfgs', max_iter = 10000)
print("get parameters after setting:")
print(lr.get_params())
```

Output

```
get parameters before setting:
{'copy_X': True, 'fit_intercept': True, 'lr_rate': 1e-08, 'max_iter': None, 'n_jobs': None,
'normalize': False, 'solver': 'lapack', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
get parameters after setting:
{'copy_X': True, 'fit_intercept': True, 'lr_rate': 1e-08, 'max_iter': 10000, 'n_jobs': None,
'normalize': False, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
```

Return Value
It simply returns “self” reference.

20.3.1.7 7. 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” reference.

20.3.1.8 8. 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 (metadata and model) 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". It would raise exception if the directory already exists with same name.

The 'LNRModel' directory has

LNRModel
|——— metadata
|——— model

The metadata file contains the number of classes, model kind, input datatype used for trained model.
Here, the model file contains information about weights, intercept.

Return Value
It returns nothing.

20.3.1.9 9. 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
```

This output will be visible on server side. It displays the weights and intercept values on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

20.3.1.10 10. 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.

20.3.1.11 11. 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'.

20.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Lasso Regression in Frovedis**
- **Ridge Regression in Frovedis**

Chapter 21

LinearSVC

21.1 NAME

LinearSVC (Support Vector Classification) - A classification algorithm used to predict the binary output with hinge loss.

21.2 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)
```

21.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
load(fname, dtype = None)
save(fname)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

21.3 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}
```

During training, the input **X** is the training data and **y** are their corresponding label values (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. The frovedis interface is almost same as Scikit-learn LinearSVC (Support Vector Classification) 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 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.

21.3.1 Detailed Description

21.3.1.1 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. Unlike Scikit-learn, currently it supports only hinge loss. (Default: 'hinge')
dual: An unused parameter. (Default: True)
tol: A double (float64) parameter specifying the convergence tolerance value. It must be zero or a positive value. (Default: 1e-4)
C: A positive float parameter, also called as inverse of regularization strength. (Default: 1.0)
multi_class: An unused parameter. (Default: 'ovr')
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)
intercept_scaling: An unused parameter. (Default: 1)
class_weight: An unused parameter. (Default: 'None')

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 unused parameter. (Default: 'None')
max_iter: A positive integer parameter specifying maximum iteration count. (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" can 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.

21.3.1.2 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 containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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 LinearSVC 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, especially 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.

21.3.1.3 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]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix

# predicting on LinearSVC using frovedis-like 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,)**.

21.3.1.4 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 LinearSVC model
svm.load("./out/SVMModel")
```

Return Value
It simply returns "self" instance.

21.3.1.5 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 (label_map, metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

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

The SVMModel contains below directory structure:
SVMModel
|——label_map
|——metadata
|——model

'label_map' contains information about labels mapped with their encoded value.
'metadata' represents the detail about model_kind and datatype of training vector.
Here, the model file contains information about model_id, model_kind and datatype of training vector.

This will save the LinearSVC model on the path '/out/SVMModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

21.3.1.6 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 containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python narray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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.

21.3.1.7 7. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by LinearSVC. It is used to get parameters and their values of LinearSVC class.

For example,

```
print(svm.get_params())
```

Output

```
{'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True, 'intercept_scaling': 1,
'loss': 'hinge', 'lr_rate': 0.01, 'max_iter': 1000,'multi_class': 'ovr', 'penalty': 'l2',
'random_state': None, 'solver': 'sag','tol': 0.0001, 'verbose': 0,'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

21.3.1.8 8. set_params(**params)

Parameters
****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by LinearSVC, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(svm.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
svm.set_params( penalty='l1', dual=False)
print("Get parameters after setting:")
print(svm.get_params())
```

Output

```
Get parameters before setting:
{'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True,
'intercept_scaling': 1,'loss': 'hinge', 'lr_rate': 0.01, 'max_iter': 1000,
'multi_class': 'ovr','penalty': 'l2', 'random_state': None, 'solver': 'sag',
'tol': 0.0001, 'verbose': 0,'warm_start': False}
Get parameters after setting:
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True,
'intercept_scaling': 1,'loss': 'hinge', 'lr_rate': 0.01, 'max_iter': 1000,
'multi_class': 'ovr','penalty': 'l1', 'random_state': None, 'solver': 'sag',
'tol': 0.0001, 'verbose': 0,'warm_start': False}
```

Return Value
It simply returns "self" reference.

21.3.1.9 9. 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,

```
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
```

This output will be visible on server side. It displays the weights, intercept and threshold values on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

21.3.1.10 10. 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.

21.3.1.11 11. 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'.

21.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **LinearSVR in Frovedis**
- **SVC in Frovedis**

Chapter 22

LinearSVR

22.1 NAME

LinearSVR (Support Vector Regression) - A regression algorithm used to predict the binary output with L1 and L2 loss.

22.2 SYNOPSIS

```
class frovedis.mllib.svm.LinearSVR(epsilon=0.0, tol=1e-4, C=1.0, loss='epsilon_insensitive',
                                   fit_intercept=True, intercept_scaling=1, dual=True,
                                   verbose=0, random_state=None, max_iter=1000,
                                   penalty='l2', lr_rate=0.01, solver='sag',
                                   warm_start=False)
```

22.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
load(fname, dtype = None)done
save(fname)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

22.3 DESCRIPTION

Based on support vector machines method, the Linear SVR is an algorithm to solve the regression problems. The Linear SVR algorithm applies linear kernel method and it works well with large datasets. L1 or L2 method can be specified as a loss function in this model.

The model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target.

LinearSVR supports ZERO, L1 and L2 regularization to address the overfit problem.

Frovedis provides implementation of LinearSVR 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. The frovedis interface is almost same as Scikit-learn LinearSVR (Support Vector 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 LinearSVR 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.

22.3.1 Detailed Description

22.3.1.1 1. LinearSVR()

Parameters
epsilon: A zero or positive double (float64) parameter used in the epsilon-insensitive loss function. (Default: 0.0)
tol: A double (float64) parameter specifying the convergence tolerance value. It must be zero or a positive value. (Default: 1e-4)
C: A positive float parameter, it is inversely proportional to regularization strength. (Default: 1.0)
loss: A string object containing the loss function type to use.
Currently, frovedis supports 'epsilon_insensitive' and 'squared_epsilon_insensitive' loss function.
The 'epsilon-insensitive' loss (standard SVR) is the L1 loss, while the squared epsilon-insensitive loss ('squared_epsilon_insensitive') is the L2 loss. (Default: 'epsilon_insensitive')
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)
intercept_scaling: An unused parameter. (Default: 1)
dual: An unused parameter. (Default: True)
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 unused parameter. (Default: None)
max_iter: A positive integer parameter specifying maximum iteration count. (Default: 1000)
penalty: A string object containing the regularizer type to use. Currently none, l1 and l2 are supported by Frovedis. (Default: 'l2')
If it is None (not specified explicitly), it will be set as 'ZERO' regularization type.
lr_rate: A positive double (float64) value of parameter containing the learning rate. (Default: 0.01)
solver: A string object specifying the solver to use. (Default: 'sag')
Currenlty, it only supports 'sag'.
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)**.
intercept_: It is a python ndarray(float or double (float64) values depending on input matrix data type) and has shape **(1,)**.
n_iter_: An integer value used to get the actual iteration point at which the problem is converged.

Purpose
It initializes a LinearSVR object with the given parameters.

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

Return Value
It simply returns "self" reference.

22.3.1.2 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 containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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

For example,

```
# let a sample matrix be 'mat 'and labels be 'lbl'
mat = np.array([[ 5.5, 0.0, 6.5, 2.3],
                [ 0.0, 7.2, 0.0, 8.4],
                [ 8.5, 0.0, 6.5, 4.3],
                [ 0.0, 8.5, 0.1, 9.1]], dtype=np.float64)
lbl = np.array([1.1, 6.2,1.2, 5.9])


# fitting input matrix and label on LinearSVR object
from frovedis.mllib.svm import LinearSVR
svr = LinearSVR().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, especially when same data would be used for multiple executions.

For example,

```
# 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)
dlbl = FrovedisDvector(lbl)
```

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

Return Value
It simply returns "self" reference.

22.3.1.3 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,

```
svr.predict(mat)
```

Output:

```
[-181.66076961 -162.62098062 -166.05339001 ... -170.80953572 -169.6636383 -171.76112166]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix

# predicting on LinearSVR using frovedis-like input
svr.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[-181.66076961 -162.62098062 -166.05339001 ... -170.80953572 -169.6636383 -171.76112166]
```

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

22.3.1.4 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 LinearSVR model
svr.load("./out/SVRModel")
```

Return Value
It simply returns "self" reference.

22.3.1.5 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
svr.save("./out/SVRModel")
```

The SVRModel contains below directory structure:
SVRModel
|——metadata
|——model

'metadata' represents the detail about model_kind and datatype of training vector.
Here, the model file contains information about model_id, model_kind and datatype of training vector.

This will save the LinearSVR model on the path '/out/SVRModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

22.3.1.6 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 containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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
 svr.score(mat, lbl)
```

Output

```
0.97
```

Return Value
It returns an R2 score of float type.

22.3.1.7 7. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by LinearSVR. It is used to get parameters and their values of LinearSVR class.

For example,

```
print(svr.get_params())
```

Output

```
{'C': 1.0, 'dual': True, 'epsilon': 0.0, 'fit_intercept': True, 'intercept_scaling': 1,
'loss': 'epsilon_insensitive', 'lr_rate': 0.01, 'max_iter': 1000, 'penalty': 'l2',
'random_state': None, 'solver': 'sag', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

22.3.1.8 8. set_params(**params)

Parameters
*****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by LinearSVR, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(svr.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
svr.set_params( penalty = 'l1', dual = False)
print("Get parameters after setting:")
print(svr.get_params())
```

Output

```
Get parameters before setting:
{'C': 1.0, 'dual': True, 'epsilon': 0.0, 'fit_intercept': True, 'intercept_scaling': 1,
'loss': 'epsilon_insensitive', 'lr_rate': 0.01, 'max_iter': 1000, 'penalty': 'l2',
'random_state': None, 'solver': 'sag', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
Get parameters before setting:
{'C': 1.0, 'dual': False, 'epsilon': 0.0, 'fit_intercept': True, 'intercept_scaling': 1,
'loss': 'epsilon_insensitive', 'lr_rate': 0.01, 'max_iter': 1000, 'penalty': 'l1',
'random_state': None, 'solver': 'sag', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
```

Return Value
It simply returns “self” reference.

22.3.1.9 9. debug_print()

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

For example,

```
svr.debug_print()
```

Output:

```
-------- Weight Vector:: --------
0.406715 0.413736 0.440404 0.539082 0.536528 0.51577 0.157345 0.520861
0.513352 -0.341082 0.460518 -0.292171 0.433433
Intercept:: 1.00832
```

This output will be visible on server side. It displays the weights and intercept values on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

22.3.1.10 10. release()

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

For example,

```
svr.release()
```

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

Return Value
It returns nothing.

22.3.1.11 11. 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'.

22.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**

- **Introduction to FrovedisDvector**
- **LinearSVC in Frovedis**
- **SVC in Frovedis**

Chapter 23

Logistic Regression

23.1 NAME

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

23.2 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)
```

23.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
predict_proba(X)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

23.3 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))
```

During training, the input **X** is the training data and **y** is the corresponding label values (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.

23.3.1 Detailed Description

23.3.1.1 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 positive float parameter, it is the inverse of regularization strength. 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 unused parameter. (Default: 1)
class_weight: An unused parameter. (Default: None)
random_state: An unused parameter. (Default: None)
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" can handle L1, L2 or no penalty.
"lbfgs" can handle only L2 penalty.
max_iter: A positive integer value specifying maximum iteration count. (Default: 1000)
multi_class: A string object specifying the 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 unused parameter. (Default: 1)
l1_ratio: An unused parameter. (Default: None)
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.

23.3.1.2 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 containing target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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 specified 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 LogisticRegression 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 frovedis-like inputs can be used to speed up the training time, especially 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 frovedis-like inputs
from frovedis.mllib.linear_model import LogisticRegression
lr = LogisticRegression(solver = 'lbfgs').fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

23.3.1.3 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]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix

# 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.

23.3.1.4 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 probability 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]]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix

# finds the probability 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 an ndarray of float or double (float64) type and of shape **(n_samples, n_classes)** containing the prediction probability values.

23.3.1.5 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 containing true labels for X. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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.

23.3.1.6 6. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by LogisticRegression. It is used to get parameters and their values of LogisticRegression class.

For example,

```
print(lr.get_params())
```

Output

```
{'C': 100.0, 'class_weight': None, 'dual': False, 'fit_intercept': True,
'intercept_scaling': 1, 'l1_ratio': None, 'lr_rate': 0.01, 'max_iter': 1000,
'multi_class': 'auto', 'n_jobs': 1, 'penalty': 'l2', 'random_state': None,
'solver': 'sag', 'tol': 0.0001, 'use_shrink': False, 'verbose': 0,
'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

23.3.1.7 7. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by LogisticRegression, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(lr.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
lr.set_params(solver='lbfgs', max_iter = 10000)
print("get parameters after setting:")
print(lr.get_params())
```

Output

```
get parameters before setting:
{'C': 100.0, 'class_weight': None, 'dual': False, 'fit_intercept': True,
'intercept_scaling': 1, 'l1_ratio': None, 'lr_rate': 0.01, 'max_iter': 1000,
'multi_class': 'auto', 'n_jobs': 1, 'penalty': 'l2', 'random_state': None,
'solver': 'sag', 'tol': 0.0001, 'use_shrink': False, 'verbose': 0,
'warm_start': False}
get parameters after setting:
{'C': 100.0, 'class_weight': None, 'dual': False, 'fit_intercept': True,
'intercept_scaling': 1, 'l1_ratio': None, 'lr_rate': 0.01, 'max_iter': 10000,
'multi_class': 'auto', 'n_jobs': 1, 'penalty': 'l2', 'random_state': None,
'solver': 'lbfgs', 'tol': 0.0001, 'use_shrink': False, 'verbose': 0,
'warm_start': False}
```

Return Value
It simply returns “self” reference.

23.3.1.8 8. 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" reference.

23.3.1.9 9. 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 (label_map, metadata and model) 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'. It would raise exception if the directory already exists with same name.

The 'LRModel' directory has

LRModel
|—-label_map
|—-metadata
|—-**model**
    |——model
    |——type

'label_map' contains information about labels mapped with their encoded value.
The metadata file contains the number of classes, model kind, input datatype used for trained model.
Here, the **model** directory contains information about weights, intercept, threshold and thier datatype.

Return Value
It returns nothing.

23.3.1.10 10. 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
```

This output will be visible on server side. It displays the weights, intercept, etc. values on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

23.3.1.11 11. 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.

23.3.1.12 12. 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'.

23.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Linear Regression in Frovedis**

Chapter 24

Multinomial Naive Bayes

24.1 NAME

Multinomial Naive Bayes - One of the variations of Naive Bayes algorithm. It is a multinomial classification algorithm.

24.2 SYNOPSIS

```
class frovedis.mllib.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None,
                                                verbose=0)
```

24.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
predict_proba(X)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
debug_print()
load(fname, dtype = None)
save(fname)
release()
is_fitted()

24.3 DESCRIPTION

Naive Bayes classifier for multinomial models. The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification).

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 MultinomialNB 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 MultinomialNB 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.

24.3.1 Detailed Description

24.3.1.1 1. MultinomialNB()

Parameters
alpha: A positive double (float64) smoothing parameter. It must be greater than or equal to 1. (Default: 1.0)
fit_prior: A boolean parameter specifying whether to learn class prior probabilities or not. If False, a uniform prior will be used. (Default: True)
class_prior: A numpy ndarray of double (float64) type values and must be of shape **(n_classes,)**. It gives prior probabilities of the classes. (Default: None)
When it is None (not specified explicitly), the priors are adjusted according to the 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.

Attributes
class_log_prior_: A python ndarray of double (float64) type values and has shape **(n_classes,)**. It contains log probability of each class (smoothed).
feature_log_prob_: A python ndarray of double (float64) type values and has shape **(n_classes, n_features)**. It contains empirical log probability of features given a class.
class_count_: A python ndarray of double (float64) type values and has shape **(n_classes,)**. It contains the number of samples encountered for each class during fitting. This value is weighted by the sample weight when provided.
classes_: A python ndarray of double (float64) type values and has shape **(n_classes,)**. It contains the of unique labels given to the classifier during training.
feature_count_: A python ndarray of double (float64) type values and has shape **(n_classes, n_features)**. It contains the number of samples encountered for each (class, feature) during fitting. This value is weighted by the sample weight when provided.
coef_: A python ndarray of double (float64) type values.
- If 'classes_' is 2, then the shape **(1, n_features)**
- If 'classes_' is more then 2, then the shape is **(n_classes, n_features)**.

It mirrors 'feature_log_prob_' for interpreting MultinomialNB as a linear model.
intercept_: A python ndarray of double (float64) type values.
- If 'classes_' are 2, then the shape **(1,)**
- If 'classes_' is more then 2, then the shape is **(n_classes,)**.

It mirrors 'class_log_prior_' for interpreting MultinomialNB as a linear model.

Purpose
It initializes a MultinomialNB object with the given parameters.

Return Value
It simply returns "self" reference.

24.3.1.2 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 FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**.
y: Any python array-like object or an instance of FrovedisDvector containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample.

Purpose
It accepts the training matrix (X) with labels (y) and trains a MultinomialNB model.

For example,

```
# loading 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 MultinomialNB object
from frovedis.mllib.naive_bayes import MultinomialNB
mnb = MultinomialNB().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, especially when same data would be used for multiple executions.

For example,

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

# MultinomialNB with pre-constructed frovedis-like inputs
from frovedis.mllib.naive_bayes import MultinomialNB
mnb = MultinomialNB().fit(rmat, dlbl)
```

Return Value
It simply returns “self” reference.

24.3.1.3 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,

```
# predicting on MultinomialNB model
mnb.predict(mat)
```

Output

```
[1 1 1 1 1]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# predicting on MultinomialNB model using pre-constructed input
mnb.predict(rmat)
```

Output

```
[1 1 1 1 1]
```

Return Value
It returns a numpy array of long (int64) type and of shape **(n_samples,)** containing the predicted outputs.

24.3.1.4 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. It has shape **(n_samples, n_features)**.

Purpose
Perform classification on an array and return probability estimates for the test vector X.

It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server. Unlike sklearn, it performs the classification on an array and returns the probability estimates for the test feature matrix (X).

For example,

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

Output

```
[[1.00000000e+00 1.31173349e-73]
[1.00000000e+00 3.02940294e-46]
[1.00000000e+00 1.60233859e-37]
...
[9.99537526e-01 4.62474254e-04]
[1.00000000e+00 2.74000590e-29]
[3.87284789e-18 1.00000000e+00]]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# finds the probablity sample for each class in the model
mnb.predict_proba(rmat)
```

Output

```
[[1.00000000e+00 1.31173349e-73]
[1.00000000e+00 3.02940294e-46]
[1.00000000e+00 1.60233859e-37]
...
[9.99537526e-01 4.62474254e-04]
[1.00000000e+00 2.74000590e-29]
[3.87284789e-18 1.00000000e+00]]
```

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

24.3.1.5 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. It has shape **(n_samples, n_features)**.
y: Any python array-like object containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), 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,

```
mnb.score(mat,lbl)
```

Output

```
0.89
```

Return Value
It returns an accuracy score of double (float64) type.

24.3.1.6 6. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by MultinomialNB. It is used to get parameters and their values of MultinomialNB class.

For example,

```
print(mnb.get_params())
```

Output

```
{'alpha': 1.0, 'class_prior': None, 'fit_prior': True, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

24.3.1.7 7. set_params(**params)

Parameters
****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by MultinomialNB, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(mnb.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
mnb.set_params(fit_prior = False)
print("Get parameters after setting:")
print(mnb.get_params())
```

Output

```
Get parameters before setting:
{'alpha': 1.0, 'class_prior': None, 'fit_prior': True, 'verbose': 0}
Get parameters after setting:
{'alpha': 1.0, 'class_prior': None, 'fit_prior': False, 'verbose': 0}
```

Return Value
It simply returns "self" reference.

24.3.1.8 8. load(fname, dtype = None)

Parameters
fname: A string object containing the name of the file having model information such as theta, cls_count, feature_count, label, pi, type 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 stored previously from the specified file (having little-endian binary data).

For example,

```
mnb.load("./out/MNBModel", dtype = np.float64)
```

Return Value
It simply returns "self" reference.

24.3.1.9 9. 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(label_map, metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the MultinomailNB model
mnb.save("./out/MNBModel")
```

The MNBModel contains below directory structure:
MNBModel
|——label_map
|——metadata
|——**model**
|——cls_count
|——feature_count
|——label
|——pi
|——**theta**
|——type

'label_map' contains information about labels mapped with their encoded value.
'metadata' represents the detail about model_kind and datatype of training vector.
Here, the **model** directory contains information about cls_count, feature_count, label, pi, **theta** and type.

This will save the naive bayes model on the path '/out/MNBModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

24.3.1.10 10. debug_print()

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

For example,

```
mnb.debug_print()
```

Output

```
model_type: multinomial
binarize: 0
feature_count: 3702.12 4580.24 24457.5 207416 21.8145  ... 65.2141 59.3469 26.5766 96.4778 28.3608
theta: node = 0, local_num_row = 2, local_num_col = 30, val = -5.08709 -4.8743 -3.19929 -1.06154
-10.1766
...
-8.83939 -8.93218 -9.71532 -8.45266 -9.65263
pi: -0.987294 -0.466145
label: 0 1
class count: 212 357
```

This output will be visible on server side. It displays the target model information like model_type, binarize, feature_count, theta, pi, etc. values on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

24.3.1.11 11. release()

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

For example,

```
mnb.release()
```

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

Return Value
It returns nothing.

24.3.1.12 12. 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 naive bayes model first.

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

24.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Bernoulli Naive Bayes in Frovedis**

Chapter 25

NearestNeighbors

25.1 NAME

NearestNeighbors - Unsupervised learner for implementing neighbor searches.

25.2 SYNOPSIS

```
class frovedis.mllib.neighbors.NearestNeighbors(n_neighbors=5, radius=1.0, algorithm='auto',
                                                leaf_size=30, metric='euclidean', p=2,
                                                metric_params=None, n_jobs=None, verbose=0,
                                                chunk_size=1.0, batch_fraction=None)
```

25.2.1 Public Member Functions

fit(X, y = None)
kneighbors(X = None, n_neighbors = None, return_distance = True)
kneighbors_graph(X = None, n_neighbors = None, mode = 'connectivity')
radius_neighbors(X = None, radius = None, return_distance = True)
radius_neighbors_graph(X = None, radius = None, mode = 'connectivity')
get_params(deep = True)
set_params(**params)
save(fname)
load(fname)
debug_print()
release()
is_fitted()

25.3 DESCRIPTION

Unsupervised nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering.

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. The number of samples can be a user-defined

constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). In general, the distance can be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply "remember" all of its training data.

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 NearestNeighbors 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 NearestNeighbors 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 operations like kneighbors() will be required 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.

25.3.1 Detailed Description

25.3.1.1 1. NearestNeighbors()

Parameters
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for 'kneighbors' queries. It must be within the range of 0 and n_samples. (Default: 5)
radius: A positive float parameter, specifying the range of parameter space to use by default for 'radius_neighbors' queries. (Default: 1.0)
algorithm: A string object parameter, specifying the algorithm used to compute the nearest neighbors. (Default: 'auto')
When it is 'auto', it will be set as 'brute'(brute-force search approach). Unlike Scikit-learn, currently it supports only 'brute'.
leaf_size: An unsed parameter. (Default: 30)
metric: A string object parameter, specifying the distance metric to use for the tree. (Default: 'euclidean')
Currenlty it only supports 'euclidean', 'seuclidean' and 'cosine' distance.
p: An unused parameter. (Default: 2)
metric_params: An unsed parameter. (Default: None)
n_jobs: An unsed parameter. (Default: None)
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.
chunk_size: A positive float parameter, specifying the amount of data (in megabytes) to be processed in one time. (Default: 1.0)
batch_fraction: A positive double (float64) parameter used to calculate the batches of specific size. These batches are used to construct the distance matrix. It must be within the range of 0.0 to 1.0. (Default: None)
When it is None (not specified explicitly), it will be set as np.finfo(np.float64).max value.

Purpose
It initializes a NearestNeighbors object with the given parameters.

The parameters: "leaf_size", "p", "metric_params" and "n_jobs" are simply kept in to make the interface uniform to the Scikit-learn NearestNeighbors module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

25.3.1.2 2. fit(X, y = 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
Fit the model using X as training data

For example,

```
# loading sample data
samples = np.loadtxt("./input/knn_data.txt", dtype = np.float64)

# fitting input data on NearestNeighbors object
from frovedis.mllib.neighbors import NearestNeighbors
knn = NearestNeighbors(n_neighbors = 3, radius = 2.0, algorithm = 'brute',
                       metric = 'euclidean')
knn.fit(samples)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading sample data
samples = np.loadtxt("./input/knn_data.txt", dtype = np.float64)

# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# fitting input data on NearestNeighbors object
from frovedis.mllib.neighbors import NearestNeighbors
knn = NearestNeighbors(n_neighbors = 3, radius = 2.0, algorithm = 'brute',
                       metric = 'euclidean')
knn.fit(rmat)
```

Return Value
It simply returns "self" reference.

25.3.1.3 3. kneighbors(X = None, n_neighbors = None, return_distance = True)

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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where 'n_queries' is the number of rows in the test data. (Default: None)
When it is None (not specified explicitly), it will be training data (X) used as input in fit().
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for 'kneighbors' queries. It must be within the range of 0 and n_queries. (Default: None)
When it is None (not specified explicitly), it will be 'n_neighbors' value used during NearestNeighbors object creation.

return_distance: A boolean parameter specifying whether or not to return the distances. (Default: True) If set to False, it will not return distances. Then, only indices are returned by this method.

Purpose
It finds the k-Neighbors of a point and returns the indices of neighbors and distances to the neighbors of each point.

For example,

```
distances, indices = knn.kneighbors(samples)
print('distances')
print(distances)
print('indices')
print(indices)
```

Output

```
distances
[[0.         1.         2.23606798]
 [0.         1.         1.41421356]
 [0.         1.41421356 2.23606798]
 [0.         1.         2.23606798]
 [0.         1.         1.41421356]
 [0.         1.41421356 2.23606798]]
 [[0.         1.         2.23606798]
indices
 [[0 1 2]
 [1 0 2]
 [2 1 0]
 [3 4 5]
 [4 3 5]
 [5 4 3]]
```

Like in fit(), frovedis-like input can be used to speed-up the computation of indices and distances at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

distances, indices = knn.kneighbors(rmat)

# Here FrovedisRowmajorMatrix().debug_print() is used
print('distances')
distances.debug_print()

# Here FrovedisRowmajorMatrix().debug_print() is used
print('indices')
indices.debug_print()
```

Output

```
distances
matrix:
num_row = 6, num_col = 3
node 0
```

```
node = 0, local_num_row = 6, local_num_col = 3, val = 0 1 2.23607 0 1 1.41421 0 1.41421
2.23607 0 1 2.23607 0 1 1.41421 0 1.41421 2.23607
indices
matrix:
num_row = 6, num_col = 3
node 0
node = 0, local_num_row = 6, local_num_col = 3, val = 0 1 2 1 0 2 2 1 0 3 4 5 4 3 5 5 4 3
```

Return Value
1. **When test data and training data used by fitted model are python native input:**
- ***distances***: A numpy array of float or double(float64) type values. It has shape **(n_queries, n_neighbors)**, where 'n_queries' is the number of rows in the test data. It is only returned by kneighbors() if return_distance = True.
- ***indices***: A numpy array of int64 type values. It has shape **(n_queries, n_neighbors)**, where 'n_queries' is the number of rows in the test data.
2. **When either test data or training data used by fitted model is frovedis-like input:**
- ***distances***: A FrovedisRowmajorMatrix of float or double(float64) type values. It has shape **(n_queries, n_neighbors)**, where 'n_queries' is the number of rows in the test data. It is only returned by kneighbors() if return_distance = True.
- ***indices***: A FrovedisRowmajorMatrix of int64 type values. It has shape **(n_queries, n_neighbors)**, where 'n_queries' is the number of rows in the test data.

25.3.1.4 4. kneighbors_graph(X = None, n_neighbors = None, mode = 'connectivity')

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 of float or double (float64) type. It has shape **(n_queries, n_features)**, where 'n_queries' is the number of rows in the test data. (Default: None)
When it is None (not specified explicitly), it will be training data (X) used as input in fit().
n_neighbors: A positive integer parameter, specifying the number of neighbors to use by default for 'kneighbors' queries. It must be within the range of 0 and n_queries. (Default: None)
When it is None (not specified explicitly), it will be 'n_neighbors' value used during NearestNeighbors object creation.
mode: A string object parameter which can be either 'connectivity' or 'distance'. It specifies the type of returned matrix.
For 'connectivity', it will return the connectivity matrix with ones and zeros, whereas for 'distance', the edges are euclidean distance between points. Type of distance depends on the selected 'metric' value in NearestNeighbors class. (Default: 'connectivity')

Purpose
It computes the (weighted) graph of k-Neighbors for points in X.

For example, when mode = 'connectivity'

```
# Here 'mode = connectivity' by default
graph = knn.kneighbors_graph(samples)
print('kneighbors graph')
print(graph)
```

Output

```
kneighbors graph
(0, 0)        1.0
(0, 1)        1.0
(0, 2)        1.0
(1, 1)        1.0
```

```
(1, 0)        1.0
(1, 2)        1.0
(2, 2)        1.0
(2, 1)        1.0
(2, 0)        1.0
(3, 3)        1.0
(3, 4)        1.0
(3, 5)        1.0
(4, 4)        1.0
(4, 3)        1.0
(4, 5)        1.0
(5, 5)        1.0
(5, 4)        1.0
(5, 3)        1.0
```

For example, when mode = 'distance'

```
# Here 'mode = distance'
graph = knn.kneighbors_graph(samples, mode = 'distance')
print('kneighbors graph')
print(graph)
```

Output

```
kneighbors graph
(0, 0)        0.0
(0, 1)        1.0
(0, 2)        2.23606797749979
(1, 1)        0.0
(1, 0)        1.0
(1, 2)        1.4142135623730951
(2, 2)        0.0
(2, 1)        1.4142135623730951
(2, 0)        2.23606797749979
(3, 3)        0.0
(3, 4)        1.0
(3, 5)        2.23606797749979
(4, 4)        0.0
(4, 3)        1.0
(4, 5)        1.4142135623730951
(5, 5)        0.0
(5, 4)        1.4142135623730951
(5, 3)        2.23606797749979
```

Like in fit(), frovedis-like input can be used to speed-up the graph construction at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# Here 'mode = connectivity' by default
graph = knn.kneighbors_graph(rmat)
print('kneighbors graph')
```

```
# Here FrovedisCRSMatrix().debug_print() is used
graph.debug_print()
```

Output

```
graph
Active Elements:  18
matrix:
num_row = 6, num_col = 6
node 0
local_num_row = 6, local_num_col = 6
val : 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
idx : 0 1 2 1 0 2 2 1 0 3 4 5 4 3 5 5 4 3
off : 0 3 6 9 12 15 18
```

Return Value
- **When test data and training data used by fitted model are python native input :**
It returns a scipy sparse csr matrix of float or double (float64) type values. It has shape **(n_queries, n_samples_fit)**, where 'n_queries' is the number of rows in the test data and 'n_samples_fit' is the number of samples in the fitted data.
- **When either test data or training data used by fitted model is frovedis-like input:**
It returns a FrovedisCRSMatrix of float or double(float64) type values. It has shape **(n_queries, n_samples_fit)**, where 'n_queries' is the number of rows in the test data and 'n_samples_fit' is the number of samples in the fitted data.

25.3.1.5 5. radius_neighbors(X = None, radius = None, return_distance = True)

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 of float or double (float64) type. It has shape **(n_samples, n_features)**. (Default: None)
When it is None (not specified explicitly), it will be training data (X) used as input in fit().
radius: A positive float parameter, specifying the limiting distance of neighbors to return. (Default: None)
When it is None (not specified explicitly), it will be 'radius' value used in NearestNeighbors object creation.
return_distance: A boolean parameter specifying whether or not to return the distances. (Default: True)
If set to False, it will not return distances. Then, only indices are returned by this method.

Purpose
It finds the neighbors within a given radius of a point or points and returns indices and distances to the neighbors of each point.

For example,

```
dist, ind = knn.radius_neighbors(samples)
print('radius neighbors')
for i in range(len(dist)):
    print(dist[i])
    print(ind[i])
```

Output

```
radius neighbors
[0. 1.]
[0 1]
[1.         0.         1.41421356]
[0 1 2]
[1.41421356 0.        ]
```

```
[1 2]
[0. 1.]
[3 4]
[1.         0.         1.41421356]
[3 4 5]
[1.41421356 0.        ]
[4 5]
```

Like in fit(), frovedis-like input can be used to speed-up the computation of indices and distances at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

rad_nbs = knn.radius_neighbors(rmat)
print('radius neighbors')

# Here FrovedisCRSMatrix().debug_print() is used
rad_nbs.debug_print()
```

Output

```
radius neighbors
Active Elements:  14
matrix:
num_row = 6, num_col = 6
node 0
local_num_row = 6, local_num_col = 6
val : 0 1 1 0 1.41421 1.41421 0 0 1 1 0 1.41421 1.41421 0
idx : 0 1 0 1 2 1 2 3 4 3 4 5 4 5
off : 0 2 5 7 9 12 14
```

Return Value
1. **When test data and training data used by fitted model are python native input :**
- ***distance***: A python list of float or double(float64) type values and has length **n_samples**. It is only returned by radius_neighbors() if return_distance = True.
- ***indices***: A python list of float or double(float64) type values and has length **n_samples**.
2. **When either test data or training data used by fitted model is frovedis-like input:**
It returns a FrovedisCRSMatrix of shape **(n_samples, n_samples_fit)**, where 'n_samples_fit' is the number of samples in the fitted data.

25.3.1.6 6. radius_neighbors_graph(X = None, radius = None, mode = 'connectivity')

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 of float or double (float64) type. It has shape **(n_samples, n_features)**. (Default: None)
When it is None (not specified explicitly), it will be training data (X) used as input in fit().
radius: A positive float parameter, specifying the limiting distance of neighbors to return. (Default: None)
When it is None (not specified explicitly), it will be 'radius' value used in NearestNeighbors object creation.
mode: A string object parameter which can be either 'connectivity' or 'distance'. It specifies the type of returned matrix.

For ‘connectivity’, it will return the connectivity matrix with ones and zeros, whereas for ‘distance’, the edges are euclidean distance between points. Type of distance depends on the selected ‘metric’ value in NearestNeighbors class. (Default: ‘connectivity’)

Purpose
It computes the (weighted) graph of Neighbors for points in X.

For example, when mode = ‘connectivity’

```
# Here 'mode = connectivity' by default
rad_graph = knn.radius_neighbors_graph(samples)
print('radius neighbors graph')
print(rad_graph)
```

Output

```
radius neighbors graph
(0, 0)        1.0
(0, 1)        1.0
(1, 0)        1.0
(1, 1)        1.0
(1, 2)        1.0
(2, 1)        1.0
(2, 2)        1.0
(3, 3)        1.0
(3, 4)        1.0
(4, 3)        1.0
(4, 4)        1.0
(4, 5)        1.0
(5, 4)        1.0
(5, 5)        1.0
```

For example, when mode = ‘distance’

```
# Here 'mode = distance'
rad_graph = knn.radius_neighbors_graph(samples, mode = 'distance')
print('radius neighbors graph')
print(rad_graph)
```

Output

```
radius neighbors graph
(0, 0)        0.0
(0, 1)        1.0
(1, 0)        1.0
(1, 1)        0.0
(1, 2)        1.4142135623730951
(2, 1)        1.4142135623730951
(2, 2)        0.0
(3, 3)        0.0
(3, 4)        1.0
(4, 3)        1.0
(4, 4)        0.0
(4, 5)        1.4142135623730951
(5, 4)        1.4142135623730951
(5, 5)        0.0
```

Like in fit(), frovedis-like input can be used to speed-up the computation of graph construction at server side.

For example,

```
# Since "samples" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(samples)

# Here 'mode = connectivity' by default
rad_graph = knn.radius_neighbors_graph(samples)
print('radius neighbors graph')

# Here FrovedisCRSMatrix().debug_print() is used
rad_graph.debug_print()
```

Output

```
radius neighbors graph
Active Elements:  14
matrix:
num_row = 6, num_col = 6
node 0
local_num_row = 6, local_num_col = 6
val : 1 1 1 1 1 1 1 1 1 1 1 1 1 1
idx : 0 1 0 1 2 1 2 3 4 3 4 5 4 5
off : 0 2 5 7 9 12 14
```

Return Value
- **When test data and training data used by fitted model are python native input :**
It returns a scipy sparse csr matrix of float or double(float64) type values. It has shape **(n_queries, n_samples_fit)**, where 'n_queries' is the number of rows in the test data and 'n_samples_fit' is the number of samples in the fitted data.
- **When either test data or training data used by fitted model is frovedis-like input:**
It returns a FrovedisCRSMatrix of float or double(float64) type values. It has shape **(n_queries, n_samples_fit)**, where 'n_queries' is the number of rows in the test data and 'n_samples_fit' is the number of samples in the fitted data.

25.3.1.7 7. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by NearestNeighbors. It is used to get parameters and their values of NearestNeighbors class.

For example,

```
print(knn.get_params())
```

Output

```
{'algorithm': 'brute', 'batch_fraction': 1.7976931348623157e+308, 'chunk_size': 1.0,
'leaf_size': 30, 'metric': 'euclidean', 'metric_params': None, 'n_jobs': None,
'n_neighbors': 3, 'p': 2, 'radius': 2.0, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

25.3.1.8 8. set_params(**params)

Parameters
***params**: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by NearestNeighbors, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(knn.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
knn.set_params(n_neighbors = 4, radius = 1.0)
print("get parameters after setting:")
print(knn.get_params())
```

Output

```
get parameters before setting:
{'algorithm': 'brute', 'batch_fraction': 1.7976931348623157e+308, 'chunk_size': 1.0,
'leaf_size': 30, 'metric': 'euclidean', 'metric_params': None, 'n_jobs': None,
'n_neighbors': 3, 'p': 2, 'radius': 2.0, 'verbose': 0}
get parameters after setting:
{'algorithm': 'brute', 'batch_fraction': 1.7976931348623157e+308, 'chunk_size': 1.0,
'leaf_size': 30, 'metric': 'euclidean', 'metric_params': None, 'n_jobs': None,
'n_neighbors': 4, 'p': 2, 'radius': 1.0, 'verbose': 0}
```

Return Value
It simply returns “self” reference.

25.3.1.9 9. save(fname)

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

Purpose
Currently this method is not supported for NearestNeighbors. It is simply kept in NearestNeighbors module to maintain uniform interface for all estimators in frovedis.

Return Value
It simply raises an AttributeError.

25.3.1.10 10. load(fname)

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

Purpose
Currently this method is not supported for NearestNeighbors. It is simply kept in NearestNeighbors module to maintain uniform interface for all estimators in frovedis.

Return Value
It simply raises an AttributeError.

25.3.1.11 11. debug_print()

Purpose
Currently this method is not supported for NearestNeighbors. It is simply kept in NearestNeighbors module to maintain uniform interface for all estimators in frovedis.

Return Value
It simply raises an AttributeError.

25.3.1.12 12. release()

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

For example,

```
knn.release()
```

This will remove the trained model, model-id present on server, along with releasing server side memory.

Return Value
It returns nothing.

25.3.1.13 13. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

25.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **KNeighbors Classifier in Frovedis**
- **KNeighbors Regressor in Frovedis**

Chapter 26

Principal Component Analysis

26.1 NAME

PCA - It's full form is Principal Component Analysis. It is an unsupervised learning algorithm that is used for dimensionality reduction in machine learning.

26.2 SYNOPSIS

```
frovedis.mllib.decomposition.pca(n_components=None, copy=True, whiten=False,
                                 svd_solver='auto', tol=0.0, iterated_power='auto',
                                 random_state=None)
```

26.2.1 Public Member Functions

fit(X, y = None)
fit_transform(X)
get_params(deep = True)
inverse_transform(X)
is_fitted()
load(path, dtype)
load_binary(path, dtype)
release()
save(path)
save_binary(path)
set_params(**params)
transform(X)

26.3 DESCRIPTION

It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

For dimensionality reduction, it uses Singular Value Decomposition of the data. The input data is centered but not scaled for each feature before applying the SVD.

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 PCA 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 PCA 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 transform-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.

Like Scikit-learn, PCA in frovedis only supports dense input data. However, when sparse input is provided in frovedis, internally, it will be converted into frovedis-like dense input before sending it to frovedis server.

26.3.1 Detailed Description

26.3.1.1 1. PCA()

Parameters
n_components: It accepts a positive integer value as parameter that specifies the number of components to keep. It must be in range **[1, min(n_samples_, n_features_) - 1)**. (Default: None)
When it is None (not specified explicitly), it wil be set as **min(n_samples_, n_features_) - 1**.
copy: It is a boolean parameter that specifies whether input data passed is overwritten or not during fitting. (Default: True)
When it is False (specified explicitly), data passed to fit() is overwritten and running fit(X).transform(X) will not yield the expected results, so fit_transform(X) is to be used instead.
whiten: It is a boolean parameter. It removes some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators. (Default: False)
When it is True (specified explicitly), the **components_** vectors are multiplied by the square root of **n_samples** and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.
svd_solver: It accepts a string value as parameter that specifies which solver to use. **Currently it supports only 'arpack' implementation of svd.** (Default: 'auto')
When it is auto, then **svd_solver = 'arpack'.**
tol: This is an unused parameter. (Default: 0.0)
iterated_power: This is an unused parameter. (Default: 'auto')
random_state: This is an unused parameter. (Default: None)

Attributes
components_: It is a numpy ndarray of shape **(n_components, n_features)**. It specifies the principal axes in feature space, representing the directions of maximum variance in the data.
explained_variance_: It is a numpy ndarray of shape **(n_components,)**. It specifies the variance of the training samples transformed by a projection to each component.
explained_variance_ratio_: It is a numpy ndarray of shape **(n_components,)**. It specifies the percentage of variance explained by each of the selected components. If **'n_components'** is not set, then all components are stored and the sum of the ratios is equal to 1.0.
mean_: It is a numpy ndarray of shape **(n_features,)**. It specifies the per-feature empirical mean, estimated from the training set. It is equal to **X.mean(axis=0)**.

n_components_: It is a positive integer value that specifies the estimated number of components.
n_features_: It is a positive integer value that specifies the number of features in the training data (X).
n_samples_: It is a positive integer value that specifies the number of samples in the training data (X).
noise_varaince_: It is a float (float32) value that specifies the estimated noise covariance. It is required to compute the estimated data covariance and score samples. It is equal to the **average of (min(n_features, n_samples) - n_components)** smallest eigenvalues of the covariance matrix of input matrix (X).
singular_values_: It is a numpy ndarray of shape **(n_components,)** that specifies the singular values corresponding to each of the selected components.

Purpose
It initializes a PCA object with the given parameters.

The parameters: "tol", "iterated_power" and "random_state" are simply kept in to make the interface uniform to the Scikit-learn PCA module. They are not used in frovedis implementation internally.

Return Value
It simply returns "self" reference.

26.3.1.2 2. fit(X, y = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It will fit the model with input matrix (X).

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[1.0, 0.0, 7.0, 0.0, 0.0],
                 [2.0, 0.0, 3.0, 4.0, 5.0],
                 [4.0, 0.0, 0.0, 6.0, 7.0]],
                dtype = np.float64)

# fitting input matrix on PCA object
from frovedis.mllib.decomposition import PCA
pca = PCA().fit(mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[1.0, 0.0, 7.0, 0.0, 0.0],
                 [2.0, 0.0, 3.0, 4.0, 5.0],
                 [4.0, 0.0, 0.0, 6.0, 7.0]],
                dtype = np.float64)

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
```

```
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# PCA with pre-constructed frovedis-like inputs
from frovedis.mllib.decomposition import PCA
pca = PCA().fit(rmat)
```

Return Value
It simply returns "self" reference.

26.3.1.3 3. fit_transform(X, y = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It will fit the model with input matrix (X) and apply the dimensionality reduction on input matrix (X).

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[1.0, 0.0, 7.0, 0.0, 0.0],
                 [2.0, 0.0, 3.0, 4.0, 5.0],
                 [4.0, 0.0, 0.0, 6.0, 7.0]],
                dtype = np.float64)

# fitting input matrix on PCA object and perform transform
from frovedis.mllib.decomposition import PCA
pca = PCA()
print(pca.fit_transform(mat))
```

Output

```
[[-6.50037234  0.28016556]
 [ 1.0441383  -0.75923769]
 [ 5.45623404  0.47907213]]
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[1.0, 0.0, 7.0, 0.0, 0.0],
                 [2.0, 0.0, 3.0, 4.0, 5.0],
                 [4.0, 0.0, 0.0, 6.0, 7.0]],
                dtype = np.float64)

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
```

```
rmat = FrovedisRowmajorMatrix(mat)

# Fitting PCA with pre-constructed frovedis-like inputs and perform transform
from frovedis.mllib.decomposition import PCA
pca = PCA()
print(pca.fit_transform(rmat))
```

Output

```
[[-6.50037234  0.28016556]
 [ 1.0441383  -0.75923769]
 [ 5.45623404  0.47907213]]
```

Return Value
For both frovedis-like input and python input, it returns a numpy ndarray of shape **(n_samples, n_components)** and double (float64) type values. It contains transformed values.

26.3.1.4 4. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by PCA. It is used to get parameters and their values of PCA class.

For example,

```
print(pca.get_params())
```

Output

```
{'copy': True, 'iterated_power': 'auto', 'n_components': 2, 'random_state': None,
'svd_solver': 'arpack', 'tol': 0.0, 'whiten': False}
```

Return Value
A dictionary of parameter names mapped to their values.

26.3.1.5 5. inverse_transform(X, y = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It transforms data back to its original space.

In other words, it returns an input matrix (X_original) whose transform would be input matrix (X).

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[1.0, 0.0, 7.0, 0.0, 0.0],
                 [2.0, 0.0, 3.0, 4.0, 5.0],
```

```
                 [4.0, 0.0, 0.0, 6.0, 7.0]],
                dtype = np.float64)

# fitting input matrix on PCA object and perform transform
from frovedis.mllib.decomposition import PCA
pca = PCA().fit(mat)
X1 = pca.transform(mat)

# inverse_transform() demo to get original input data
print(pca.inverse_transform(X1))
```

Output

```
[[ 1.00000000e+00  0.00000000e+00  7.00000000e+00  5.77315973e-15
   1.24344979e-14]
 [ 2.00000000e+00  0.00000000e+00  3.00000000e+00  4.00000000e+00
   5.00000000e+00]
 [ 4.00000000e+00  0.00000000e+00 -7.99360578e-15  6.00000000e+00
   7.00000000e+00]]
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[1.0, 0.0, 7.0, 0.0, 0.0],
                 [2.0, 0.0, 3.0, 4.0, 5.0],
                 [4.0, 0.0, 0.0, 6.0, 7.0]],
                dtype = np.float64)

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# PCA with pre-constructed frovedis-like inputs and perform transform
from frovedis.mllib.decomposition import PCA
pca = PCA().fit(rmat)
X1 = pca.transform(rmat)

# inverse_transform() demo to get original frovedis-like input data
X2 = pca.inverse_transform(X1)
X2.debug_print()
```

Output

```
matrix:
num_row = 3, num_col = 5
node 0
node = 0, local_num_row = 3, local_num_col = 5, val = 1 0 7 5.77316e-15 1.24345e-14
2 0 3 4 5 4 0 -7.99361e-15 6 7
```

Return Value

- **When X is python native input:**

It returns a numpy ndarray of shape **(n_samples, n_components)** and double (float64) type values. It contains original input data values.

- **When X is frovedis-like input:**
  It returns a FrovedisRowmajorMatrix instance of shape **(n_samples, n_components)** and double (float64) type values. It contains original input data values.

26.3.1.6 6. is_fitted()

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

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

26.3.1.7 7. load(path, dtype)

Parameters
path: A string object containing the name of the file having model information to be loaded.
dtype: It is the data-type of the loaded model with trainig data samples. Currently, expected input data-type is either float (float32) or double (float64).

Purpose
It loads the model from the specified file.

For example,

```
pca.load("./out/PCAModel", dtype = np.float64)
```

Return Value
It simply returns "self" reference.

26.3.1.8 8. load_binary(path, dtype)

Parameters
path: A string object containing the name of the binary file having model information to be loaded.
dtype: It is the data-type of the loaded model with trainig data samples. Currently, expected input data-type is double (float64).

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

For example,

```
pca.load("./out/PCA_BinaryModel", dtype = np.float64)
```

Return Value
It simply returns "self" reference.

26.3.1.9 9. save(path)

Parameters
path: 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 (components, mean, score, etc.) in the specified file. Otherwise, it throws an exception.

For example,

```
# To save the pca model
pca.save("./out/PCAModel")
```

This will save the pca model on the path '/out/PCAModel'. It would raise exception if the directory already exists with same name.

The 'PCAModel' directory has

PCAModel
|—-components
|—-mean
|—-score
|—-singular_values
|—-variance
|—-variance_ratio

The saved model contains the after-fitted attribute values.

Return Value
It returns nothing.

26.3.1.10 10. save_binary(path)

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

Purpose
On success, it writes the model information (components, mean, score, etc.) in binary format in the specified file. Otherwise, it throws an exception.

For example,

```
# To save the pca binary model
pca.save("./out/PCA_BinaryModel")
```

This will save the pca model on the path '/out/PCA_BinaryModel'. It would raise exception if the directory already exists with same name.

The 'PCA_BinaryModel' directory has

PCA_BinaryModel
|—-**components**
|—-mean
|—-**score**
|—-singular_values
|—-variance
|—-variance_ratio

The saved binary model contains the after-fitted attribute values.

Return Value
It returns nothing.

26.3.1.11 11. set_params(**params)

Parameters
***params**: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by PCA, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(pca.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
pca.set_params(whiten=True)
print("get parameters after setting:")
print(pca.get_params())
```

Output

```
get parameters before setting:
{'copy': True, 'iterated_power': 'auto', 'n_components': 2, 'random_state': None,
'svd_solver': 'arpack', 'tol': 0.0, 'whiten': False}
get parameters after setting:
{'copy': True, 'iterated_power': 'auto', 'n_components': 2, 'random_state': None,
'svd_solver': 'arpack', 'tol': 0.0, 'whiten': True}
```

Return Value
It simply returns "self" reference.

26.3.1.12 12. transform(X, y = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It applies the dimensionality reduction on input matrix (X).

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[1.0, 0.0, 7.0, 0.0, 0.0],
                 [2.0, 0.0, 3.0, 4.0, 5.0],
                 [4.0, 0.0, 0.0, 6.0, 7.0]],
                dtype = np.float64)

# fitting input matrix on PCA object and perform transform
from frovedis.mllib.decomposition import PCA
pca = PCA().fit(mat)
print(pca.transform(mat))
```

Output

```
[[-6.50037234  0.28016556]
 [ 1.0441383  -0.75923769]
 [ 5.45623404  0.47907213]]
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[1.0, 0.0, 7.0, 0.0, 0.0],
                 [2.0, 0.0, 3.0, 4.0, 5.0],
                 [4.0, 0.0, 0.0, 6.0, 7.0]],
                dtype = np.float64)

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# Fitting PCA with pre-constructed frovedis-like inputs and perform transform
from frovedis.mllib.decomposition import PCA
pca = PCA().fit(rmat)
X1 = pca.transform(rmat)
X1.debug_print()
```

Output

```
matrix:
num_row = 3, num_col = 2
node 0
node = 0, local_num_row = 3, local_num_col = 2, val = -6.50037 0.280166 1.04414
-0.759238 5.45623 0.479072
```

Return Value

- **When X is python native input:**
  It returns a numpy ndarray of shape **(n_samples, n_components)** and double (float64) type values. It contains the projection of input matrix (X) in the first principal components.

- **When X is frovedis-like input:**
  It returns a FrovedisRowmajorMatrix instance of shape **(n_samples, n_components)** and double (float64) type values. It contains the projection of input matrix (X) in the first principal components.

26.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Latent Dirichlet Allocation in frovedis**

Chapter 27

RandomForestClassifier

27.1 NAME

RandomForestClassifier - A classification algorithm that contains multiple decision trees on various subsets of the given dataset. It contains the predictions of multiple decision trees and based on the majority votes of predictions, it predicts the final output.

27.2 SYNOPSIS

```
class frovedis.mllib.ensemble.forest.RandomForestClassifier(n_estimators=100, criterion='gini',
                                                            max_depth=None,
                                                            min_samples_split=2,
                                                            min_samples_leaf=1,
                                                            min_weight_fraction_leaf=0.0,
                                                            max_features='auto',
                                                            max_leaf_nodes=None,
                                                            min_impurity_decrease=0.0,
                                                            min_impurity_split=None,
                                                            bootstrap=True, oob_score=False,
                                                            n_jobs=None, random_state=None,
                                                            verbose=0, warm_start=False,
                                                            class_weight=None, ccp_alpha=0.0,
                                                            max_samples=None, max_bins=32)
```

27.2.1 Public Member Functions

fit(X, y)
predict(X)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
score(X, y, sample_weight = None)
save(fname)
debug_print()

release()
is_fitted()

27.3 DESCRIPTION

Random Forest Classifier is a supervised machine learning algorithm used for classification using decision trees. The classifier creates a set of decision trees from randomly selected subsets of the training set. It is basically a set of decision trees from a randomly selected subset of the training set and then it collects the votes from different decision trees to decide the final prediction. **Frovedis supports both binary and multinomial random forest classification algorithms.**

During training, the input **X** is the training data and **y** is the corresponding label values (Frovedis supports any values for labels, but internally it encodes the input binary labels to 0 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.

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 RandomForestClassifier 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 RandomForestClassifier 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.

27.3.1 Detailed Description

27.3.1.1 1. RandomForestClassifier()

Parameters
n_estimators: A positive integer parameter that specifies the number of trees in the forest. (Default: 100)
criterion: A string object parameter that specifies the function to measure the quality of a split. Supported criteria are 'gini' and 'entropy'. (Default: 'gini')
- **'gini' impurity**: calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly.
- **'entropy' (information gain)**: it is applied to quantify which feature provides maximal information about the classification based on the notion of entropy.

max_depth: A positive integer parameter that specifies the maximum depth of the tree. (Default: None)
If it is None (not specified explicitly), then 'max_depth' is set to 4.
min_samples_split: An integer or float value that specifies the minimum number of samples required to split an internal node. (Default: 2)
min_samples_leaf: A positive integer or float value that specifies the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least 'min_samples_leaf' training samples in each of the left and right branches. (Default: 1)
- If it is an integer, then 'min_samples_leaf' should be greater than 0.
- If it is float, then it is set as **int(np.ceil(self.min_samples_split * self.n_samples_))**

min_weight_fraction_leaf: An unused parameter. (Default: 0.0)
max_features: A string object parameter that specifies the number of features to consider when looking for the best split:
- If it is an integer, then it will be set as **(max_features * 1.0) / n_features_**.
- If it is float, then it will be **'max_features'** number of features at each split.
- If it is 'auto', then it will be set as **sqrt(n_features_)**.
- If 'sqrt', then it will be set as **sqrt(n_features_)** (same as 'auto').
- If 'log2', then it will be set as **log2(n_features_)**.
- If None, then it will be set as **n_features_**. (Default: 'auto')

max_leaf_nodes: An unused parameter. (Default: None)
min_impurity_decrease: A positive double (float64) parameter. A node will be split if this split induces a decrease of the impurity greater than or equal to this value. (Default: 0.0)
min_impurity_split: An unused parameter. (Default: None)
bootstrap: An unused parameter. (Default: True)
oob_score: An unused parameter. (Default: False)
n_jobs: An unused parameter. (Default: None)
random_state: An integer parameter that controls the sampling of the features to consider when looking for the best split at each node (if max_features < n_features). (Default: None)
If it is None (not specified explicitly), then 'random_state' is set as -1.
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.
warm_start: An unused parameter. (Default: False)
class_weight: An unused parameter. (Default: None)
ccp_alpha: An unused parameter. (Default: 0.0)
max_samples: An unused parameter. (Default: None)
max_bins: A positive integer parameter that specifies the maximum number of bins created by ordered splits. (Default: 32)

Attributes
classes_: It is a python ndarray (any type) of unique labels given to the classifier during training. It has shape **(n_classes,)**.
n_features_: An integer value specifying the number of features when fitting the estimator.

Purpose
It initializes a RandomForestClassifier object with the given parameters.

The parameters: "min_weight_fraction_leaf", "max_leaf_nodes", "min_impurity_split", "bootstrap", "oob_score", "n_jobs", "warm_start", "class_weight", "ccp_alpha" and "max_samples" are simply kept in to make the interface uniform to the Scikit-learn RandomForestClassifier module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

27.3.1.2 2. fit(X, y)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisColmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.
y: Any python array-like object or an instance of FrovedisDvector containing the target values. It has shape **(n_samples,)**.

Purpose
It builds a forest of trees from the training data X and labels y.

For example,

```
# loading a sample matrix and labels dense data
import numpy as np
mat = np.array([[10, 0, 1, 0, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 0, 1],
                [1, 0, 0, 1, 0, 1, 0]], dtype = np.float64)
lbl = np.array([100, 500, 100, 600], dtype = np.float64)

# fitting input matrix and label on RandomForestClassifier object
from frovedis.mllib.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 10, max_depth = 4, min_samples_split = 0.5
                             min_samples_leaf = 1.2, random_state = 324)
rfc.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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels dense data
import numpy as np
mat = np.array([[10, 0, 1, 0, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 0, 1],
                [1, 0, 0, 1, 0, 1, 0]], dtype = np.float64)
lbl = np.array([100, 500, 100, 600], dtype = np.float64)

# 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)

# fitting input matrix and label on RandomForestClassifier object
from frovedis.mllib.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 10, max_depth = 4,
                             min_samples_split = 0.5, min_samples_leaf = 1.2,
                             random_state = 324)
rfc.fit(cmat,dlbl)
```

Return Value
It simply returns "self" reference.

27.3.1.3 3. predict(X)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.

Purpose
Predict class for X.

The predicted class of an input sample is a vote by the trees in the forest, weighted by their probability estimates. That is, the predicted class is the one with highest mean probability estimate across the trees.

For example,

```
# predicting on random forest classifier model
rfc.predict(mat)
```

Output

```
[100. 500. 100. 500.]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# predicting on random forest classifier model
rfc.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[100. 500. 100. 500.]
```

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

27.3.1.4 4. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by RandomForestClassifier. It is used to get parameters and their values of RandomForestClassifier class.

For example,

```
print(rfc.get_params())
```

Output

```
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini',
'max_bins': 32, 'max_depth': 4, 'max_features': 'auto', 'max_leaf_nodes': None,
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 2, 'min_samples_split': 0.5, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_state': 324,
'verbose': 0, 'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

27.3.1.5 5. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by RandomForestClassifier, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(rfc.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
rfc.set_params(criterion = 'entropy', max_depth = 5)
print("get parameters after setting:")
print(rfc.get_params())
```

Output

```
get parameters before setting:
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini',
'max_bins': 32, 'max_depth': 4, 'max_features': 'auto', 'max_leaf_nodes': None,
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 2, 'min_samples_split': 0.5, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_state': 324,
'verbose': 0, 'warm_start': False}
get parameters after setting:
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy',
'max_bins': 32, 'max_depth': 5, 'max_features': 'auto', 'max_leaf_nodes': None,
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 2, 'min_samples_split': 0.5, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_state': 324,
'verbose': 0, 'warm_start': False}
```

Return Value
It simply returns "self" reference.

27.3.1.6 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,

```
rfc.load("./out/rf_classifier_model")
```

Return Value
It simply returns "self" reference.

27.3.1.7 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 (label_map, metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the random forest classifier model
dtc.save("./out/rf_classifier_model")
```

This will save the random forest classifier model on the path "/out/rf_classifier_model". It would raise exception if the directory already exists with same name.

The 'rf_classifier_model' directory has

rf_classifier_model
|—label_map
|—metadata
|—model

'label_map' file contains information about labels mapped with their encoded value.
The 'metadata' file contains the number of classes, model kind and input datatype used for trained model.
The 'model' file contains the random forest model saved in binary format.

Return Value
It returns nothing.

27.3.1.8 8. score(X, y, sample_weight = None)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.
y: Any python array-like object containing the true labels for X. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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
rfc.score(mat,lbl)
```

Output

```
0.75
```

Return Value
It returns an accuracy score of float type.

27.3.1.9 9. debug_print()

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

For example,

```
rfc.debug_print()
```

Output

```
-------- Classification Trees (Random Forest):: --------
# of trees: 10
---- [0] ----
  # of nodes: 3, height: 1
  <1> Split: feature[2] < 0.25, IG: 0.5
   \_ (2) Predict: 2 (100%)
   \_ (3) Predict: 0 (100%)
---- [1] ----
  # of nodes: 3, height: 1
  <1> Split: feature[6] < 0.25, IG: 0.375
   \_ (2) Predict: 1 (50%)
   \_ (3) Predict: 0 (100%)
---- [2] ----
  # of nodes: 1, height: 0
  (1) Predict: 0 (50%)
---- [3] ----
  # of nodes: 3, height: 1
  <1> Split: feature[6] < 0.25, IG: 0.125
   \_ (2) Predict: 0 (50%)
   \_ (3) Predict: 0 (100%)
---- [4] ----
  # of nodes: 3, height: 1
  <1> Split: feature[3] < 0.25, IG: 0.375
   \_ (2) Predict: 0 (100%)
   \_ (3) Predict: 1 (50%)
---- [5] ----
  # of nodes: 3, height: 1
  <1> Split: feature[1] < 0.25, IG: 0.5
   \_ (2) Predict: 2 (100%)
   \_ (3) Predict: 1 (100%)
---- [6] ----
  # of nodes: 1, height: 0
  (1) Predict: 0 (100%)
---- [7] ----
  # of nodes: 1, height: 0
  (1) Predict: 2 (75%)
---- [8] ----
  # of nodes: 1, height: 0
  (1) Predict: 1 (50%)
---- [9] ----
  # of nodes: 3, height: 1
  <1> Split: feature[5] < 0.25, IG: 0.375
   \_ (2) Predict: 0 (100%)
   \_ (3) Predict: 1 (50%)
```

This output will be visible on server side. It displays the random forest having maximum depth of 4 and total 10 decision trees.

No such output will be visible on client side.

Return Value
It returns nothing.

27.3.1.10 10. release()

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

For example,

```
rfc.release()
```

This will reset the after-fit populated attributes (like classes_, n_features_) to None, along with releasing server side memory.

Return Value
It returns nothing.

27.3.1.11 11. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

27.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisDvector**
- **Random Forest Regressor in Frovedis**
- **Decision Tree Classifier in Frovedis**

Chapter 28

RandomForestRegressor

28.1 NAME

RandomForestRegressor - A regression algorithm used for predicting the final output by taking the mean of all the predictions from multiple decision trees.

28.2 SYNOPSIS

```
class frovedis.mllib.ensemble.forest.RandomForestRegressor(n_estimators=100, criterion='mse',
                                                           max_depth=None,
                                                           min_samples_split=2,
                                                           min_samples_leaf=1,
                                                           min_weight_fraction_leaf=0.0,
                                                           max_features='auto',
                                                           max_leaf_nodes=None,
                                                           min_impurity_decrease=0.0,
                                                           min_impurity_split=None,
                                                           bootstrap=True, oob_score=False,
                                                           n_jobs=None, random_state=None,
                                                           verbose=0, warm_start=False,
                                                           ccp_alpha=0.0, max_samples=None,
                                                           max_bins=32)
```

28.2.1 Public Member Functions

fit(X, y)
predict(X)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
score(X, y, sample_weight = None)
debug_print()
release()
is_fitted()

28.3 DESCRIPTION

Random Forest Regressor is a supervised machine learning algorithm used for regression using decision trees. Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn't depend on one decision tree but on multiple decision trees. The regressor creates a set of decision trees from randomly selected subsets of the training set. Then, it takes the average to improve the predictive accuracy of that dataset. So, instead of relying on one decision tree, it combines multiple decision trees in determining the final output.

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 RandomForestRegressor 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 RandomForestRegressor 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.

28.3.1 Detailed Description

28.3.1.1 1. RandomForestRegressor()

Parameters
n_estimators: A positive integer parameter that specifies the number of trees in the forest. (Default: 100)
criterion: A string object parameter that specifies the function to measure the quality of a split. (Default: 'mse')
Currently, supported criteria are 'mse' and 'mae'. The mean squared error (mse), which is equal to variance reduction as feature selection criterion. The mean absolute error (mae) uses reduction in Poisson deviance to find splits.
max_depth: A positive integer parameter that specifies the maximum depth of the tree. (Default: None)
If it is None (not specified explicitly), then 'max_depth' is set to 4.
min_samples_split: An integer or float value that specifies the minimum number of samples required to split an internal node. (Default: 2)
min_samples_leaf: A positive integer or float value that specifies the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least 'min_samples_leaf' training samples in each of the left and right branches. (Default: 1)
- If it is an integer, then 'min_samples_leaf' should be greater than 0.
- If it is float, then it is set as **int(np.ceil(self.min_samples_split * self.n_samples_))**

min_weight_fraction_leaf: An unused parameter. (Default: 0.0)
max_features: A string object parameter that specifies the number of features to consider when looking for the best split:
- If it is an integer, then it will be set as **(max_features * 1.0) / n_features_**.
- If it is float, then it will be **'max_features'** number of features at each split.
- If it is 'auto', then it will be set as **sqrt(n_features_)**.
- If 'sqrt', then it will be set as **sqrt(n_features_)** (same as 'auto').
- If 'log2', then it will be set as **log2(n_features_)**.
- If None, then it will be set as **n_features_**. (Default: 'auto')

max_leaf_nodes: An unused parameter. (Default: None)
min_impurity_decrease: A positive double (float64) parameter. A node will be split if this split induces a decrease of the impurity greater than or equal to this value. (Default: 0.0)
min_impurity_split: An unused parameter. (Default: None)
bootstrap: An unused parameter. (Default: True)
oob_score: An unused parameter. (Default: False)
n_jobs: An unused parameter. (Default: None)
random_state: An integer parameter that controls the sampling of the features to consider when looking for the best split at each node (if max_features < n_features). (Default: None)
If it is None (not specified explicitly), then 'random_state' is set as -1.
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.
warm_start: An unused parameter. (Default: False)
ccp_alpha: An unused parameter. (Default: 0.0)
max_samples: An unused parameter. (Default: None)
max_bins: A positive integer parameter that specifies the maximum number of bins created by ordered splits. (Default: 32)

Attributes
n_features_: An integer value specifying the number of features when fitting the estimator.

Purpose
It initializes a RandomForestRegressor object with the given parameters.

The parameters: "min_weight_fraction_leaf", "max_leaf_nodes", "min_impurity_split", "bootstrap", "oob_score", "n_jobs", "warm_start", "ccp_alpha" and "max_samples" are simply kept in to make the interface uniform to the Scikit-learn RandomForestRegressor module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

28.3.1.2 2. fit(X, y)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisColmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.
y: Any python array-like object or an instance of FrovedisDvector containing the target values. It has shape **(n_samples,)**.

Purpose
It builds a forest of trees from the training data X and labels y.

For example,

```
# loading a sample matrix and labels dense data
import numpy as np
mat = np.array([[10, 0, 1, 0, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 0, 1],
                [1, 0, 0, 1, 0, 1, 0]], dtype = np.float64)
lbl = np.array([1.2,0.3,1.1,1.9])

# fitting input matrix and label on RandomForestRegressor object
from frovedis.mllib.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators = 10, criterion = 'mae', max_depth = 5)
```

```
rfr.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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels dense data
import numpy as np
mat = np.array([[10, 0, 1, 0, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 0, 1],
                [1, 0, 0, 1, 0, 1, 0]], dtype = np.float64)
lbl = np.array([1.2,0.3,1.1,1.9])

# 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)

# fitting input matrix and label on RandomForestRegressor object
from frovedis.mllib.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators = 10, criterion = 'mae', max_depth = 5)
rfr.fit(cmat,dlbl)
```

Return Value
It simply returns "self" reference.

28.3.1.3 3. predict(X)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.

Purpose
Predict regression value for X.

The predicted regression target of an input sample is computed as the mean predicted regression targets of the trees in the forest.

For example,

```
# predicting on random forest regressor model
rfr.predict(mat)
```

Output

```
[1.29083333 0.8075     1.0475     1.48083333]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# predicting on random forest regressor model using pre-constructed input
rfr.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[1.29083333 0.8075     1.0475     1.48083333]
```

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

28.3.1.4 4. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by RandomForestRegressor. It is used to get parameters and their values of RandomForestRegressor class.

For example,

```
print(rfr.get_params())
```

Output

```
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'mae', 'max_bins': 32,
'max_depth': 5, 'max_features': 'auto', 'max_leaf_nodes': None,
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_state': -1,
'verbose': 0, 'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

28.3.1.5 5. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by RandomForestRegressor, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(rfr.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
rfr.set_params(max_depth = 4)
print("get parameters after setting:")
print(rfr.get_params())
```

Output

```
get parameters before setting:
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'mae', 'max_bins': 32,
'max_depth': 5, 'max_features': 'auto', 'max_leaf_nodes': None,
```

```
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_state': -1,
'verbose': 0, 'warm_start': False}
get parameters after setting:
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'mae', 'max_bins': 32,
'max_depth': 4, 'max_features': 'auto', 'max_leaf_nodes': None,
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_state': -1,
'verbose': 0, 'warm_start': False}
```

Return Value
It simply returns "self" reference.

28.3.1.6 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,

```
rfr.load("./out/rf_regressor_model")
```

Return Value
It simply returns "self" reference.

28.3.1.7 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the random forest regressor model
rfr.save("./out/rf_regressor_model")
```

This will save the random forest regressor model on the path "/out/rf_regressor_model". It would raise exception if the directory already exists with same name.

The 'rf_regressor_model' directory has
rf_regressor_model
|—metadata
|—model

The 'metadata' file contains the model kind and input datatype used for trained model.
The 'model' file contains the random forest model saved in binary format.

Return Value
It returns nothing.

28.3.1.8 8. score(X, y, sample_weight = None)

Parameters
X: A numpy dense or any python array-like object or an instance of FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**. Currently, it supports only dense data as input.
y: Any python array-like object containing the true values for X. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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
rfr.score(mat,lbl)
```

Output

```
0.65
```

Return Value
It returns an R2 score of float type.

28.3.1.9 9. debug_print()

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

For example,

```
rfr.debug_print()
```

Output

```
-------- Regression Trees (Random Forest):: --------
# of trees: 10
---- [0] ----
  # of nodes: 3, height: 1
  <1> Split: feature[1] < 0.25, IG: 0.191667
   \_ (2) Predict: 1.43333
   \_ (3) Predict: 0.3
---- [1] ----
  # of nodes: 5, height: 2
  <1> Split: feature[2] < 0.25, IG: 0.0416667
   \_ <2> Split: feature[1] < 0.5, IG: 0.711111
   |  \_ (4) Predict: 1.9
```

```
   |  \_ (5) Predict: 0.3
   \_ (3) Predict: 1.2
---- [2] ----
  # of nodes: 1, height: 0
  (1) Predict: 1.375
---- [3] ----
  # of nodes: 5, height: 2
  <1> Split: feature[1] < 0.25, IG: 0.4
   \_ (2) Predict: 1.9
   \_ <3> Split: feature[3] < 0.5, IG: 0.4
      \_ (6) Predict: 1.1
      \_ (7) Predict: 0.3
---- [4] ----
  # of nodes: 3, height: 1
  <1> Split: feature[1] < 0.25, IG: 0.4
   \_ (2) Predict: 1.9
   \_ (3) Predict: 1.1
---- [5] ----
  # of nodes: 3, height: 1
  <1> Split: feature[4] < 0.25, IG: 0.4
   \_ (2) Predict: 0.3
   \_ (3) Predict: 1.1
---- [6] ----
  # of nodes: 5, height: 2
  <1> Split: feature[5] < 0.25, IG: 0.141667
   \_ (2) Predict: 1.1
   \_ <3> Split: feature[2] < 0.5, IG: 0.311111
      \_ (6) Predict: 1.9
      \_ (7) Predict: 1.2
---- [7] ----
  # of nodes: 7, height: 3
  <1> Split: feature[0] < 2.5, IG: 0.025
   \_ <2> Split: feature[6] < 0.5, IG: 1.11022e-16
   |  \_ <4> Split: feature[1] < 0.5, IG: 0.8
   |  |  \_ (8) Predict: 1.9
   |  |  \_ (9) Predict: 0.3
   |  \_ (5) Predict: 1.1
   \_ (3) Predict: 1.2
---- [8] ----
  # of nodes: 3, height: 1
  <1> Split: feature[2] < 0.25, IG: 0.2625
   \_ (2) Predict: 1.9
   \_ (3) Predict: 1.2
---- [9] ----
  # of nodes: 5, height: 2
  <1> Split: feature[3] < 0.25, IG: 0.279167
   \_ <2> Split: feature[2] < 0.5, IG: 0.0444444
   |  \_ (4) Predict: 1.1
   |  \_ (5) Predict: 1.2
   \_ (3) Predict: 0.3
```

This output will be visible on server side. It displays the random forest having maximum depth of 5 and total 10 decision trees.

No such output will be visible on client side.

Return Value
It returns nothing.

28.3.1.10 10. release()

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

For example,

```
rfr.release()
```

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

Return Value
It returns nothing.

28.3.1.11 11. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

28.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisDvector**
- **Random Forest Classifier in Frovedis**
- **Decision Tree Regressor in Frovedis**

Chapter 29

Ridge Regression

29.1 NAME

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

29.2 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)
```

29.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

29.3 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
```

During training, the input **X** is the training data and **y** is the corresponding label values 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 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 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.

29.3.1 Detailed Description

29.3.1.1 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: An unused parameter (Default: False)
copy_X: An unused parameter (Default: True)
max_iter: An integer parameter specifying the 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. Initially solver is "auto" by default. In such cases, it will select "sag" solver. Both "sag" and "lbfgs" can handle L2 penalty.

random_state: An unused parameter. (Default: None)
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 initializes 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 module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

29.3.1.2 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 containing the target values. It has shape **(n_samples,)**.
sample_weight: Python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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.

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, especially when same data would be used for multiple executions.

Return Value
It simply returns "self" reference.

29.3.1.3 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.

In case pre-constructed frovedis-like training data such as FrovedisColmajorMatrix (X) is provided during prediction, then "X.to_frovedis_rowmatrix()" will be used for prediction.

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

29.3.1.4 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 containing the target values for X. It has shape **(n_samples,)**.
sample_weight: Python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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.

Return Value
It returns an R2 score of float type.

29.3.1.5 5. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by Ridge. It is used to get parameters and their values of Ridge class.

Return Value
A dictionary of parameter names mapped to their values.

29.3.1.6 6. set_params(**params)

Parameters
*****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by Ridge, used to set parameter values.

Return Value
It simply returns "self" reference.

29.3.1.7 7. 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).

Return Value
It simply returns "self" reference.

29.3.1.8 8. 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

Suppose 'RidgeModel' directory is the model created, It will have

RidgeModel
|——— metadata
|——— model

The metadata file contains the number of classes, model kind, input datatype used for trained model.
Here, the model file contains information about weights, intercept and threshold.

It would raise exception if the 'RidgeModel' directory already existed with same name.

Return Value
It returns nothing.

29.3.1.9 9. debug_print()

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

No such output will be visible on client side.

Return Value
It returns nothing.

29.3.1.10 10. release()

Purpose
It can be used to release the in-memory model at frovedis server. With this, after-fit populated attributes are reset to None, along with releasing server side memory.

Return Value
It returns nothing.

29.3.1.11 11. 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'.

29.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Linear Regression in Frovedis**
- **Lasso Regression in Frovedis**

Chapter 30

SGDClassifier

30.1 NAME

SGDClassifier - A classification algorithm used to predict the labels with various loss functions. This estimator implements regularized linear models with stochastic gradient descent (SGD) learning.

30.2 SYNOPSIS

```
class frovedis.mllib.linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001,
                                                 l1_ratio=0.15, fit_intercept=True,
                                                 max_iter=1000, tol=1e-3,
                                                 shuffle=True, verbose=0,
                                                 epsilon=0.1, n_jobs=None,
                                                 random_state=None,
                                                 learning_rate="invscaling",
                                                 eta0=1.0, power_t=0.5,
                                                 early_stopping=False,
                                                 validation_fraction=0.1,
                                                 n_iter_no_change=5,
                                                 class_weight=None,
                                                 warm_start=False,
                                                 average=False)
```

30.2.1 Public Member Functions

fit(X, y, coef_init = None, intercept_init = None, sample_weight = None)
predict(X)
predict_proba(X)
score(X, y, sample_weight = None)
load(fname, dtype = None)
save(fname)
get_params(deep = True)
set_params(**params)
debug_print()

release()
is_fitted()

30.3 DESCRIPTION

Stochastic Gradient Descent (SGD) is used for discriminative learning of linear classifiers under convex loss functions such as SVM and Logistic regression. It has been successfully applied to large-scale datasets because the update to the coefficients is performed for each training instance, rather than at the end of instances. **Frovedis supports both binary and multinomial Stochastic Gradient Descent (SGD) classifier algorithms.**

Stochastic Gradient Descent (SGD) classifier basically implements a plain SGD learning routine supporting various loss functions and penalties for classification. It implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule. It is a linear method which uses the following loss functions:
1) hinge
2) log
3) squared_loss

It supports ZERO, L1 and L2 regularization to address the overfit problem.

During training, the input **X** is the training data and **y** are their corresponding label values (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.

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 SGDClassifier (Stochastic Gradient Descent Classification) 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 SGDClassifier 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.

30.3.1 Detailed Description

30.3.1.1 1. SGDClassifier()

Parameters
loss: A string object parameter containing the loss function type to use. Currently, frovedis supports 'hinge', 'log'and 'squared_loss' functions. (Default: 'hinge')
penalty: A string object parameter containing the regularizer type to use. Currently none, l1 and l2 are supported by Frovedis. (Default: 'l2')
If it is None (not specified explicitly), it will be set as 'ZERO' regularization type.
alpha: Zero or a positive double (float64) smoothing parameter. (Default: 0.0001)
l1_ratio: An unused parameter. (Default: 0.15)
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the

decision function. (Default: True)
max_iter: A positive integer parameter specifying maximum iteration count. (Default: 1000)
tol: A double (float64) parameter specifying the convergence tolerance value. It must be zero or a positive value. (Default: 1e-3)
shuffle: An unused parameter. (Default: True)
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.
epsilon: An unused parameter. (Default: 0.1)
n_jobs: An unused parameter. (Default: None)
random_state: An unused parameter. (Default: None)
learning_rate: A string object parameter containing the learning rate. (Default: 'invscaling')
Unlike sklearn, Frovedis only supports 'invscaling' learning rate. 'invscaling' gradually decreases the learning rate **'learning_rate_'** at each time step **'t'** using an inverse scaling exponent of **'power_t'**.
learning_rate_ = eta0 / pow(t, power_t)
eta0: A double (float64) parameter specifying the initial learning rate for the 'invscaling' schedules. (Default: 1.0)
power_t: A double (float64) parameter specifying the exponent for inverse scaling learning rate. (Default: 0.5)
early_stopping: An unused parameter. (Default: False)
validation_fraction: An unused parameter. (Default: 0.1)
n_iter_no_change: An unused parameter. (Default: 5)
class_weight: An unused parameter. (Default: None)
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)
average: An unused parameter. (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.
Shape of this attribute depends on the n_classes.
- If 'classes_' is 2, then the shape **(1, n_features)**
- If 'classes_' is more then 2, then the shape is **(n_classes, n_features)**.

intercept_: It is a python ndarray(float or double (float64) values depending on input matrix data type) and has shape **(1,)**. It specifies the constants in decision function.
classes_: It is a python ndarray(any type) of unique labels given to the classifier during training. It has shape **(n_classes,)**. This attribute is not available for **squared_loss**.
n_iter: An integer value used to get the actual iteration point at which the problem is converged.

Purpose
It initializes a SGDClassifier object with the given parameters.

The parameters: "l1_ratio", "shuffle", "epsilon", "n_jobs", "random_state", "early_stopping", "validation_fraction", "n_iter_no_change", "class_weight" and "average" are simply kept to make the interface uniform to Scikit-learn SGDClassifier module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

30.3.1.2 2. fit(X, y, coef_init = None, intercept_init = None, 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 containing the target labels. It has shape

(n_samples,).
coef_init: An unused parameter that specifies the initial coefficients to warm-start the optimization. (Default: None)
intercept_init: An unused parameter that specifies the initial intercept to warm-start the optimization. (Default: None)
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

Purpose
It accepts the training feature matrix (X) and corresponding output labels (y) as inputs from the user and trains a SGDClassifier model with specified 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 SGDClassifier object
from frovedis.mllib.linear_model import SGDClassifier
sgd_clf = SGDClassifier().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, especially 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)
dlbl = FrovedisDvector(lbl)

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

Return Value
It simply returns "self" reference.

30.3.1.3 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,

```
sgd_clf.predict(mat)
```

Output:

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

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# predicting on SGDClassifier using frovedis-like input
sgd_clf.predict(cmat.to_frovedis_rowmatrix())
```

Output

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

Return Value
For **squared_loss** loss, it returns a numpy array of **double (float64)** type and for other loss functions it returns a numpy array of **int64** type containing the predicted outputs. It has shape **(n_samples,)**.

30.3.1.4 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. It has shape **(n_samples, n_features)**.

Purpose
Perform classification on an array and return probability estimates for the test vector X.

It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server. Unlike sklearn, it performs the classification on an array and returns the probability estimates for the test feature matrix (X).

This method is not available for "hinge" and "squared_loss" function.

For example,

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

Output

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

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)
```

```
# finds the probablity sample for each class in the model
sgd_clf.predict_proba(rmat)
```

Output

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

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

30.3.1.5 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. It has shape **(n_samples, n_features)**.
y: Any python array-like object containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python narray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**.
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample. (Default: None)

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

For 'log' and 'hinge' loss, 'accuracy_score' is calculated and for 'squared_loss', 'r2_score' is calculated.

For example,

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

Output

```
0.91
```

Return Value
For 'log' and 'hinge' loss, it returns 'accuracy_score' and for 'squared_loss', it returns 'r2_score' of double (float64) type.

30.3.1.6 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)
For 'squared_loss' this method doesn't load the saved file 'label_map'.

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

For example,

```
# loading the SGDClassifier model
sgd_clf.load("./out/SCLFModel")
```

Return Value
It simply returns "self" instance.

30.3.1.7 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 (label_map, metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
sgd_clf.save("./out/SCLFModel")
```

The SCLFModel contains below directory structure:
SCLFModel
|——label_map
|——metadata
|——model

'label_map' contains information about labels mapped with their encoded value. **This information is not saved for 'squared_loss' loss function.**

'metadata' represents the detail about loss function, model_kind and datatype of training vector.
Here, the model file contains information about model_id, model_kind and datatype of training vector.

This will save the SGDClassifier model on the path '/out/SCLFModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

30.3.1.8 8. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by SGDClassifier. It is used to get parameters and their values of SGDClassifier class.

For example,

```
print(sgd_clf.get_params())
```

Output

```
{'alpha': 0.0001, 'average': False, 'class_weight': None,
'early_stopping': False, 'epsilon': 0.1, 'eta0': 1.0, 'fit_intercept': True,
'l1_ratio': 0.15, 'learning_rate': 'invscaling', 'loss': 'hinge', 'max_iter': 1000,
'n_iter_no_change': 5, 'n_jobs': None, 'penalty': 'l2', 'power_t': 0.5,
'random_state': None, 'shuffle': True, 'tol': 0.001, 'validation_fraction': 0.1,
'verbose': 0, 'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

30.3.1.9 9. set_params(**params)

Parameters
***params**: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by SGDClassifier, used to set parameter values.

For example,

```
print("Get parameters before setting:")
print(sgd_clf.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
sgd_clf.set_params(penalty = 'l1', fit_intercept = False)
print("Get parameters before setting:")
print(sgd_clf.get_params())
```

Output

```
Get parameters before setting:
{'alpha': 0.0001, 'average': False, 'class_weight': None, 'early_stopping': False,
'epsilon': 0.1, 'eta0': 1.0, 'fit_intercept': True, 'l1_ratio': 0.15,
'learning_rate': 'invscaling', 'loss': 'hinge', 'max_iter': 1000,
'n_iter_no_change': 5, 'n_jobs': None, 'penalty': 'l2', 'power_t': 0.5,
'random_state': None, 'shuffle': True, 'tol': 0.001, 'validation_fraction': 0.1,
'verbose': 0, 'warm_start': False}
Get parameters before setting:
{'alpha': 0.0001, 'average': False, 'class_weight': None, 'early_stopping': False,
'epsilon': 0.1, 'eta0': 1.0, 'fit_intercept': False, 'l1_ratio': 0.15,
'learning_rate': 'invscaling', 'loss': 'hinge', 'max_iter': 1000,
'n_iter_no_change': 5, 'n_jobs': None, 'penalty': 'l1', 'power_t': 0.5,
'random_state': None, 'shuffle': True, 'tol': 0.001, 'validation_fraction': 0.1,
'verbose': 0, 'warm_start': False}
```

Return Value
It simply returns “self” reference.

30.3.1.10 10. 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,

```
sgd_clf.debug_print()
```

Output:

```
-------- Weight Vector:: --------
14072.2 22454.1 83330.6 45451.2 139.907 -12.3658 -191.893 -82.7364 265.627 109.118
46.0452 1510.72 -250.729 -38123.1 9.74564 -4.16035 -16.0092 0.837207 26.3526 3.59399
14821.1 29161.9 85356 -57710.4 183.668 -99.8164 -348.61 -70.554 386.649 113.928
Intercept:: 1845.32
Threshold:: 0
```

This output will be visible on server side. It displays the weights, intercept and threshold values on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

30.3.1.11 11. release()

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

For example,

```
sgd_clf.release()
```

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

Return Value
It returns nothing.

30.3.1.12 12. 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'.

30.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **SGDRegressor in Frovedis**

Chapter 31

SGDRegressor

31.1 NAME

SGDRegressor - A regression algorithm used to predict the labels with various loss functions. This estimator implements regularized linear models with stochastic gradient descent (SGD) learning.

31.2 SYNOPSIS

```
class frovedis.mllib.linear_model.SGDRegressor(loss='squared_loss', penalty='l2', alpha=0.0001,
                                               l1_ratio=0.15, fit_intercept=True,
                                               max_iter=1000, tol=1e-3,
                                               shuffle=True, verbose=0,
                                               epsilon=0.1, random_state=None,
                                               learning_rate="invscaling",
                                               eta0=0.001, power_t=0.25,
                                               early_stopping=False,
                                               validation_fraction=0.1,
                                               n_iter_no_change=5,
                                               warm_start=False,
                                               average=False)
```

31.2.1 Public Member Functions

fit(X, y, coef_init = None, intercept_init = None, sample_weight = None)
predict(X)
score(X, y, sample_weight = None)
load(fname, dtype = None)
save(fname)
get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

31.3 DESCRIPTION

Stochastic Gradient Descent (SGD) is used for discriminative learning of linear regressors under convex loss functions such as SVM and Logistic regression. It has been successfully applied to large-scale datasets because the update to the coefficients is performed for each training instance, rather than at the end of instances.

Stochastic Gradient Descent (SGD) regressor basically implements a plain SGD learning routine supporting various loss functions and penalties for regression.

It implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule. It is a linear method which uses the following loss functions:
1) squared_loss
2) epsilon_insensitive
3) squared_epsilon_insensitive

It supports ZERO, L1 and L2 regularization to address the overfit problem.

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 SGDRegressor (Stochastic Gradient Descent 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 SGDRegressor 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.

31.3.1 Detailed Description

31.3.1.1 1. SGDRegressor()

Parameters
loss: A string object parameter containing the loss function type to use. Currently, frovedis supports 'squared_loss', 'epsilon_insensitive'and 'squared_epsilon_insensitive' functions. (Default: 'squared_loss')
penalty: A string object parameter containing the regularizer type to use. Currently none, l1 and l2 are supported by Frovedis. (Default: 'l2')
If it is None (not specified explicitly), it will be set as 'ZERO' regularization type.
alpha: Zero or a positive double (float64) smoothing parameter. (Default: 0.0001)
l1_ratio: An unused parameter. (Default: 0.15)
fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)
max_iter: A positive integer parameter specifying maximum iteration count. (Default: 1000)
tol: A double (float64) parameter specifying the convergence tolerance value. It must be zero or a positive value. (Default: 1e-3)
shuffle: An unused parameter. (Default: True)
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.
epsilon: A zero or positive double (float64) parameter used in the epsilon-insensitive loss function. (Default: 0.1)

random_state: An unused parameter. (Default: None)
learning_rate: A string object parameter containing the learning rate. (Default: 'invscaling')
Unlike sklearn, Frovedis only supports 'invscaling' learning rate. 'invscaling' gradually decreases the learning rate **'learning_rate_'** at each time step **'t'** using an inverse scaling exponent of **'power_t'**.
learning_rate_ = eta0 / pow(t, power_t)
eta0: A double (float64) parameter specifying the initial learning rate for the 'invscaling' and 'optimal' schedules. (Default: 0.001)
power_t: An unused parameter which specifies the exponent for inverse scaling learning rate. Although, this parameter is unused in frovedis but it must be of double (float64) type. This is simply done to keep the behavior consistent with Scikit-learn. (Default: 0.25)
early_stopping: An unused parameter. (Default: False)
validation_fraction: An unused parameter. (Default: 0.1)
n_iter_no_change: An unused parameter. (Default: 5)
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)
average: An unused parameter. (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 is of shape **(n_features,)**.
intercept_: It is a python ndarray (float or double (float64) values depending on input matrix data type) and has shape **(1,)**. It specifies the constants in decision function.
_n_iter_: An integer value used to get the actual iteration point at which the problem is converged.

Purpose
It initializes a SGDRegressor object with the given parameters.

The parameters: "l1_ratio", "shuffle", "random_state", "early_stopping", "validation_fraction", "n_iter_no_change" and "average" are simply kept to make the interface uniform to Scikit-learn SGDRegressor module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

31.3.1.2 2. fit(X, y, coef_init = None, intercept_init = None, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python ndarray or an instance of FrovedisCRSMatrix for sparse data and FrovedisColmajorMatrix for dense data. It has shape **(n_samples, n_features)**.
y: Any python ndarray or an instance of FrovedisDvector containing the target values. It has shape **(n_samples,)**.
coef_init: An unused parameter that specifies the initial coefficients to warm-start the optimization. (Default: None)
intercept_init: An unused parameter that specifies the initial intercept to warm-start the optimization. (Default: None)
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), an uniform weight vector is assigned on each input sample.

Purpose
It accepts the training feature matrix (X) and corresponding target values (y) as inputs from the user and trains a SGDRegressor model with specified regularization with those data at frovedis server.

For example,

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

```
mat, lbl = load_diabetes(return_X_y = True)

# fitting input matrix and label on SGDRegressor object
from frovedis.mllib.linear_model import SGDRegressor
sgd_reg = SGDRegressor(max_iter = 50000000, loss = "squared_epsilon_insensitive",
                       penalty = 'none', tol = 0.0000001).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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels data
from sklearn.datasets import load_diabetes
mat, lbl = load_diabetes(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)
dlbl = FrovedisDvector(lbl)

# SGDRegressor with pre-constructed frovedis-like inputs
from frovedis.mllib.linear_model import SGDRegressor
sgd_reg = SGDRegressor(max_iter = 50000000, loss = "squared_epsilon_insensitive",
                       penalty = 'none', tol = 0.0000001).fit(cmat,dlbl)
```

Return Value
It simply returns "self" reference.

31.3.1.3 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,

```
sgd_reg.predict(mat)
```

Output:

```
[187.30000503 79.39151311 168.59790769 155.75933185 129.99656706
 89.71373962 96.12127367 152.37303585 163.56868847 181.57865709
 .....
 .....
 123.18827921 64.13627628 185.64720615 136.37181437 139.34641795
 185.9899156 72.21133803]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "cmat" is FrovedisColmajorMatrix, we have created FrovedisRowmajorMatrix.
# predicting on SGDRegressor using frovedis-like input
sgd_reg.predict(cmat.to_frovedis_rowmatrix())
```

Output

```
[187.30000503 79.39151311 168.59790769 155.75933185 129.99656706
 89.71373962 96.12127367 152.37303585 163.56868847 181.57865709
 .....
 .....
 123.18827921 64.13627628 185.64720615 136.37181437 139.34641795
 185.9899156 72.21133803]
```

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

31.3.1.4 4. score(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python ndarray or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**.
y: Any python ndarray containing the target values. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), 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
sgd_reg.score(mat, lbl)
```

Output

```
0.52
```

Return Value
It returns an R2 score of double (float64) type.

31.3.1.5 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 SGDRegressor model
sgd_reg.load("./out/SGDRegressorModel")
```

Return Value
It simply returns "self" reference.

31.3.1.6 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
sgd_reg.save("./out/SGDRegressorModel")
```

The **SGDRegressorModel** contains below directory structure:

SGDRegressorModel
|——metadata
|——model

'metadata' represents the detail about loss function, model_kind and datatype of training vector.
The 'model' file contains the SGDRegressor model saved in binary format.

This will save the SGDRegressor model on the path '/out/SGDRegressorModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

31.3.1.7 7. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by SGDRegressor. It is used to get parameters and their values of SGDRegressor class.

For example,

```
print(sgd_reg.get_params())
```

Output

```
{'alpha': 0.0001, 'average': False, 'early_stopping': False, 'epsilon': 0.1,
'eta0': 0.001, 'fit_intercept': True, 'l1_ratio': 0.15, 'learning_rate': 'invscaling',
'loss': 'squared_loss', 'max_iter': 1000, 'n_iter_no_change': 5, 'penalty': 'l2',
'power_t': 0.25, 'random_state': None, 'shuffle': True, 'tol': 0.001,
'validation_fraction': 0.1, 'verbose': 0, 'warm_start': False}
```

Return Value
A dictionary of parameter names mapped to their values.

31.3.1.8 8. set_params(**params)

Parameters
***params**: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by SGDRegressor, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(sgd_reg.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
sgd_reg.set_params(penalty = 'l1', fit_intercept = False)
print("get parameters before setting:")
print(sgd_reg.get_params())
```

Output

```
get parameters before setting:
{'alpha': 0.0001, 'average': False, 'early_stopping': False, 'epsilon': 0.1,
'eta0': 0.001, 'fit_intercept': True, 'l1_ratio': 0.15, 'learning_rate': 'invscaling',
'loss': 'squared_loss', 'max_iter': 1000, 'n_iter_no_change': 5, 'penalty': 'l2',
'power_t': 0.25, 'random_state': None, 'shuffle': True, 'tol': 0.001,
'validation_fraction': 0.1, 'verbose': 0, 'warm_start': False}
get parameters before setting:
{'alpha': 0.0001, 'average': False, 'early_stopping': False, 'epsilon': 0.1,
'eta0': 0.001, 'fit_intercept': False, 'l1_ratio': 0.15, 'learning_rate': 'invscaling',
'loss': 'squared_loss', 'max_iter': 1000, 'n_iter_no_change': 5, 'penalty': 'l1',
'power_t': 0.25, 'random_state': None, 'shuffle': True, 'tol': 0.001,
'validation_fraction': 0.1, 'verbose': 0, 'warm_start': False}
```

Return Value
It simply returns “self” reference.

31.3.1.9 9. 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,

```
sgd_reg.debug_print()
```

Output:

```
-------- Weight Vector:: --------
-8.26881e+10 -1.23538e+11 -1.81876e+11 -9.355e+08 -8.27686e+09 -8.80985e+10 -1.0433e+12
-4.84075e+10 -1.69485e+11 -6.51606e+12 -2.66758e+11 -4.63121e+12 -2.0491e+11
Intercept:: -1.42647e+10
```

This output will be visible on server side. It displays the weights and intercept values on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

31.3.1.10 10. release()

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

For example,

```
sgd_reg.release()
```

This will reset the after-fit populated attributes (like coef_, intercept_, n_iter_) to None, along with releasing server side memory.

Return Value
It returns nothing.

31.3.1.11 11. 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'.

31.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **SGDRegressor in Frovedis**

Chapter 32

Spectral Clustering

32.1 NAME

Spectral Clustering - A clustering algorithm commonly used in EDA (exploratory data analysis). It uses the spectrum (eigenvalues) of the similarity matrix of the data to perform clustering.

32.2 SYNOPSIS

```
class frovedis.mllib.cluster.SpectralClustering(n_clusters=8, eigen_solver=None,
                                                n_components=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,
                                                max_iter=300, eps=1e-4, norm_laplacian=True,
                                                mode=3, drop_first=True)
```

32.2.1 Public Member Functions

fit(X, y = None)
fit_predict(X, y = None)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
load(fname, dtype = None)
save(fname)
debug_print()
release()
is_fitted()

32.3 DESCRIPTION

Clustering is an unsupervised learning problem where 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.

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 Spectral Clustering 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 Spectral Clustering 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.

32.3.1 Detailed Description

32.3.1.1 1. SpectralClustering()

Parameters
n_clusters: A positive integer parameter specifying the number of clusters. The number of clusters should be greater than 0 and less than or equal to n_samples. (Default: 8)
eigen_solver: A string object parameter. It is the eigenvalue decomposition strategy to use. (Default: None)
When it is None (not specified explicitly), it will be set as 'arpack'. Only 'arpack' eigen solver is supported.
n_components: A positive integer parameter containing the number of components for clusters. It is used to compute the number of eigenvectors for spectral embedding. The number of components should be in between 1 to n_features. (Default: None)
When it is None (not specified explicitly), it will be equal to the number of clusters.
random_state: Zero or positive integer parameter. It is None by default.
When it is None (not specified explicitly), it will be set as 0. (unused)
n_init: A positive integer parameter that specifies the number of times the k-means algorithm will be run with different centroid seeds. (Default: 10)
gamma: The double (float64) parameter required for computing nearby relational meaningful eigenvalues. (Default: 1.0)
When it is None (specified explicitly), it will be set as 1.0.
Kernel coefficient for rbf is "[np.exp(-gamma * d(X,X) ** 2)]" kernel. Ignored for affinity='nearest_neighbors'.
affinity: A string object parameter which tells how to construct the affinity matrix. (Default: 'rbf')
When it is None (specified explicitly), it will be set as 'rbf'. Only 'rbf', 'nearest_neighbors' and 'precomputed' are supported.
- **'nearest_neighbors'**: construct the affinity matrix by computing a graph of nearest neighbors.
- **'rbf'**: construct the affinity matrix using a radial basis function (RBF) kernel.
- **'precomputed'**: interpret X as a precomputed affinity matrix, where larger values indicate greater similarity between instances.

n_neighbors: A positive integer parameter that specifies the number of neighbors to be used when constructing the affinity matrix using the nearest neighbors method. It must be in between 1 to n_samples. It is applicable only when affinity = 'nearest_neighbors'. (Default: 10)
eigen_tol: Stopping criterion for eigen decomposition of the Laplacian matrix when using 'arpack' eigen_solver. (unused)
assign_labels: A string object parameter that specifies the strategy to use to assign labels in the embedding

space. When it is None (specified explicitly), it will be set as 'kmeans'. Only 'kmeans' is supported. (Default: 'kmeans')
degree: Degree of the polynomial kernel. (unused)
coef0: Zero coefficient for polynomial and sigmoid kernels. (unused)
kernel_params: Parameters (keyword arguments) and values for kernel. (unused)
n_jobs: The number of parallel jobs to run. (unused)
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.
max_iter: A positive integer parameter containing the maximum number of iteration count for kmeans assignment. (Default: 300)
eps: Zero or a positive double parameter containing the tolerance value for kmeans. (Default: 1e-4)
norm_laplacian: A boolean parameter if set to True, then compute normalized Laplacian, else not. (Default: True)
mode: An integer parameter required to set the eigen computation method. It can be either 1 (for generic) or 3 (for shift-invert). It is applicable only for dense data. For more details refer ARPACK computation modes. (Default: 3)
drop_first: A boolean parameter if set to True, then drops the first eigenvector. The first eigenvector of a normalized Laplacian is full of constants, thus if it is set to True, then (n_components + 1) eigenvectors are computed and will drop the first vector. Otherwise, it will calculate 'n_components' number of eigenvectors. (Default: True)

Attributes
affinity_matrix_:
1. **For python native dense input:**
- When affinity = 'precomputed/rbf', it returns a numpy array
- When affinity = 'nearest_neighbors', it returns a scipy matrix
2. **For frovedis-like dense input:**
- When affinity = 'precomputed/rbf', returns a FrovedisRowmajorMatrix
- When affinity = 'nearest_neighbors', returns a FrovedisCRSMatrix
3. **For python native sparse input:**
- When affinity = 'precomputed/nearest_neighbors', it returns a scipy matrix
- When affinity = 'rbf', it returns a numpy array
4. **For frovedis-like sparse input:**
- When affinity = 'precomputed/nearest_neighbors', it a returns FrovedisCRSMatrix
- When affinity = 'rbf', it returns a FrovedisRowmajorMatrix

In all cases, the output is of float or double (float64) type and of shape **(n_samples, n_samples)**.
labels_: A python ndarray of int64 type values and has shape **(n_clusters,)**. It contains the predicted cluster labels for each point.

Purpose
It initializes a Spectral Clustering object with the given parameters.

The parameters: "eigen_tol", "degree", "coef0", "kernel_params" and "n_jobs", "random_state" are simply kept in to make the interface uniform to the Scikit-learn Spectral Clustering module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

32.3.1.2 2. fit(X, y = 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. If affinity="precomputed", it needs to be of

shape **(n_samples, n_samples)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, as in Scikit-learn as well.

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

For example,

```
# loading sample matrix data
mat = np.loadtxt("./input/spectral_data.txt")

# fitting input matrix on SpectralClustering object
from frovedis.mllib.cluster import SpectralClustering
spec = SpectralClustering(n_clusters = 2).fit(mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading sample matrix data
mat = np.loadtxt("./input/spectral_data.txt")

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# Spectral Clustering with pre-constructed frovedis-like inputs
from frovedis.mllib.cluster import SpectralClustering
spec = SpectralClustering(n_clusters = 2).fit(rmat)
```

Return Value
It simply returns "self" reference.

32.3.1.3 3. fit_predict(X, y = 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. If affinity="precomputed", it needs to be of shape **(n_samples, n_samples)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, as in Scikit-learn as well.

Purpose
It fits the given data points (X) and returns the predicted labels based on cluster formed during the fit.

For example,

```
# loading sample matrix data
mat = np.loadtxt("./input/spectral_data.txt")

# fitting input matrix on Spectral Clustering object
from frovedis.mllib.cluster import SpectralClustering
spec = SpectralClustering(n_clusters = 2)
print(spec.fit_predict(mat))
```

Output

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

It prints the predicted cluster labels after training is completed.

Like in fit(), we can also provide frovedis-like input in fit_predict() for faster computation.

For example,

```
# loading sample matrix data
mat = np.loadtxt("./input/sample_data.txt")

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# using pre-constructed input matrix
from frovedis.mllib.cluster import SpectralClustering
spec = SpectralClustering(n_clusters = 2)
print(spec.fit_predict(rmat))
```

Output

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

It prints the predicted cluster labels after training is completed.

Return Value
It returns a numpy array of int32 type values containing the cluster labels. It has a shape **(n_samples,)**.

32.3.1.4 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. If affinity="precomputed", it needs to be of shape **(n_samples, n_samples)**.
y: A python ndarray or an instance of FrovedisVector containing the true labels for X. It has shape **(n_samples,)**.
sample_weight: An unused parameter whose default value is None. It is simply ignored in frovedis implementation.

Purpose
It uses homogeneity score on given true labels and predicted labels i.e homogeneity score of self.predict(X, y) wrt. y.

For example,

```
spec.score(train_mat, [0, 0, 1, 1, 1])
```

Output

```
1.0
```

Return Value
It returns a homogeneity score of float type.

32.3.1.5 5. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by SpectralClustering. It is used to get parameters and their values of SpectralClustering class.

For example,

```
print(spec.get_params())
```

Output

```
{'affinity': 'rbf', 'assign_labels': 'kmeans', 'coef0': 1, 'degree': 3,
'drop_first': True, 'eigen_solver': 'arpack', 'eigen_tol': 0.0, 'eps': 0.0001,
'gamma': 1.0, 'kernel_params': None, 'max_iter': 300, 'mode': 3, 'n_clusters': 2,
'n_components': 2, 'n_init': 10, 'n_jobs': None, 'n_neighbors': 10,
'norm_laplacian': True, 'random_state': None, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

32.3.1.6 6. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by SpectralClustering, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(spec.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
spec.set_params(n_clusters = 3, affinity = 'precomputed')
print("get parameters after setting:")
print(spec.get_params())
```

Output

```
get parameters before setting:
{'affinity': 'rbf', 'assign_labels': 'kmeans', 'coef0': 1, 'degree': 3,
'drop_first': True, 'eigen_solver': 'arpack', 'eigen_tol': 0.0, 'eps': 0.0001,
'gamma': 1.0, 'kernel_params': None, 'max_iter': 300, 'mode': 3, 'n_clusters': 2,
'n_components': 2, 'n_init': 10, 'n_jobs': None, 'n_neighbors': 10,
'norm_laplacian': True, 'random_state': None, 'verbose': 0}
get parameters after setting:
{'affinity': 'precomputed', 'assign_labels': 'kmeans', 'coef0': 1, 'degree': 3,
'drop_first': True, 'eigen_solver': 'arpack', 'eigen_tol': 0.0, 'eps': 0.0001,
'gamma': 1.0, 'kernel_params': None, 'max_iter': 300, 'mode': 3, 'n_clusters': 3,
'n_components': 2, 'n_init': 10, 'n_jobs': None, 'n_neighbors': 10,
'norm_laplacian': True, 'random_state': None, 'verbose': 0}
```

Return Value
It simply returns "self" reference.

32.3.1.7 7. 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 a spectral clustering model stored previously from the specified file (having little-endian binary data).

For example,

```
spec.load("./out/MySpecClusteringModel", dtype = np.float64)
```

Return Value
It simply returns "self" reference.

32.3.1.8 8. 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 (metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the spectral clustering model
spec.save("./out/MySpecClusteringModel")
```

This will save the spectral clustering model on the path "/out/MySpecClusteringModel".
It would raise exception if the directory already exists with same name.

The 'MySpecClusteringModel' directory has

MySpecClusteringModel
|——metadata
|——**model**
  |——— aff_type
  |——— **affinity**
  |——— cluster_size
  |——— label

The metadata file contains the number of clusters, number of components, model kind, input datatype used for trained model.
Here, the **model** directory contains information about affinity type, labels, cluster size and **affinity** matrix (sparse or dense, depending upon python/frovedis input and affinity is 'rbf', 'precomputed' or 'nearest_neighbors').

Return Value
It returns nothing.

32.3.1.9 9. debug_print()

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

For example,

```
spec.debug_print()
```

Output

```
affinity matrix:
num_row = 5, num_col = 5
node 0
node = 0, local_num_row = 5, local_num_col = 5, val = 1 0.970446 6.2893e-104 2.92712e-106
1.28299e-108 0.970446 1 1.27264e-101 6.2893e-104 2.92712e-106 6.2893e-104 1.27264e-101 1
0.970446 0.88692 2.92712e-106 6.2893e-104 0.970446 1 0.970446 1.28299e-108 2.92712e-106
0.88692 0.970446 1
labels:
0 0 1 1 1
ncluster: 2
```

This output will be visible on server side. It dispays the affinity matrix.

No such output will be visible on client side.

Return Value
It returns nothing.

32.3.1.10 10. release()

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

For example,

```
spec.release()
```

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

Return Value
It returns nothing.

32.3.1.11 11. is_fitted()

Purpose
It can be used to confirm if the model is already fitted or not.

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

32.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**

- **Introduction to FrovedisCRSMatrix**
- **Agglomerative Clustering in Frovedis**
- **DBSCAN in Frovedis**
- **KMeans in Frovedis**

Chapter 33

Spectral Embedding

33.1 NAME

Spectral Embedding - One of 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.

33.2 SYNOPSIS

```
class frovedis.mllib.manifold.SpectralEmbedding(n_components=2, affinity='nearest_neighbors',
                                                gamma=1.0, random_state=None, eigen_solver=None,
                                                n_neighbors=None, n_jobs=None, verbose=0,
                                                norm_laplacian=True, mode=3, drop_first=True)
```

33.2.1 Public Member Functions

fit(X, y = None)
load(fname, dtype = None)
save(fname)
get_params(deep = True)
set_params(**params)
debug_print()
release()
is_fitted()

33.3 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.

This module provides a client-server implementation, where the client application is a normal python program. Frovedis is almost same as Scikit-learn manifold module providing Spectral Embedding 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 Spectral Embedding 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, the 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.

33.3.1 Detailed Description

33.3.1.1 1. SpectralEmbedding()

Parameters
n_components: An integer parameter containing the number of component count. (Default: 2)
affinity: A string object parameter which specifies how to construct the affinity matrix. (Default: 'nearest_neighbors')
- **'nearest_neighbors'**: construct the affinity matrix by computing a graph of nearest neighbors.
- **'precomputed'**: interpret X as a precomputed affinity matrix, where larger values indicate greater similarity between instances.

Only 'nearest_neighbors' and 'precomputed' are supported.
gamma: The double (float64) parameter required for computing nearby relational meaningful eigenvalues. (Default: 1.0)
random_state: An unused parameter. (Default: None)
eigen_solver: An unused parameter. (Default: None)
n_neighbors: An unused parameter. (Default: None)
n_jobs: An unused parameter. (Default: None)
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.
norm_laplacian: A boolean parameter if set to True, then computes the normalized Laplacian. (Default: True)
mode: An integer parameter required to set the eigen computation method. It can be either 1 (for generic) or 3 (for shift-invert). (Default: 3)
drop_first: A boolean parameter if set to True, then drops the first eigenvector. The first eigenvector of a normalized Laplacian is full of constants, thus if it is set to true, then (n_components + 1) eigenvectors are computed and will drop the first vector. Otherwise, it will calculate 'n_components' number of eigenvectors. (Default: True)

Attribute
affinity_matrix:

1. **For python native dense input:**
   - When affinity = 'precomputed', it returns a numpy array
   - When affinity = 'nearest_neighbors', it returns a numpy array
2. **For frovedis-like dense input:**

- When affinity = 'precomputed', returns a FrovedisRowmajorMatrix
- When affinity = 'nearest_neighbors', returns a FrovedisRowmajorMatrix

3. **For python native sparse input:**
   - When affinity = 'precomputed', it returns a scipy matrix
   - When affinity = 'nearest_neighbors', it returns a numpy array
4. **For frovedis-like sparse input:**
   - When affinity = 'nearest_neighbors', it a returns a FrovedisRowmajorMatrix
   - When affinity = 'precomputed', it returns a FrovedisRowmajorMatrix

In all cases, the output is of float or double (float64) type and of shape **(n_samples, n_samples)**.

embedding_:

1. **For python native dense input:**
   - When affinity = 'precomputed', it returns a numpy array
   - When affinity = 'nearest_neighbors', it returns a numpy array
2. **For frovedis-like dense input:**
   - When affinity = 'precomputed', returns a FrovedisRowmajorMatrix
   - When affinity = 'nearest_neighbors', returns a FrovedisRowmajorMatrix
3. **For python native sparse input:**
   - When affinity = 'precomputed', it returns a numpy array
   - When affinity = 'nearest_neighbors', it returns a numpy array
4. **For frovedis-like sparse input:**
   - When affinity = 'nearest_neighbors', it a returns a FrovedisRowmajorMatrix
   - When affinity = 'precomputed', it returns a FrovedisRowmajorMatrix

In all cases, the output is of float or double (float64) type and of shape **(n_samples, n_components)**.

Note: affinity = 'precomputed' should be used with square matrix input only, Otherwise, it throws an exception.

Purpose
It initializes a Spectral Embedding object with the given parameters.

The parameters: 'random_state', 'eigen_solver', 'n_neighbors' and 'n_jobs' are simply kept in to make the interface uniform to the Scikit-learn Spectral Embedding module. They are not used anywhere within frovedis implementation.

After getting the affinity matrix by computing distance co-relation, this is used to extract meaningful patterns in high dimensional data.

Return Value
It simply returns "self" reference.

33.3.1.2 2. fit(X, y = 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 of float or double (float64) type. It has shape **(n_samples, n_features)**.

y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose

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

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("spectral_data.txt")

# fitting input matrix on Spectral Embedding object
from frovedis.mllib.manifold import SpectralEmbedding
sem = SpectralEmbedding(n_components = 2, drop_first = True).fit(train_mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading sample matrix data
train_mat = np.loadtxt("spectral_data.txt")

# Since "train_mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(train_mat)

# fitting input matrix on Spectral Embedding object
from frovedis.mllib.manifold import SpectralEmbedding
sem = SpectralEmbedding(n_components = 2, drop_first = True).fit(rmat)
```

Return Value
It simply returns "self" reference.

33.3.1.3 3. 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
sem.load("./out/MySemModel",dtype=np.float64)
```

Return Value
It simply returns "self" instance.

33.3.1.4 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(metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
sem.save("./out/MySemModel")
```

The MySemModel contains below directory structure:
MySemModel
|——metadata
|——**model**
|——aff_type
|——**affinity**
|——**embedding**

'metadata' represents the detail about n_components, model_kind and datatype of training vector. Here, the **model** directory contains information about type of affinity matrix, affinity matrix and embedding matrix.

If the directory already exists with the same name then it will raise an exception.

Return Value
It returns nothing.

33.3.1.5 5. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by Spectral Embedding. It is used to get parameters and their values of Spectral Embedding class.

For example,

```
print(sem.get_params())
```

Output

```
{'affinity': 'nearest_neighbors', 'drop_first': True, 'eigen_solver': None, 'gamma': 1.0,
'mode': 3, 'n_components': 2, 'n_jobs': None, 'n_neighbors': None, 'norm_laplacian': True,
'random_state': None, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

33.3.1.6 6. set_params(**params)

Parameters
****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by Spectral Embedding, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(sem.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
sem.set_params(n_components = 3, drop_first = False)
print("get parameters after setting:")
print(sem.get_params())
```

Output

```
get parameters before setting:
{'affinity': 'nearest_neighbors', 'drop_first': True, 'eigen_solver': None,
'gamma': 1.0, 'mode': 3, 'n_components': 2, 'n_jobs': None, 'n_neighbors': None,
'norm_laplacian': True, 'random_state': None, 'verbose': 0}
get parameters after setting:
{'affinity': 'nearest_neighbors', 'drop_first': False, 'eigen_solver': None,
'gamma': 1.0, 'mode': 3, 'n_components': 3, 'n_jobs': None, 'n_neighbors': None,
'norm_laplacian': True, random_state': None, 'verbose': 0}
```

Return Value
It simply returns "self" reference.

33.3.1.7 7. debug_print()

Purpose
It shows the target model information (affinity and embed matrix) on the server side user terminal. It is mainly used for debugging purpose.

For example,

```
sem.debug_print()
```

Output

```
affinity matrix:
num_row = 5, num_col = 5
node 0
node = 0, local_num_row = 5, local_num_col = 5, val = 1 0.970446 6.2893e-104 2.92712e-106
1.28299e-108 0.970446 1 1.27264e-101 6.2893e-104 2.92712e-106 6.2893e-104 1.27264e-101
1 0.970446 0.88692 2.92712e-106 6.2893e-104 0.970446 1 0.970446 1.28299e-108 2.92712e-106
0.88692 0.970446 1
embed matrix:
num_row = 5, num_col = 2
node 0
node = 0, local_num_row = 5, local_num_col = 2, val = -0.628988 -0.345834 -0.628988
-0.345834 -0.202594 0.368471 -0.202594 0.368471 -0.202594 0.368471
```

This output will be visible on server side. It displays the affinity matrix and embedding matrix on the trained model which is currently present on the server.

No such output will be visible on client side.

Return Value
It returns nothing.

33.3.1.8 8. release()

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

For example,

```
sem.release()
```

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

Return Value
It returns nothing.

33.3.1.9 9. is_fitted()

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

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

33.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Spectral Clustering in Frovedis**

Chapter 34

StandardScaler

34.1 NAME

StandardScaler - It is a preprocessing technique which is used to perform scaling of distribution values so that mean of observed values is zero and the standard deviation is unit variance.

34.2 SYNOPSIS

```
frovedis.mllib.preprocessing.StandardScaler(copy = True, with_mean= True,
                                            with_std=True, sam_std = False, verbose=0)
```

34.2.1 Public Member Functions

fit(X, y = None, sample_weight = None)
fit_transform(X, y = None)
get_params(deep = True)
inverse_transform(X, copy = None)
is_fitted()
partial_fit(X, y = None, sample_weight = None)
release()
set_params(**params)
transform(X, copy = None)

34.3 DESCRIPTION

This machine learning algorithm standardizes the features by removing the mean and scaling to unit variance.

The standard score of a sample x is calculated as:

```
z = (x - u) / s
```

where,
- u is the mean of the training samples or zero if with_mean = False.
- s is the standard deviation of the training samples or one if with_std = False.

Centering and scaling happens independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used later on during **transform()**.

Currently, this class in frovedis will first create a copy of input data and then will perform standard scaling. It does not support inplace scaling yet.

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 StandardScaler 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 StandardScaler 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 transform-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.

34.3.1 Detailed Description

34.3.1.1 1. StandardScaler()

Parameters
copy: This is an unused parameter. (Default: True)
with_mean: It is a boolean parameter that specifies whether to perform centering before scaling of data. (Default: True)
When it is True (not specified explicitly), it will center the data before scaling.
Note: Also, use 'with_mean = False' when attempting centering on sparse matrices. Otherwise it raises an exception.
When it is set as None (specified explicitly), it will be set as False.
with_std: It is a boolean parameter that specifies whether to scale the data to unit variance or not. (Default: True)
When it is True (not specified explicitly), it will scale the data to unit variance.
When it is set as None (specified explicitly), it will be set as False.
sam_std: It is a boolean parameter that specifies whether to enable unbiased or biased sample standard deviation. (Default: False)

- If it is False (not specified explicitly), it will compute biased standard deviation (where **1 / n_samples** is used).
- If it is True (specified explicitly), it will compute unbiased standard deviation (where **1 / ('n_samples' - 1)** is used).

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.

Attributes
mean_: It is a numpy ndarray of shape **(n_features,)** and having double (float64) type values. It specifies the mean value for each feature in the training set.
var_: It is a numpy ndarray of shape **(n_features,)** and having double (float64) type values. It specifies the variance for each feature in the training set.
scale_: It is a numpy ndarray of shape **(n_features,)** and having double (float64) type values. It specifies the per feature relative scaling of the data to achieve zero mean and unit variance.

Purpose
It initializes a StandardScaler object with the given parameters.

The parameters: "copy" is simply kept in to make the interface uniform to the Scikit-learn StandardScaler module. They are not used in frovedis implementation internally.

Return Value
It simply returns "self" reference.

34.3.1.2 2. fit(X, y = None, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float (float32) or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float (float32) or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.
sample_weight: None or any python array-like object (any shape). It is simply ignored in frovedis implementation.

Purpose
It computes the mean and standard deviation to be used for later scaling.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# fitting input matrix on StandardScaler object
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler().fit(mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# StandardScaler with pre-constructed frovedis-like inputs
```

```
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler().fit(rmat)
```

Return Value
It simply returns "self" reference.

34.3.1.3 3. fit_transform(X, y = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float (float32) or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float (float32) or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation.

Purpose
It will fit the model with input matrix (X) and then returns a transformed version of input matrix (X).

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# fitting input matrix on StandardScaler object and perform transform
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler()
print(ss.fit_transform(mat))
```

Output

```
[[-1.23598774 -1.23598774 -1.23598774]
 [-1.21318353 -1.21318353 -1.21318353]
 [ 0.79358622  0.79358622  0.79358622]
 [ 0.81639042  0.81639042  0.81639042]
 [ 0.83919462  0.83919462  0.83919462]]
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)
```

```
# Fitting StandardScaler with pre-constructed frovedis-like inputs and perform transform
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler()
print(ss.fit_transform(rmat))
```

Output

```
[[-1.23598774 -1.23598774 -1.23598774]
 [-1.21318353 -1.21318353 -1.21318353]
 [ 0.79358622  0.79358622  0.79358622]
 [ 0.81639042  0.81639042  0.81639042]
 [ 0.83919462  0.83919462  0.83919462]]
```

Return Value

- **When dense data is used as input:**
  For both frovedis-like input and python input, it returns a numpy matrix of shape **(n_samples, n_features)** and double (float64) type values. It contains transformed values.

- **When sparse data is used as input:**
  For both frovedis-like input and python input, it returns a scipy sparse matrix of shape **(n_samples, n_features)** and double (float64) type values. It contains transformed values.

34.3.1.4 4. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by StandardScaler. It is used to get parameters and their values of StandardScaler class.

For example,

```
print(ss.get_params())
```

Output

```
{'copy': True, 'sam_std': False, 'verbose': 0, 'with_mean': True, 'with_std': True}
```

Return Value
A dictionary of parameter names mapped to their values.

34.3.1.5 5. inverse_transform(X, copy = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float (float32) or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajor-Matrix for dense data of float (float32) or double (float64) type. It has shape **(n_samples, n_features)**.
copy: This is an unsed parameter.

Purpose
It scales back the data to the original representation.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# fitting input matrix on StandardScaler object and perform transform
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(mat)
X1 = ss.transform(mat)

# inverse_transform() demo to get original input data
print(ss.inverse_transform(X1))
```

Output

```
[[0.1 0.1 0.1]
 [0.2 0.2 0.2]
 [9.  9.  9. ]
 [9.1 9.1 9.1]
 [9.2 9.2 9.2]]
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# StandardScaler with pre-constructed frovedis-like inputs and perform transform
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(rmat)
X1 = ss.transform(rmat)

# inverse_transform() demo to get original frovedis-like input data
print(ss.inverse_transform(X1))
```

Output

```
[[0.1 0.1 0.1]
 [0.2 0.2 0.2]
```

```
 [9.  9.  9. ]
 [9.1 9.1 9.1]
 [9.2 9.2 9.2]]
```

Return Value

- **When dense data is used as input:**
  For both frovedis-like input and python input, it returns a numpy matrix of shape **(n_samples, n_features)** and double (float64) type values. It contains transformed values (original data).

- **When sparse data is used as input:**
  For both frovedis-like input and python input, it returns a scipy sparse matrix of shape **(n_samples, n_features)** and double (float64) type values. It contains transformed values (original data).

34.3.1.6 6. is_fitted()

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

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

34.3.1.7 7. partial_fit(X, y = None, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float (float32) or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float (float32) or double (float64) type. It has shape **(n_samples, n_features)**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.
sample_weight: None or any python array-like object (any shape). It is simply ignored in frovedis implementation.

Purpose
It performs incremental computation of mean and standard deviation on each new batch of samples present in input matrix (X) and the finalised mean and standard deviation computed is later used for scaling.

All of input matrix (X) is processed as a single batch. This is intended for cases when fit is not feasible due to very large number of 'n_samples' or because input matrix (X) is read from a continuous stream.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# partial_fit() demo with input matrix on StandardScaler object
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler().partial_fit(mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# StandardScaler with pre-constructed frovedis-like inputs
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler().partial_fit(rmat)
```

Return Value
It simply returns "self" reference.

34.3.1.8 8. release()

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

For example,

```
ss.release()
```

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

Return Value
It returns nothing.

34.3.1.9 9. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by StandardScaler, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(ss.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
ss.set_params(with_mean = False, sam_std = False, verbose = 1)
print("get parameters after setting:")
print(ss.get_params())
```

Output

```
get parameters before setting:
{'copy': True, 'sam_std': False, 'verbose': 0, 'with_mean': True, 'with_std': True}
get parameters after setting:
{'copy': True, 'sam_std': False, 'verbose': 1, 'with_mean': False, 'with_std': True}
```

Return Value
It simply returns "self" reference.

34.3.1.10 10. transform(X, copy = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float (float32) or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float (float32) or double (float64) type. It has shape **(n_samples, n_features)**.
copy: This is an unsed parameter.

Purpose
It performs standardization by centering and scaling.

It will always perform transform on a copy of input matrix (X).

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
                 [9.2, 9.2, 9.2]])

# fitting input matrix on StandardScaler object and perform transform
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(mat)
print(ss.transform(mat))
```

Output

```
[[-1.23598774 -1.23598774 -1.23598774]
 [-1.21318353 -1.21318353 -1.21318353]
 [ 0.79358622  0.79358622  0.79358622]
 [ 0.81639042  0.81639042  0.81639042]
 [ 0.83919462  0.83919462  0.83919462]]
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0.1, 0.1, 0.1],
                 [0.2, 0.2, 0.2],
                 [9., 9., 9. ],
                 [9.1, 9.1, 9.1],
```

```
                [9.2, 9.2, 9.2]])

# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# Fitting StandardScaler with pre-constructed frovedis-like inputs and perform transform
from frovedis.mllib.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(rmat)
print(ss.transform(rmat))
```

Output

```
[[-1.23598774 -1.23598774 -1.23598774]
 [-1.21318353 -1.21318353 -1.21318353]
 [ 0.79358622  0.79358622  0.79358622]
 [ 0.81639042  0.81639042  0.81639042]
 [ 0.83919462  0.83919462  0.83919462]]
```

Return Value

- **When dense data is used as input:**
  For both frovedis-like input and python input, it returns a numpy matrix of shape **(n_samples, n_features)** and double (float64) type values. It contains transformed values.

- **When sparse data is used as input:**
  For both frovedis-like input and python input, it returns a scipy sparse matrix of shape **(n_samples, n_features)** and double (float64) type values. It contains transformed values.

34.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**

- **Introduction to FrovedisCRSMatrix**

Chapter 35

SVC

35.1 NAME

SVC (Support Vector Classification) - A classification algorithm used to predict the binary output with different kernel functions.

35.2 SYNOPSIS

```
class frovedis.mllib.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0,
                             shrinking=True, probability=False, tol=0.001, cache_size=128,
                             class_weight=None, verbose=False, max_iter=-1,
                             decision_function_shape='ovr', break_ties=False,
                             random_state=None)
```

35.2.1 Public Member Functions

fit(X, y, sample_weight = None)
predict(X)
predict_proba(X) load(fname, dtype = None)
save(fname)
score(X, y, sample_weight = None)
get_params(deep = True)
set_params(**params)
release()
is_fitted()

35.3 DESCRIPTION

Support vector machines (SVMs) are a set of supervised learning methods used for classification and regression.

During training, the input X is the training data and y are their corresponding label values (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. **Frovedis supports only binary SVC classification algorithm**.

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 SVC (Support Vector Classification) 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 SVC 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.

35.3.1 Detailed Description

35.3.1.1 1. SVC()

Parameters
C: A positive double (float64) parameter, also called as inverse of regularization strength. (Default: 1.0)
kernel: A string object parameter that specifies the kernel type to be used. Unlike sklearn, frovedis supports 'linear', 'poly', 'rbf' and 'sigmoid' kernel function. (Default: 'rbf')
degree: A positive integer parameter specifying the degree of the polynomial kernel. (Default: 3)
gamma: A string object parameter that specifies the kernel coefficient. (Default: 'scale')
For all python like input gamma ='scale' will be used, otherwise 'auto' will be used.
Depending on the **kernel coefficient**, **gamma** will be calculated as:
- If **gamma = 'scale'** (default) is used then **gamma = 1.0 / (n_features * variance),**
- If **gamma = 'auto'** is used then **gamma = 1.0 / n_features**

coef0: A double (float64) parameter which is independent term in kernel function. (Default: 0.0)
shrinking: An unused parameter. (Default: 'True')
probability: An unused parameter. (Default: 'False')
tol: A zero or positive double (float64) parameter specifying the convergence tolerance value. (Default: 0.001)
cache_size: An interger parameter that specifies the size of the kernel cache(in megabytes). It must be greater than 2. (Default: 128)
class_weight: An unused parameter. (Default: 'None')
verbose: A boolean parameter that specifies the log level to use. (Default: 'False')
Its value is False by default (for INFO mode). But it can be set to True (for DEBUG mode or TRACE mode) for getting training time logs from frovedis server.
max_iter: A positive integer parameter specifying the hard limit on iterations within solver, or -1 for no limit. (Default: -1)
decision_function_shape: An unused parameter. (Default: 'ovr')
break_ties: An unused parameter. (Default: 'False')
random_state: An unused parameter. (Default: 'None')

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)**.
This attribute is supported only for 'linear' kernel, otherwise it will be None.
intercept_: It is a python ndarray(float or double (float64) values depending on input matrix data type) and has shape **(1,)**. It specifies the constants in decision function.
This attribute is supported only for 'linear' kernel, otherwise it will be None.
classes_: It is an int64 type python ndarray of unique labels given to the classifier during training. It has shape **(n_classes,)**.

support_: It is an int32 type python ndarray that specifies the support vectors indices. It has shape **(n_support_vectors_,)**, where **n_support_vectors_ = len(support vectors) / n_features**
support_vectors_: It is double (float64) type python ndarray specifies the support vectors. It has shape **(n_support_vectors_, n_features)**.

Purpose
It initializes a SVC object with the given parameters.

The parameters: "shrinking", "probability", "class_weight", "decision_function_shape", "break_ties"and "random_state" are simply kept to make the interface uniform to Scikit-learn SVC module. They are not used anywhere within frovedis implementation.

Return Value
It simply returns "self" reference.

35.3.1.2 2. fit(X, y, sample_weight = None)

Parameters
X: A numpy dense or a scipy sparse matrix or any python ndarray. It has shape **(n_samples, n_features)**.
For Frovedis input data:
- if kernel = 'linear', then it can be an instance of FrovedisColmajorMatrix for dense data and FrovedisCRS-Matrix for sparse data.
- if kernel != 'linear', then it can be an instance of FrovedisRowmajorMatrix.

y: Any python array-like object or an instance of FrovedisDvector containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), 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 SVC model at frovedis server.

Depending upon the input data, usage of of 'gamma' will vary:
- For **python** dense data, **gamma = 'scale'** and **gamma = 'auto'** will be used.
- For **python** sparse input, **gamma = 'auto'** will be used.
- For all **frovedis** like input, **gamma = 'auto'** only will be used.

For example,

```
# loading a sample matrix and labels data
mat = np.array([[-1.0, -1.0], [-2.0, -1.0], [1.0, 1.0], [2.0, 1.0]])
lbl = np.array([1.0, 1.0, 2.0, 2.0])

# fitting input matrix and label on SVC object
from frovedis.mllib.svc import SVC
svc = SVC(gamma ='auto').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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample matrix and labels data
mat = np.array([[-1.0, -1.0], [-2.0, -1.0], [1.0, 1.0], [2.0, 1.0]])
lbl = np.array([1.0, 1.0, 2.0, 2.0])
```

```
# Since "mat" is numpy dense data and kernel = 'rbf' by default,
we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
from frovedis.matrix.dvector import FrovedisDvector
rmat = FrovedisRowmajorMatrix(mat)
dlbl = FrovedisDvector(lbl)

# SVC with pre-constructed frovedis-like inputs
from frovedis.mllib.svm import SVC
svc = SVC(gamma ='auto').fit(rmat, dlbl)
```

User can also provide FrovedisColmajorMatrix or FrovedisCRSMatrix as a training data but only with **kernel = 'linear'** and **gamma ='auto'**.

```
# loading a sample matrix and labels data
mat = np.array([[-1.0, -1.0], [-2.0, -1.0], [1.0, 1.0], [2.0, 1.0]])
lbl = np.array([1.0, 1.0, 2.0, 2.0])

# Since "mat" is numpy dense data and kernel = 'linear', 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)

# SVC with pre-constructed frovedis-like inputs
from frovedis.mllib.svm import SVC
svc = SVC(kernel = 'linear', gamma = 'auto').fit(cmat, dlbl)
```

Return Value
It simply returns "self" reference.

35.3.1.3 3. predict(X)

Parameters
X: A numpy dense or scipy sparse matrix or any python ndarray 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,

```
svc.predict(mat)
```

Output:

```
[1. 1. 2. 2.]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# predicting on SVC using frovedis-like input
```

```
svc.predict(rmat)
```

Output

```
[1. 1. 2. 2.]
```

Return Value
It returns a numpy array of int64, float or double (float64) type containing the predicted outputs. It has shape **(n_samples,)**.

35.3.1.4 4. predict_proba(X)

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

Purpose
Perform classification on an array and return probability estimates for the test vector X.

It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server. Unlike sklearn, it performs the classification on an array and returns the probability estimates for the test feature matrix (X).

This method is not available for kernel = 'linear'.

For example,

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

Output

```
[[6.17597327e-04 9.99382403e-01]
 [1.98150882e-24 1.00000000e+00]
 [7.31058579e-01 2.68941421e-01]
 [9.38039081e-01 6.19609192e-02]]
```

Like in fit(), frovedis-like input can be used to speed-up the prediction making on the trained model at server side.

For example,

```
# Since "mat" is numpy dense data, we have created FrovedisRowmajorMatrix.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# finds the probablity sample for each class in the SVC model
svc.predict_proba(rmat)
```

Output

```
[[6.17597327e-04 9.99382403e-01]
 [1.98150882e-24 1.00000000e+00]
 [7.31058579e-01 2.68941421e-01]
 [9.38039081e-01 6.19609192e-02]]
```

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

35.3.1.5 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 SVC model
svc.load("./out/SVCModel")
```

Return Value
It simply returns "self" reference.

35.3.1.6 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 (label_map, metadata and model) in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# saving the model
svc.save("./out/SVCModel")
```

The **SVCModel** contains below directory structure:

SVCModel
|——label_map
|——metadata
|——model

'label_map' contains information about labels mapped with their encoded value.
'metadata' represents the detail about numbre of classes, model_kind and datatype of training vector.
Here, the model file contains information about model in binary format.

This will save the SVC model on the path '/out/SVCModel'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

35.3.1.7 7. score(X, y, sample_weight = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python ndarray or an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data. It has shape **(n_samples, n_features)**.
y: Any python ndarray containing the target labels. It has shape **(n_samples,)**.
sample_weight: A python ndarray containing the intended weights for each input samples and it should be the shape of **(n_samples,)**. (Default: None)
When it is None (not specified explicitly), 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
svc.score(mat, lbl)
```

Output

```
1.0
```

Return Value
It returns an accuracy score of double (float64) type.

35.3.1.8 8. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by SVC. It is used to get parameters and their values of SVC class.

For example,

```
print(svc.get_params())
```

Output

```
{'C': 1.0, 'break_ties': False, 'cache_size': 128, 'class_weight': None, 'coef0': 0.0,
'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 0.5, 'kernel': 'rbf',
'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True,
'tol': 0.001, 'verbose': False}
```

Return Value
A dictionary of parameter names mapped to their values.

35.3.1.9 9. set_params(**params)

Parameters
*****params***: All the keyword arguments are passed this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by SVC, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(svc.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
svc.set_params( kernel = "linear", shrinking = False)
print("get parameters after setting:")
print(svc.get_params())
```

Output

```
get parameters before setting:
{'C': 1.0, 'break_ties': False, 'cache_size': 128, 'class_weight': None, 'coef0': 0.0,
'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 0.5, 'kernel': 'rbf',
'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True,
'tol': 0.001, 'verbose': False}
get parameters after setting:
{'C': 1.0, 'break_ties': False, 'cache_size': 128, 'class_weight': None, 'coef0': 0.0,
'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 0.5, 'kernel': 'linear',
'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': False,
'tol': 0.001, 'verbose': False}
```

Return Value
It simply returns "self" reference.

35.3.1.10 10. release()

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

For example,

```
svc.release()
```

This will reset the after-fit populated attributes (like coef_, intercept_, classes_, etc) to None, along with releasing server side memory.

Return Value
It returns nothing.

35.3.1.11 11. 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'.

35.4 SEE ALSO

- **Introduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisColmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **LinearSVC in Frovedis**
- **LinearSVR in Frovedis**

Chapter 36

t-Distributed Stochastic Neighbor Embedding

36.1 NAME

TSNE - It's full form is T-distributed Stochastic Neighbor Embedding. It is an unsupervised algorithm primarily used for data exploration and visualizing high-dimensional data.

36.2 SYNOPSIS

```
frovedis.mllib.manifold.tsne(n_components=2, perplexity=30.0,
                             early_exaggeration=12.0, learning_rate=200.0,
                             n_iter=1000, n_iter_without_progress=300,
                             min_grad_norm=1e-7, metric="euclidean",
                             init="random", verbose=0, random_state=None,
                             method="exact", angle=0.5, n_jobs=None)
```

36.2.1 Public Member Functions

fit(X, y = None)
fit_transform(X, y = None)
get_params(deep = True)
set_params(**params)

36.3 DESCRIPTION

It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two or three dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.

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 TSNE 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 TSNE 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 tranform-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.

36.3.1 Detailed Description

36.3.1.1 1. TSNE()

Parameters
n_components: It is an integer parameter that specifies the dimension of the embedded space. (Default: 2) Currently, it supports **'n_components = 2'** only as parameter value.
perplexity: It must be a positive double (float64) parameter that specifies the number of nearest neighbors. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. Different values can result in significantly different results. (Default: 30.0)
early_exaggeration: It must be a positive double (float64) parameter that controls how tight natural clusters in the original space are in the embedded space and how much space will be between them. For larger values, the space between natural clusters will be larger in the embedded space. (Default: 12.0)
learning_rate: It accepts the word 'auto' or a positive double (float64) as parameter value that controls the step size of the gradient updates. It is usually in the range **[10.0, 1000.0]**. **Currently, 'auto' as parameter value cannot be used.** (Default: 200.0)

- **If the 'learning_rate' is too high**: then the data may look like a **'ball'** when plotted on a graph with any point approximately equidistant from its nearest neighbours.

- **If the 'learning_rate' is too low**: then most data points may look compressed in a **dense cloud** with few outliers when plotted on a graph.

n_iter: It must be a positive integer value that specifies the maximum number of iterations for the optimization. It must be atleast 250. (Default: 1000)
n_iter_without_progress: It is an integer parameter that specifies the maximum number of iterations without progress before we abort the optimization. (Default: 300)
min_grad_norm: It is a double (float64) parameter that specifies whether in case the gradient norm is below this threshold, then the optimization will be stopped. (Default: 1e-7)
metric: It is a string object parameter that specifies the metric to use when calculating distance between instances in a feature array. It supports **'euclidean'** or **'precomputed'** distances. (Default: 'euclidean')
init: It is a string object parameter that specifies the initialization of embedding. (Default: 'random')
Currently, only random initialization is supported for this method in frovedis.
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: This is an unused parameter. (Default: None)
method: It is a string object parameter that specifies the t-SNE implementation method to use. (Default: 'exact')

- **'exact'**: it calculates the pair-wise distance between every pair of data points. **Currently, only exact implementation of tsne is supported in frovedis.**

- **'barnes_hut'**: it calculates the distance between each data point and its closest neighboring points only. **Currently, this implementation of tsne is not supported in frovedis.**

angle: This is an unused parameter. (Default: 0.5)
n_jobs: This is an unused parameter. (Default: None)

Attributes
n_iter_: It is a positive integer value that specifies the number of iterations run.
kl_divergence_: It is a double (float64) type value that specifies the Kullback-Leibler divergence after optimization.
embedding_: It is a numpy ndarray of double (float64) type values or FrovedisRowmajorMatrix instance, having shape **(n_samples, n_components)**, where **n_samples** is the number of samples in the input matrix (X). It stores the embedding vectors.

Purpose
It initializes a TSNE object with the given parameters.

The parameters: "random_state", "angle", "n_jobs" are simply kept in to make the interface uniform to the Scikit-learn TSNE module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

36.3.1.2 2. fit(X, y = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float or double (float64) type. It has shape **(n_samples, n_features)**. **If metric = 'precomputed'**, then input matrix (X) is assumed to be a **squared distance matrix**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It will fit input matrix (X) into an embedded space.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0, 0, 0, 0], [0, 1, 1, 1],
                 [1, 0, 1, 0], [1, 1, 1, 0],
                 [1, 1, 1, 1]], dtype = np.float64)

# fitting input matrix on TSNE object
from frovedis.mllib.manifold import TSNE
tsne = TSNE().fit(mat)
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0, 0, 0, 0], [0, 1, 1, 1],
                 [1, 0, 1, 0], [1, 1, 1, 0],
                 [1, 1, 1, 1]], dtype = np.float64)

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

```
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# TSNE with pre-constructed frovedis-like inputs
from frovedis.mllib.manifold import TSNE
tsne = TSNE().fit(rmat)
```

Return Value
It simply returns "self" reference.

36.3.1.3 3. fit_transform(X, y = None)

Parameters
X: A numpy dense or scipy sparse matrix or any python array-like object of int, float or double (float64) type values. It can also be an instance of FrovedisCRSMatrix for sparse data and FrovedisRowmajorMatrix for dense data of float or double (float64) type. It has shape **(n_samples, n_features)**. **If metric = 'precomputed'**, then input matrix (X) is assumed to be a **squared distance matrix**.
y: None or any python array-like object (any shape). It is simply ignored in frovedis implementation, like in Scikit-learn.

Purpose
It will fit input matrix (X) into an embedded space and will return the transformed output.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0, 0, 0, 0], [0, 1, 1, 1],
                 [1, 0, 1, 0], [1, 1, 1, 0],
                 [1, 1, 1, 1]], dtype = np.float64)

# fitting input matrix on TSNE object and perform transform
from frovedis.mllib.manifold import TSNE
tsne = TSNE()
print(tsne.fit_transform(mat))
```

Output

```
[[ -15.15708437 -218.36584687]
 [  31.18096218   62.89769738]
 [ -79.72509032  -59.56844016]
 [ 129.59496966 -137.71666144]
 [ 195.56724491   14.28454464]]
```

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, especially when same data would be used for multiple executions.

For example,

```
# loading a sample numpy dense data
import numpy as np
mat = np.matrix([[0, 0, 0, 0], [0, 1, 1, 1],
                 [1, 0, 1, 0], [1, 1, 1, 0],
                 [1, 1, 1, 1]], dtype = np.float64)

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

```
# For scipy sparse data, FrovedisCRSMatrix should be used instead.
from frovedis.matrix.dense import FrovedisRowmajorMatrix
rmat = FrovedisRowmajorMatrix(mat)

# TSNE with pre-constructed frovedis-like inputs and perform transform
from frovedis.mllib.manifold import TSNE
embedding = TSNE().fit_transform(rmat))
embedding.debug_print()
```

Output

```
matrix:
num_row = 5, num_col = 2
node 0
node = 0, local_num_row = 5, local_num_col = 2, val = -15.1571 -218.366 31.181 62.8977 -79.7251
-59.5684 129.595 -137.717 195.567 14.2845
```

Return Value

- **When X is python native input:**
  It returns a python ndarray of shape **(n_samples, n_components)** and double (float64) type values. It contains the embedding of the training data in low-dimensional space.

- **When X is frovedis-like input:**
  It returns a FrovedisRowmajorMatrix instance of shape **(n_samples, n_components)** and double (float64) type values, containing the embedding of the training data in low-dimensional space.

36.3.1.4 4. get_params(deep = True)

Parameters
deep: A boolean parameter, used to get parameters and their values for an estimator. If True, it will return the parameters for an estimator and contained subobjects that are estimators. (Default: True)

Purpose
This method belongs to the BaseEstimator class inherited by TSNE. It is used to get parameters and their values of TSNE class.

For example,

```
print(tsne.get_params())
```

Output

```
{'angle': 0.5, 'early_exaggeration': 12.0, 'init': 'random', 'learning_rate': 200.0,
'method': 'exact', 'metric': 'euclidean', 'min_grad_norm': 1e-07, 'n_components': 2,
'n_iter': 1000, 'n_iter_without_progress': 300, 'n_jobs': None, 'perplexity': 30.0,
'random_state': None, 'verbose': 0}
```

Return Value
A dictionary of parameter names mapped to their values.

36.3.1.5 5. set_params(**params)

Parameters
****params***: All the keyword arguments are passed to this function as dictionary. This dictionary contains parameters of an estimator with its given values to set.

Purpose
This method belongs to the BaseEstimator class inherited by TSNE, used to set parameter values.

For example,

```
print("get parameters before setting:")
print(tsne.get_params())
# User just needs to provide the arguments and internally it will create a
dictionary over the arguments given by user
tsne.set_params(perplexity = 15, metric = 'precomputed')
print("get parameters after setting:")
print(tsne.get_params())
```

Output

```
get parameters before setting:
{'angle': 0.5, 'early_exaggeration': 12.0, 'init': 'random', 'learning_rate': 200.0,
'method': 'exact', 'metric': 'euclidean', 'min_grad_norm': 1e-07, 'n_components': 2,
'n_iter': 1000, 'n_iter_without_progress': 300, 'n_jobs': None, 'perplexity': 30.0,
'random_state': None, 'verbose': 0}
get parameters after setting:
{'angle': 0.5, 'early_exaggeration': 12.0, 'init': 'random', 'learning_rate': 200.0,
'method': 'exact', 'metric': 'precomputed', 'min_grad_norm': 1e-07, 'n_components': 2,
'n_iter': 1000, 'n_iter_without_progress': 300, 'n_jobs': None, 'perplexity': 15.0,
'random_state': None, 'verbose': 0}
```

Return Value
It simply returns "self" reference.

Note: In order to release the embedding vector from the server (generated by TSNE algorithm in frovedis), we can use release() of FrovedisRowmajorMatrix class.

For example,

```
tsne.embedding_.release()
```

This will remove the embedding vector from the server side memory.

36.4 SEE ALSO

- **Intorduction to FrovedisRowmajorMatrix**
- **Introduction to FrovedisCRSMatrix**
- **Introduction to FrovedisDvector**
- **Spectral Embedding in frovedis**

Chapter 37

Word2Vec

37.1 NAME

Word2Vec - Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

37.2 SYNOPSIS

```
class frovedis.mllib.feature.Word2Vec(sentences=None, corpusFile=None,
                                      outDirPath=None, hiddenSize=100,
                                      learningRate=0.025, n_iter=1,
                                      minCount=5, window=5, threshold=1e-3,
                                      negative=5, modelSyncPeriod=0.1, minSyncWords=1024,
                                      fullSyncTimes=0, messageSize=1024, numThreads=None)
```

37.2.1 Public Member Functions

build_vocab(corpusIterable = None, corpusFile = None, outDirPath = None, update = False)
build_vocab_and_dump(corpusIterable = None, corpusFile = None, outDirPath = None, update = False)
to_gensim_model()
train(corpusIterable = None, corpusFile = None)
fit(corpusIterable = None, corpusFile = None)
save(modelPath, binary = False)
transform(corpusIterable = None, corpusFile = None, func = None)
fit_transform(corpusIterable = None, corpusFile = None, func = None)

37.3 DESCRIPTION

Word2vec is a two-layer neural net that processes text by "vectorizing" words. Its input is a text corpus and its output is a set of vectors (feature vectors that represent words in that corpus). While Word2vec is not a deep neural network, it turns text into a numerical form that deep neural networks can understand.

Word2vec's applications extend beyond parsing sentences in the wild. It can be applied just as well to genes, code, likes, playlists, social media graphs and other verbal or symbolic series in which patterns may be discerned.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis interface is almost same as Gensim Word2Vec interface. It needs to be used when a system has Gensim 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 Word2Vec 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 transform-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.

37.3.1 Detailed Description

37.3.1.1 1. Word2Vec()

Parameters
sentences: This parameter is an iterable of list of strings. If not provided, the 'corpusFile' parameter must be provided for creating the model. (Default: None)
corpusFile: A string object parameter which specifies the path to a corpus file. If not provided, the 'sentences' parameter must be provided for creating the model. (Default: None)
outDirPath: A string object parameter specifying the path of output directory. The newly built vocabulary generated from the input data file is dumped into the provided output file path. (Default: None)
hiddenSize: An integer parameter specifying the dimensionality of the word vectors. For fast computation, this value should be an even value and less than 512 during training. (Default: 100)
learningRate: An unused parameter specifying the initial learning rate. (Default: 0.025)
n_iter: An unused parameter specifying the number of iterations over the corpus. (Default: 1)
minCount: An integer parameter that ignores all words with total frequency lower than this value during vocabulary building. This value should be within 1 to largest total frequency of a word in vocabulary. (Default: 5)
window: An integer parameter specifying the maximum distance between the current and predicted word within a sentence. For fast computation, this value should be less than or equal to 8 during training. (Default: 5)
threshold: An unused parameter specifying the threshold for configuring which higher-frequency words are randomly downsampled. (Default: 1e-3)
negative: An integer parameter. This value specifies how many "noise words" should be drawn (usually between 5-20). The default value should be used for fast computation during training. (Default: 5)
modelSyncPeriod: An unused parameter specifying the model synchronous period. (Default: 0.1)
minSyncWords: An unused parameter specifying the minimum number of words to be synced at each model sync. (Default: 1024)
fullSyncTimes: An unused parameter specifying the full-model sync-up time during training. (Default: 0)
messageSize: An unused parameter specifying the message size in megabytes. (Default: 1024)
numThreads: An integer parameter specifying the number of worker threads to be used to train the model. Ideally, the number of threads should range between 1 and 8. (Default: None)
In case the environment variable 'VE_OMP_NUM_THREADS' is defined, then the value for 'numThreads' would be derived from VE_OMP_NUM_THREADS, otherwise, 'numThreads' would default to value 1.
Attributes
wv: The trained words are stored in KeyedVectors (mapping between keys such as words and embeddings as arrays) instance. It contains the mapping between words and embeddings.

For example,

```
# Using an iterable data
data = [["cat", "say", "meow"], ["dog", "say", "woof"]]

# Training the Word2Vec model using build_vocab() and train()
from frovedis.mllib import Word2Vec
wv_model = Word2Vec(minCount = 2)
wv_model.build_vocab(data)
wv_model.train(data)
print(wv_model.wv)
```

Output

```
{'</s>': array([ 0.00400269,  0.0044194 , -0.00383026, ...,  0.0015239 ,  0.00305939,
    0.00019363]), 'say': array([ 0.00227966, -0.00495255,  0.00431488, ...,  0.00019882,
    -0.0016861 , -0.00112656])}
```

It displays words as keys and embeddings as arrays.

Purpose
It initializes a Word2Vec object with the given parameters.

The parameters: "learningRate", "n_iter", "threshold", "modelSyncPeriod", "minSyncWords", "fullSyncTimes" and "messageSize" are simply kept in to to make the interface uniform to the gensim Word2Vec module. They are not used anywhere within the frovedis implementation.

Return Value
It simply returns "self" reference.

37.3.1.2 2. build_vocab(corpusIterable = None, corpusFile = None, outDirPath = None, update = False)

Parameters
corpusIterable: Use the 'sentences' iterable which itself must be an iterable of list of list of tokens. If None, the 'corpusFile' must be provided for building the vocabulary. (Default: None)
corpusFile: A string object parameter specifying the path of a corpus file. If None, the 'corpusIterable' must be provided for building the vocabulary. (Default: None)
outDirPath: A string object parameter specifying the path of output directory './out'. The newly built vocabulary generated from the input data file is dumped into provided output file path. (Default: None)
update: A boolean parameter if set to True, will add the new words present in sentences to the model's vocabulary. (Default: False)

Purpose
It builds the vocabulary from input data file and dumped into provided output files. It also initializes the 'wv' attribute.

For example, building vocabulary from an iterable

```
# Using an iterable data
data = [["cat", "say", "meow"], ["dog", "say", "woof"]]

# Building vocabulary from input iterable data
from frovedis.mllib import Word2Vec
wv_model = Word2vec(minCount = 2)
wv_model.build_vocab(corpusIterable = data)
```

For example, building vocabulary from a text file

```
# Using a text file
textfile = "./input/text8-10k"
modelpath = "./out/text_model.txt"

# Building vocabulary from input text file
from frovedis.mllib import Word2Vec
wv_model = Word2Vec(minCount = 2)
wv_model.build_vocab(corpusFile = textfile, outDirPath = modelpath)
```

Return Value
It simply returns "self" reference.

37.3.1.3 3. build_vocab_and_dump(corpusIterable = None, corpusFile = None, outDirPath = None, update = False)

Parameters
corpusIterable: Use the 'sentences' iterable which itself must be an iterable of list of list of tokens. If None, the 'corpusFile' must be provided for building the vocabulary. (Default: None)
corpusFile: A string object parameter specifying the path to a corpus file. If None, the 'corpusIterable' must be provided for building the vocabulary. (Default: None)
outDirPath: A string object parameter specifying the path to output directory './out'. The newly build vocabulary generated from the input data file is dumped into provided output file path. (Default: None)
update: A boolean parameter if set to True, will add the new words present in sentences to the model's vocabulary. (Default: False)

Purpose
It builds the vocabulary from input data file and dump into provided output files. It also initializes the 'wv' attribute. This method is an alias to build_vocab().

For example, building vocabulary from an iterable

```
# Using an iterable data
data = [["cat", "say", "meow"], ["dog", "say", "woof"]]

# Building vocabulary from input iterable data
from frovedis.mllib import Word2Vec
wv_model = Word2vec(minCount = 2)
wv_model.build_vocab_and_dump(corpusIterable = data)
```

For example, building vocabulary from a text file

```
# Using a text file
textfile = "./input/text8-10k"
modelpath = "./out/text_model.txt"

# Building vocabulary from input text file
from frovedis.mllib import Word2Vec
wv_model = Word2Vec(minCount = 2)
wv_model.build_vocab_and_dump(corpusFile = textfile, outDirPath = modelpath)
```

Return Value
It simply returns "self" reference.

37.3.1.4 4. to_gensim_model()

Purpose
It generates a gensim like 'wv' (KeyedVectors instance) attribute for Word2Vec model.

For example,

```
wv_model.to_gensim_model()
```

Note:- In order to use this method, gensim version installed by users needs to be 4.0.1 or above.

Return Value
It returns gensim like KeyedVectors instance.

37.3.1.5 5. train(corpusIterable = None, corpusFile = None)

Parameters
corpusIterable: Use the 'sentences' iterable which itself must be an iterable of list of list of tokens. If None, the 'corpusFile' must be provided for building the vocabulary. (Default: None)
corpusFile: A string object parameter specifying the path to a corpus file. If None, the 'corpusIterable' must be provided for building the vocabulary. (Default: None)

Purpose
It trains the Word2Vec model on input vocabulary.

For example, training with an iterable data

```
# Using an iterable data
data = [["cat", "say", "meow"], ["dog", "say", "woof"]]

# Training the model
from frovedis.mllib import Word2Vec
wv_model = Word2vec(minCount = 2)
wv_model.build_vocab(corpusIterable = data)
wv_model.train(corpusIterable = data)
```

For example, training with a text file

```
# Using a text file
textfile = "./input/text8-10k"
modelpath = "./out/text_model.txt"

# Training the model
from frovedis.mllib import Word2Vec
wv_model = Word2Vec(minCount = 2)
wv_model.build_vocab(corpusFile = textfile, outDirPath = modelpath)
wv_model.train(corpusFile = textfile)
```

Return Value
It simply returns "self" reference.

37.3.1.6 6. fit(corpusIterable = None, corpusFile = None)

Parameters
corpusIterable: Use the 'sentences' iterable which itself must be an iterable of list of list of tokens. If None, the 'corpusFile' must be provided for building the vocabulary. (Default: None)
corpusFile: A string object parameter specifying the path to a corpus file. If None, the 'corpusIterable' must be provided for building the vocabulary. (Default: None)

Purpose
It trains the Word2Vec model on input vocabulary. This method is an alias to train().

For example, training with an iterable data

```
# Using an iterable data
data = [["cat", "say", "meow"], ["dog", "say", "woof"]]

# Training the model
from frovedis.mllib import Word2Vec
wv_model = Word2vec(minCount = 2)
wv_model.build_vocab(corpusIterable = data)
wv_model.fit(corpusIterable = data)
```

For example, training with a text file

```
# Using a text file
textfile = "./input/text8-10k"
modelpath = "./out/text_model.txt"

# Training the model
from frovedis.mllib import Word2Vec
wv_model = Word2Vec(minCount = 2)
wv_model.build_vocab(corpusFile = textfile, outDirPath = modelpath)
wv_model.fit(corpusFile = textfile)
```

Return Value
It simply returns "self" reference.

37.3.1.7 7. save(modelPath, binary = False)

Parameters
modelPath: A string object parameter specifying the path of the output file in order to save the embeddings.
binary: A boolean parameter if set to True, will save the data in binary format, otherwise, it will be saved in plain text. (Default: False)

Purpose
It saves the word2vec model information to a file.

On success, it writes the model information (after-fit populated attribute like 'wv') in the specified file as little-endian binary data. Otherwise, it throws an exception.

For example,

```
# To save the word embeddings in a file
model = "./out/text_model.txt"
wv_model.save(model, binary = False)
```

This will save the word2vec model information on the path "/out/text_model.txt".
It would raise exception if the 'text_model.txt' file already existed with same name.

The 'text_model.txt' file contains the count of all the words with total frequency greater than 'minCount', 'hiddenSize' and dictionary having words (as keys) and embeddings (as values).

Return Value
It returns nothing.

37.3.1.8 8. transform(corpusIterable = None, corpusFile = None, func = None)

Parameters
corpusIterable: Use the 'sentences' iterable which itself must be an iterable of list of list of tokens. If None, the 'corpusFile' must be provided for building the vocabulary. (Default: None)
corpusFile: A string object parameter specifying the path to a corpus file. If None, the 'corpusIterable' must be provided for building the vocabulary. (Default: None)
func: A function to apply on document vector. (Default: None) If None, then np.mean() is used as 'func'.

Purpose
It transforms the document text to word2vec embeddings.

For example, training with an iterable

```
# Using an iterable data
data = [["cat", "say", "meow"], ["dog", "say", "woof"]]

# Training the model
from frovedis.mllib import Word2Vec
wv_model = Word2vec(minCount = 2)
wv_model.build_vocab(corpusIterable = data)
wv_model.fit(corpusIterable = data)
embeddings = wv_model.transform(corpusIterable = data)
print(embeddings)
```

Output

```
[[ 0.00227966 -0.00495255  0.00431488 ...  0.00019882 -0.0016861
  -0.00112656]
 [ 0.00227966 -0.00495255  0.00431488 ...  0.00019882 -0.0016861
  -0.00112656]]
```

For example, training with a text file

```
# Using a text file
textfile = "./input/text8-10k"

# Training the model
wv_model = Word2Vec(minCount = 2)
wv_model.build_vocab(corpusFile = textfile)
wv_model.fit(corpusFile = textfile)
embeddings = wv_model.transform(corpusFile = textfile)
print(embeddings)
```

Output

```
[[-0.00013626 -0.00059639  0.00063703 ...  0.0004084  -0.00033636
  -0.00013291]]
```

Return Value
It returns document embeddings (numpy array) of shape **(n_samples, hiddenSize)**.

37.3.1.9 9. fit_transform(corpusIterable = None, corpusFile = None, func = None)

Parameters
corpusIterable: Use the 'sentences' iterable which itself must be an iterable of list of list of tokens. If None, the 'corpusFile' must be provided for building the vocabulary. (Default: None)
corpusFile: A string object parameter specifying the path to a corpus file. If None, the 'corpusIterable'

must be provided for building the vocabulary. (Default: None)
func: A function to apply on document vector. (Default: None) If None, then np.mean() is used as 'func'.

Purpose
It trains the word2vec model on input document and transforms the document text to word2vec embeddings.

For example, training with an iterable

```
# Using an iterable data
data = [["cat", "say", "meow"], ["dog", "say", "woof"]]

# Training the model
from frovedis.mllib import Word2Vec
wv_model = Word2vec(minCount = 2)
wv_model.build_vocab(corpusIterable = data)
embeddings = wv_model.fit_transform(corpusIterable = data)
print(embeddings)
```

Output

```
[[ 0.00227966 -0.00495255  0.00431488 ...  0.00019882 -0.0016861
  -0.00112656]
 [ 0.00227966 -0.00495255  0.00431488 ...  0.00019882 -0.0016861
  -0.00112656]]
```

For example, with a text file

```
# Using a text file
textfile = "./input/text8-10k"

# Training the model
wv_model = Word2Vec(minCount = 2)
wv_model.build_vocab(corpusFile = textfile)
embeddings = wv_model.fit_transform(corpusFile = textfile)
print(embeddings)
```

Output

```
[[-0.00013626 -0.00059639  0.00063703 ...  0.0004084  -0.00033636
  -0.00013291]]
```

Return Value
It returns document embeddings (numpy array) of shape **(n_samples, hiddenSize)**.

Chapter 38

linalg

38.1 NAME

linalg - a frovedis module which provides user-friendly interfaces for commonly used linear algebra functions.

38.2 SYNOPSIS

```
import frovedis.linalg.linalg
```

38.3 Public Member Functions

```
1. dot(a, b, out=None)
2. eigsh(A, M=None, k=6, sigma=None, which='LM', v0=None, ncv=None, maxiter=None,
         tol=0., return_eigenvectors=True, Minv=None, OPinv=None, mode='normal')
3. inv(a)
4. matmul(x1, x2, out=None, casting='same_kind', order='K', dtype=None,
          subok=True, signature=None, extobj=None)
5. solve(a, b)
6. svd(a, full_matrices=False, compute_uv=True)
```

38.4 DESCRIPTION

The frovedis linear algebra functions rely on PBLAS and ScaLAPACK wrappers in frovedis to provide efficient low level implementations of standard linear algebra algorithms. These functions are used to compute matrices and vector product, matrix decompostion, solving linear equations, inverting matrices, etc.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis interface for all linear algebra functions is almost same as numpy linear algebra function interfaces, but it doesn't have any dependency on numpy. It can be used simply even if the system doesn't have numpy installed. Thus, in this implementation, a python client can interact with a frovedis server sending the required python matrix data to frovedis server side. Python data is converted into frovedis compatible data (blockcyclic matrix) internally and then python client can request frovedis server for any of the supported linalg functions on that matrix. Once the operation is completed, the frovedis server sends back the resultant matrix as equivalent python data.

38.4.1 Detailed Description

38.4.1.1 1. dot(a, b, out = None)

Parameters
a: It accepts scalar values and python array-like inputs (having dimensions <= 2) of int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
b: It accepts scalar values, and python array-like inputs (having dimensions <= 2) of int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
out: A numpy ndarray of int, float (float32) or double (float64) type values which is where the result is written. (Default: None)
When it is None (not specified explicitly), a freshly-allocated array having double (float64) type values is returned. Otherwise if provided, it must have a shape that matches the signature (n,k),(k,m)->(n,m) and same datatype as input matrices.

Purpose
It computes dot product of two arrays.

1. If both inputs 'a' and 'b' are scalar values,

For example,

```
# dot() demo with scalar values as input
from frovedis.linalg import dot
prod = dot(3,4)
print(prod)
```

Output

```
12
```

Here, result is also a scalar value.

2. If both 'a' and 'b' are 1D arrays,

For example,

```
import numpy as np
from frovedis.linalg import dot
vec1 = np.array([1., 9., 8.])
vec2 = np.array([4., 1., 7.])

# dot() demo with both inputs as 1D array
prod = dot(vec1,vec2)
print(prod)
```

Output

```
69.0
```

Here, dot product is the inner product of vectors.

3. If either 'a' or 'b' is a scalar value,

For example,

```
import numpy as np
from frovedis.linalg import dot
vec1 = np.array([[1, 9, 8], [7, 1, 5], [1, 2, 4]])
```

```
# dot() demo with 1D array and scalar value as input
prod = dot(vec1,4)
print(prod)
```

Output

```
[[ 4. 36. 32.]
 [28.  4. 20.]
 [ 4.  8. 16.]]
```

Here, result is multiplication of scalar over the vector.

4. If both a and b are 2D arrays,

For example,

```
import numpy as np
from frovedis.linalg import dot
mat1 = np.array([[1, 9, 8], [7, 1, 5], [1, 2, 4]])
mat2 = np.array([[4, 1, 7], [2, 2, 9], [1, 2, 3]])

# dot() demo with 2D arrays as input
prod = dot(mat1, mat2)
print(prod)
```

Output

```
[[ 30.  35. 112.]
 [ 35.  19.  73.]
 [ 12.  13.  37.]]
```

Here, result is simply matrix multiplication.

When both 'a' and 'b' inputs are matrices or instances of FrovedisBlockCyclicMatrix, then the column size of 'a' must be same as row size of 'b' before computing dot product as internally it performs matrix-matrix multiplication.

5. If a is an ND array and b is a 1-D array,

Currenlty, frovedis supports only N <= 2, so that dot() between ND array and 1D array can be interpreted currently as:
- dot product between both 1D arrays.
- dot product between 2D array and 1D array.

For example,

```
import numpy as np
from frovedis.linalg import dot
mat1 = np.array([[1, 9, 8], [7, 1, 5], [1, 2, 4]])
mat2 = np.array([4, 2, 1])

# dot() demo with 'a' as 2D array, 'b' as 1D array as input
prod = dot(mat1, mat2)
print(prod)
```

Output

```
[30. 35. 12.]
```

In above example, result will be the sum product over the last axis of 'a' and 'b'.

Currently, dot product between inputs 'a' (an N-D array) and 'b' (an M-D array (where M>=2) is not supported in frovedis.

For example,

```
import numpy as np
from frovedis.linalg import dot
mat1 = np.array([[1., 9., 8.], [7., 1., 5.], [1., 2., 4.]])
mat2 = np.array([[4., 1., 7.], [2., 2., 9.], [1., 2., 3.]])

# dot() demo with out != None
prod = dot(mat1, mat2, out = np.array([[1., 1., 1.],[1., 1., 1.],[1., 1., 1.]]))
print(prod)
```

Output

```
[[ 30.  35. 112.]
 [ 35.  19.  73.]
 [ 12.  13.  37.]]
```

Here, the result will be written in the user provided ndarray and shape and datatype same as input matrix.

For example,

```
import numpy as np
from frovedis.linalg import dot
mat1 = np.array([[1, 9, 8], [7, 1, 5], [1, 2, 4]])
mat2 = np.array([[4, 1, 7], [2, 2, 9], [1, 2, 3]])

# creating FrovedisBlockcyclicMatrix instance
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
A_Bmat = FrovedisBlockcyclicMatrix(mat1, np.float64)
B_Bmat = FrovedisBlockcyclicMatrix(mat2, np.float64)

# dot() demo with FrovedisBlockcyclicMatrix instance
prod = dot(A_Bmat, B_Bmat)
prod.debug_print()
```

Output

```
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 30 35 12 35 19 13 112 73 37
```

This output will be only visible on server side terminal.

Return Value
It returns the dot product of 'a' and 'b'.
If 'a' and 'b' are both scalars or both 1-D arrays then a scalar is returned. Otherwise, an array is returned.
If out is given, then it is returned.
If either 'a' or 'b' is a FrovedisBlockCyclicMatrix instance, then it returns a FrovedisBlockCyclicMatrix instance.

38.4.1.2 2. eigsh(A, M = None, k = 6, sigma = None, which = 'LM', v0 = None, ncv = None, maxiter = None, tol = 0., return_eigenvectors = True, Minv = None, OPinv = None, mode = 'normal')

Parameters
A: 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. The input matrix provided must be squared symmetric in nature.
M: This is an unused parameter. (Default: None)
k: A positive integer parameter that specifies the number of eigenvalues and eigenvectors desired. It must be in range 0 to N, where N is the number of rows in squared symmetric matrix. (Default: 6)
sigma: It accepts a float (float32) type parameter that is used to find eigenvalues near sigma using shift-invert mode. Currently, it is only supported for dense matrices. (Default: None)
When it is None (not specified explicitly), it will be set as largest possible value of float (float32) type.
which: A string object parameter that specifies which **k** eigenvectors and eigenvalues to find:
- **'LM'** : Largest (in magnitude) eigenvalues. It is the default value.
- **'SM'** : Smallest (in magnitude) eigenvalues.
- **'LA'** : Largest (algebraic) eigenvalues.
- **'SA'** : Smallest (algebraic) eigenvalues.
- **'BE'** : Half (k/2) from each end of the spectrum.

When sigma != None, 'which' refers to the shifted eigenvalues.
v0: This is an unused parameter. (Default: None)
ncv: This is an unused parameter. (Default: None)
maxiter: A positive integer that specifies the maximum number of iterations allowed. The convergance may occur on or before the maximum iteration limit is reached. In case, the convergance does not occur before the iteration limit, then this method will not generate the resultant **'k'** eigenvalues and eigenvectors. (Default: None)
When it is None (not specified explicitly), it will be set as 10 * N, where N is the number of rows in squared symmetric matrix.
tol: A float (float32) type parameter that specifies the tolerance values used to find relative accuracy for eigenvalues (stopping criterion). (Default: 0.)
return_eigenvectors: A boolean parameter that specifies whether to return eigenvectors in addition to eigenvalues. (Default: True)
When it is set to False, it will only return **k** eigenvalues.
Minv: This is an unused parameter. (Default: None)
OPinv: This is an unused parameter. (Default: None)
mode: A string object parameter that specifies the strategy to use for shift-invert mode. This parameter applies only when sigma != None. This parameter can be **'normal'**, **'cayley'** or **'buckling' modes**. **Currently, frovedis supports only normal mode**. (Default: 'normal')
The choice of mode will affect which eigenvalues are selected by the 'which' parameter.

Purpose
It finds **k** eigenvalues and eigenvectors of the symmetric square matrix A.

The parameters: "M", "vc0", "ncv", "Minv" and "OPinv" are simply kept in to to make the interface uniform to the scipy.sparse.linalg.eigsh() module. They are not used anywhere within the frovedis implementation.

For example,

```
# dense square symmetric matrix
import numpy as np
mat = np.asarray([[2.,-1.,0.,0.,-1.,0.], [-1.,3.,-1.,0.,-1.,0.], [0.,-1.,2.,-1.,0.,0.],
                  [0.,0.,-1.,3.,-1.,-1], [-1.,-1.,0.,-1.,3.,0], [0.,0.,0.,-1.,0.,1.]])

# eigsh() demo
from frovedis.linalg import eigsh
eigen_vals, eigen_vecs = eigsh(mat, k = 3)
print("frovedis computed eigen values:\n")
print(eigen_vals)
print("\nfrovedis computed eigen vectors:\n")
print(eigen_vecs)
```

Output

```
frovedis computed eigen values:

[3.         3.70462437 4.89121985]

frovedis computed eigen vectors:

[[-0.28867513  0.56702343  0.03232265]
 [-0.28867513 -0.65812747  0.46850031]
 [-0.28867513  0.20514371 -0.35640753]
 [ 0.57735027  0.30843449  0.56195221]
 [ 0.57735027 -0.30843449 -0.56195221]
 [-0.28867513 -0.11403968 -0.14441544]]
```

Since, input dense matrix has shape (3,3), so 0< k <=3.

For example,

```
# eigsh() demo when sigma != None
eigen_vals, eigen_vecs = eigsh(mat, k = 3, sigma = -1)
print("frovedis computed eigen values:\n")
print(eigen_vals)
print("\nfrovedis computed eigen vectors:\n")
print(eigen_vecs)
```

Output

```
frovedis computed eigen values:

[0.         0.72158639 1.68256939]

frovedis computed eigen vectors:

[[ 0.40824829  0.41486979 -0.50529469]
 [ 0.40824829  0.30944167  0.04026854]
 [ 0.40824829  0.0692328   0.75901025]
 [ 0.40824829 -0.22093352  0.20066454]
 [ 0.40824829  0.22093352 -0.20066454]
 [ 0.40824829 -0.79354426 -0.29398409]]
```

Here, the resultant eigenvalues and eigenvectors are generated using shift invert mode.

For example,

```
# eigsh() demo where which = 'SM'
eigen_vals, eigen_vecs = eigsh(mat, k = 3, which = 'SM')
print("frovedis computed eigen values:\n")
print(eigen_vals)
print("\nfrovedis computed eigen vectors:\n")
print(eigen_vecs)
```

Output

```
frovedis computed eigen values:

[2.70826973e-16 7.21586391e-01 1.68256939e+00]

frovedis computed eigen vectors:
```

```
[[ 0.40824829  0.41486979  0.50529469]
 [ 0.40824829  0.30944167 -0.04026854]
 [ 0.40824829  0.0692328  -0.75901025]
 [ 0.40824829 -0.22093352 -0.20066454]
 [ 0.40824829  0.22093352  0.20066454]
 [ 0.40824829 -0.79354426  0.29398409]]
```

Here, the resultant eigenvalues and eigenvectors are smallest in magnitute for the given dense input matrix.

For example,

```
# eigsh() demo where maxiter = 10
eigen_vals, eigen_vecs = eigsh(mat, k = 3, maxiter = 10)
print("frovedis computed eigen values:\n")
print(eigen_vals)
print("\nfrovedis computed eigen vectors:\n")
print(eigen_vecs)
```

Output

```
frovedis computed eigen values:

[3.         3.70462437 4.89121985]

scipy computed eigen vectors:

[[-0.28867513  0.56702343  0.03232265]
 [-0.28867513 -0.65812747  0.46850031]
 [-0.28867513  0.20514371 -0.35640753]
 [ 0.57735027  0.30843449  0.56195221]
 [ 0.57735027 -0.30843449 -0.56195221]
 [-0.28867513 -0.11403968 -0.14441544]]
```

The eigenvalues and eigenvectors will be generated only when convergence occurs.
In above example, this will happen on or before maxiter = 10 is reached.

For example,

```
# eigsh() demo where return_eigenvectors = False
eigen_vals = eigsh(mat, k = 3, return_eigenvectors = False)
print("frovedis computed eigen values:\n")
print(eigen_vals)
```

Output

```
frovedis computed eigen values:

[3.         3.70462437 4.89121985]
```

Here, only eigenvalues are returned by this method.

Return Value
1. **If return_eigenvectors = True:**
- It returns eigenvectors in addition to eigenvalues as numpy ndarrays having double (float64) type values. Also, eigenvectors has shape **(k,1)** and eigenvalues has shape **(M,k)**, where **M** is the number of rows of input matrix 'A'.
2. **If return_eigenvectors = False:**
- It returns only eigenvalues as numpy ndarray having shape **(k,1)** and double (float64) type values.

38.4.1.3 3. inv(a)

Parameters
a: It accepts a python array-like input or numpy matrix (having dimensions <= 2) of int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values. It specifies the matrix to be inverted. It must be a square matrix of shape **(M, M)** , where **M** is the number of rows and columns of input squared matrix.

Purpose
This method computes the inverse matrix of a given matrix.

Internally it uses **getrf()** and **getri()** methods present in **frovedis.linalg.scalapack** module.

For example,

```
import numpy as np
from frovedis.linalg import inv

# input array contains integer values
mat = np.array([[1, 2], [3, 4]])

# inv(0 demo with numpy array as input
matinv = inv(mat)
print(matinv)
```

Output

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

It returns an ndarray as the inverse of given matrix having double (float64) type values.

For example,

```
import numpy as np
from frovedis.linalg import inv

# input array contains float (float320 type values
mat = np.array([[1., 2.], [3., 4.]], dtype = np.float32)

# inv() demo with numpy array as input
matinv = inv(mat)
print(matinv)
```

Output

```
[[-2.0000002   1.0000001 ]
 [ 1.5000001  -0.50000006]]
```

It returns an ndarray as the inverse of given matrix having float (float32) type values.

For example,

```
import numpy as np
from frovedis.linalg import inv

# input array contains double (float64) type values
mat = np.array([[1., 2.], [3., 4.]])

# inv(0 demo with numpy array as input
matinv = inv(mat)
```

```
print(matinv)
```

Output

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

It returns an ndarray as the inverse of given matrix having double (float64) type values.

For example,

```
import numpy as np
from frovedis.linalg import inv

# creating FrovedisBlockcyclicMatrix instance
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat = np.array([[1., 2.], [3., 4.]])
A_Bmat = FrovedisBlockcyclicMatrix(mat)

# inv() demo FrovedisBlockcyclicMatrix instance
matinv = inv(A_Bmat)
matinv.debug_print()
```

Output

```
matrix:
num_row = 2, num_col = 2
node 0
node = 0, local_num_row = 2, local_num_col = 2, type = 2, descriptor = 1 1 2 2 2 2 0 0 2 2 2
val = -2 1.5 1 -0.5
```

This output will be only visible on server side terminal.

Return Value
1. **When numpy array-like input is used**:
- It returns a numpy ndarray having float (float32) or double (float64) type values and has shape **(M, M)**, where **M** is number of rows and columns of input ndarray.
2. **When input is a numpy matrix**:
- It returns a numpy matrix having float (float32) or double (float64) type values and has shape **(M, M)**, where **M** is number of rows and columns of input matrix.
3. **When input is a FrovedisBlockCyclicMatrix instance**:
- It returns an instance of FrovedisBlockCyclicMatrix.

38.4.1.4 4. matmul(x1, x2, out = None, casting = 'same_kind', order = 'K', dtype = None, subok = True, signature = None, extobj = None)

Parameters
x1: It accepts a python array-like input or numpy maxtrix (having dimensions <= 2) of int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values. It must not be scalar values.
x2: It accepts scalar values, python array-like inputs or numpy maxtrix (having dimensions <= 2) of int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values. It must not be scalar values.
out: A numpy ndarray of int, float or double (float64) type values which is where the result is written. (Default: None)
When it is None (not specified explicitly), a freshly-allocated array having double (float64) type values is returned. Otherwise if provided, it must have a shape that matches the signature (n,k),(k,m)->(n,m) and same datatype as input matrices.

casting: This is an unused parameter. (Default: 'same_kind')
order: This is an unused parameter. (Default: 'K')
dtype: This is an unused parameter. (Default: None)
subok: This is an unused parameter. (Default: True)
signature: This is an unused parameter. (Default: None)
extobj: This is an unused parameter. (Default: None)

Purpose
It computes matrix product of two arrays.

The parameters: "casting", "order", "dtype", "subok" and "signature" and "extobj" are simply kept in to to make the interface uniform to the numpy.matmul() module. They are not used anywhere within the frovedis implementation.

For example,

```
import numpy as np
from frovedis.linalg import matmul

# inputs are having integer type values
mat1 = np.array([[1, 9, 8], [7, 1, 5], [1, 2, 4]])
mat2 = np.array([[4, 1, 7], [2, 2, 9], [1, 2, 3]])

# matmul() demo with numpy array as input
prod = matmul(mat1, mat2)
print(prod)
```

Output

```
[[ 30.  35. 112.]
 [ 35.  19.  73.]
 [ 12.  13.  37.]]
```

Here, the result will be written in a newly generated array having double (float64) type values.

For inputs 'mat1' and 'mat2', the column size of 'mat1' must be same as row size of 'mat2' before computing matrix multiplication.

For example,

```
import numpy as np
from frovedis.linalg import matmul

# inputs are having double (float64) type values
mat1 = np.array([[1.5, 9, 8], [7, 1, 5.5], [1, 2.5, 4]])
mat2 = np.array([[4, 1, 7.5], [2.5, 2, 9], [1, 2, 3.5]])

# matmul() demo with numpy array as input
prod = matmul(mat1, mat2)
print(prod)
```

Output

```
[[ 36.5   35.5  120.25]
 [ 36.    20.    80.75]
 [ 14.25  14.    44.  ]]
```

Here, the result will be written in a newly generated array having double (float64) type values.

For inputs having float (float32) type values, the newly generated array will also have float (float32) type values.

For example,

```
import numpy as np
from frovedis.linalg import matmul

mat1 = np.array([[1., 9., 8.], [7., 1., 5.], [1., 2., 4.]])
mat2 = np.array([[4., 1., 7.], [2., 2., 9.], [1., 2., 3.]])

# matmul() demo with out != None
prod = matmul(mat1, mat2, out = np.array([[1.,1.,1.],[1.,1.,1.],[1.,1.,1.]]))
print(prod)
```

Output

```
[[ 30.  35. 112.]
 [ 35.  19.  73.]
 [ 12.  13.  37.]]
```

Here, the result will be written in the user provided ndarray.

For example,

```
import numpy as np
from frovedis.linalg import matmul
mat1 = np.array([[1, 9, 8], [7, 1, 5], [1, 2, 4]])
mat2 = np.array([[4, 1, 7], [2, 2, 9], [1, 2, 3]])

# creating FrovedisBlockcyclicMatrix instance
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
A_Bmat = FrovedisBlockcyclicMatrix(mat1, np.float64)
B_Bmat = FrovedisBlockcyclicMatrix(mat2, np.float64)

# matmul() demo with FrovedisBlockcyclicMatrix instance
prod = matmul(A_Bmat, B_Bmat)
prod.debug_print()
```

Output

```
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 30 35 12 35 19 13 112 73 37
```

This output will be only visible on server side terminal.

Return Value
1. **If any python array like input (having dimensions <= 2) and out = None:**
- It returns a numpy ndarray as output and having float (float32) or double (float64) type values.
2. **If any python array like input (having dimensions <= 2) and out != None:**
- It returns a numpy ndarray as output. Datatype for output values will be same as datatype of input 'x1' such as int, float (float32) or double (float64) type.
3. **If numpy matrix (having dimensions <= 2) and out = None:**
- It returns a numpy matrix as output and having float (float32) or double (float64) type values.
4. **If numpy matrix (having dimensions <= 2) and out != None:**
- It returns a numpy matrix as output. Datatype for output values will be same as datatype of input 'x1' such as int, float (float32) or double (float64) type.
5. **If inputs are FrovedisBlockCyclicMatrix instances:**
- It returns a FrovedisBlockCyclicMatrix instance.

38.4.1.5 5. solve(a, b)

Parameters
a: It accepts a python array-like input or numpy maxtrix (having dimensions <= 2) of int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values. It specifies a coefficient matrix. It must be a square matrix of shape **(nRows, nCols)** , where **nRows = nCols**.
b: It accepts a python array-like inputs or numpy matrix (having dimensions <= 2) of int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values. It specifies the variables to be determined while solving the equation 'Ax = B'.

Purpose
This function solves a linear matrix equation Ax = B, or system of linear scalar equations.

Internally it uses **gesv()** method present in **frovedis.linalg.scalapack** module.

For example,

```
import numpy as np
from frovedis.linalg import solve

# inputs have integer type values
A = np.array([[3,4,3],[1,2,3],[4,2,1]])
B = np.array([[1,1,1], [3,0,2], [8,2,5]])

#solve() demo with numpy array input
X = solve(A,B)
print(X)
```

Output

```
[[ 3.57142857  0.57142857  2.14285714]
 [-4.57142857 -0.07142857 -2.64285714]
 [ 2.85714286 -0.14285714  1.71428571]]
```

Here, it returns an array with solution matrix having double (float64) type values.

For example,

```
import numpy as np
from frovedis.linalg import solve

# inputs have float (float32) type values
A = np.array([[3,4,3],[1,2,3],[4,2,1]], np.float32)
B = np.array([[1,1,1], [3,0,2], [8,2,5]], np.float32)

#solve() demo with numpy array input
X = solve(A,B)
print(X)
```

Output

```
[[ 3.5714288   0.57142854  2.142857  ]
 [-4.571429   -0.07142858 -2.6428573 ]
 [ 2.857143   -0.14285713  1.7142859 ]]
```

Here, it returns an array with solution matrix having float (float32) type values.

For example,

```
import numpy as np
from frovedis.linalg import solve

# inputs have integer type values
A = np.array([[3.,4.,3.],[1.,2.,3.],[4.,2.,1.]])
B = np.array([[1.,1.,1.], [3.,0.,2.], [8.,2.,5.]])

#solve() demo with numpy array input
X = solve(A,B)
print(X)
```

Output

```
[[ 3.57142857  0.57142857  2.14285714]
 [-4.57142857 -0.07142857 -2.64285714]
 [ 2.85714286 -0.14285714  1.71428571]]
```

Here, it returns an array with solution matrix having double (float64) type values.

For example,

```
import numpy as np
from frovedis.linalg import solve
A = np.array([[3,4,3],[1,2,3],[4,2,1]])
B = np.array([[1,1,1], [3,0,2], [8,2,5]])

#creating FrovedisBlockCyclicMatrix instance
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
A_Bmat = FrovedisBlockcyclicMatrix(A, np.float64)
B_Bmat = FrovedisBlockcyclicMatrix(B, np.float64)

#solve() demo with FrovedisBlockCyclicMatrix input
X = solve(A_Bmat,B_Bmat)
X.debug_print()
```

Output

```
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 3.57143 -4.57143 2.85714 0.571429 -0.0714286 -0.142857 2.14286 -2.64286 1.71429
```

It stores result as a FrovedisBlockCyclicMatrix instance.

Also, this output will be only visible on server side terminal.

Return Value
1. **When python array-like input is used**:
- It returns a numpy ndarray having float (float32) or double (float64) type values and shape **(nRows, nCols)**, where **nRows** is number of rows and **nCols** is number of columns of resultant array.
2. **When input is numpy matrix**:
- It returns a numpy matrix having float (float32) or double (float64) type values and shape **(nRows, nCols)**, where **nRows** is number of rows and **nCols** is number of columns of resultant matrix.
3. **When input is a FrovedisBlockCyclicMatrix instance**:
- It returns an instance of FrovedisBlockCyclicMatrix.

38.4.1.6 6. svd(a, full_matrices = False, compute_uv = True)

Parameters
a: It accepts python array-like inputs or numpy matrix (having dimensions <= 2) containing int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
full_matrices: It is a boolean parameter. Currently, it can only be set as False. Also, the unitary 2D arrays 'u' and 'vh' shapes are (M, K) and (K, N), respectively, where **K = min(M, N)**, **M** is the number of rows and **N** is the number of columns of input matrix. (Default: False)
compute_uv: It is a boolean parameter that specifies whether or not to compute unitary arrays 'u' and 'vh' in addition to 1D array of singular values 's'. (Default: True)

Purpose
It computes the factor of an array by Singular Value Decomposition.

In a nutshell, this method performs factorization of matrix into three matrices 'u', 'vh' and 's'.
Here,
- **u** and **vh** are orthogonal matrices.
- **s** is a diagonal matrix of singular values.

This method internally **gesvd()** method present in **frovedis.linalg.scalapack** module.

For example,

```
import numpy as np
from frovedis.linalg import svd

# input is having integer values
mat1 = np.array([[3,4,3],[1,2,3],[4,2,1]])

# svd() demo with numpy 2D array
u, s, vh = svd(mat1)
print(u)
print(s)
print(vh)
```

Output

```
[[-0.73553325 -0.18392937 -0.65204358]
 [-0.42657919 -0.62196982  0.65664582]
 [-0.52632788  0.76113306  0.37901904]]

[7.87764972 2.54031671 0.69958986]

[[-0.60151068  0.73643349  0.30959751]
 [-0.61540527 -0.18005275 -0.76737042]
 [-0.5093734  -0.65210944  0.5615087 ]]
```

Here it returns 'u', 's' and 'vh' as arrays having double (float64) dtype values.

For example,

```
import numpy as np
from frovedis.linalg import svd

# input is having float (float32) type values
mat1 = np.array([[3.,4.,3.],[1.,2.,3.],[4.,2.,1.]], dtype = np.float32)

# svd() demo with numpy 2D array
```

```
u, s, vh = svd(mat1)
print(u)
print(s)
print(vh)
```

Output

```
[[-0.7355336  -0.18392955 -0.65204364]
 [-0.4265793  -0.62197     0.656646  ]
 [-0.5263281   0.7611333   0.3790189 ]]
[7.8776484 2.540317  0.69959  ]
[[-0.60151064  0.73643357  0.30959752]
 [-0.6154053  -0.1800527  -0.76737046]
 [-0.5093734  -0.6521094   0.5615088 ]]
```

Here it returns 'u', 's' and 'vh' as arrays having float (float32) dtype values.

For example,

```
import numpy as np
from frovedis.linalg import svd

# input is having double (float64) type values
mat1 = np.array([[3.,4.,3.],[1.,2.,3.],[4.,2.,1.]])

# svd() demo with numpy 2D array
u, s, vh = svd(mat1)
print(u)
print(s)
print(vh)
```

Output

```
[[-0.73553325 -0.18392937 -0.65204358]
 [-0.42657919 -0.62196982  0.65664582]
 [-0.52632788  0.76113306  0.37901904]]

[7.87764972 2.54031671 0.69958986]

[[-0.60151068  0.73643349  0.30959751]
 [-0.61540527 -0.18005275 -0.76737042]
 [-0.5093734  -0.65210944  0.5615087 ]]
```

Here it returns 'u', 's' and 'vh' as arrays having double (float64) dtype values.

For example,

```
import numpy as np
from frovedis.linalg import svd
mat1 = np.array([[3,4,3],[1,2,3],[4,2,1]])

# svd() demo with compute_uv = False
s = svd(mat1, compute_uv = False)
print(s)
```

Output

```
[7.87764972 2.54031671 0.69958986]
```

Here, it returns only 's' as an array having double (float64) type values.

For example,

```
import numpy as np
from frovedis.linalg import svd
mat1 = np.array([[3,4,3],[1,2,3],[4,2,1]])

# create FrovedisBlockcyclicMatrix instance
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat2 = FrovedisBlockcyclicMatrix(mat1, np.float64)

# svd() demo with FrovedisBlockcyclicMatrix object as input
u, s, vh = svd(mat2)
u.debug_print()
s.debug_print()
vh.debug_print()
```

Output

```
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -0.735533 -0.426579 -0.526328 -0.183929 -0.62197 0.761133 -0.652044 0.656646 0.379019

vector:
[7.87764972 2.54031671 0.69958986]

matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -0.601511 -0.615405 -0.509373 0.736433 -0.180053 -0.652109 0.309598 -0.76737 0.561509
```

This output will be only visible on server side terminal.

For example,

```
import numpy as np
from frovedis.linalg import svd
mat1 = np.array([[3,4,3],[1,2,3],[4,2,1]])

# create FrovedisBlockcyclicMatrix instance
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat2 = FrovedisBlockcyclicMatrix(mat1, np.float64)

# svd() demo with FrovedisBlockcyclicMatrix object as input and compute_cv = False
s = svd(mat2, compute_uv = False)
s.debug_print()
```

Output

```
vector:
[7.87764972 2.54031671 0.69958986]
```

This output will be only visible on server side terminal.

Return Value
1. **When input is numpy array**:
- If compute_cv = True, then it returns a tuple **(u, s, vh)**, where **u**, **s** and **vh** are numpy ndarrays having

float (float32) or double (float64) type values.
- If compute_cv = False, then it returns only array of singular values **s** of float (float32) or double (float64) type.
2. **When input is numpy matrix**:
- If compute_cv = True, then it returns a tuple **(u, s, vh)**, where **u**, **vh** are matrices and **s** is a numpy ndarray having float (float32) or double (float64) type values.
- If compute_cv = False, then it returns only array of singular values **s** of float (float32) or double (float64) type.
3. **When input is a FrovedisBlockcyclicMatrix instance**:
- If compute_cv = True, then it returns a tuple **(u, s, vh)**, where **u**, **vh** are FrovedisBlockcyclicMatrix instances and **s** is FrovedisVector instance.
- If compute_cv = False, then it returns **s** is an instance of FrovedisVector.

38.5 SEE ALSO

- **Scalapack Functions**

Chapter 39

scalapack

39.1 NAME

scalapack - it contains wrapper functions created for low level functions from ScaLAPACK library (present in frovedis).

39.2 SYNOPSIS

```
import frovedis.linalg.scalapack
```

39.3 Public Member Functions

```
1. dgels(a, b, trans='N', lwork=0, overwrite_a=0, overwrite_b=0)
2. gels(a, b, trans='N', lwork=0, overwrite_a=0, overwrite_b=0, dtype=np.float64)
3. sgels(a, b, trans='N', lwork=0, overwrite_a=0, overwrite_b=0)
4. dgesv(a, b, overwrite_a=0, overwrite_b=0)
5. gesv(a, b, overwrite_a=0, overwrite_b=0, dtype=np.float64)
6. sgesv(a, b, overwrite_a=0, overwrite_b=0)
7. dgesvd(a, compute_uv=1, full_matrices=0, lwork=0, overwrite_a=0)
8. gesvd(a, compute_uv=1, full_matrices=0, lwork=0, overwrite_a=0, dtype=np.float64)
9. sgesvd(a, compute_uv=1, full_matrices=0, lwork=0, overwrite_a=0)
10. dgetrf(a, overwrite_a=0)
11. getrf(a, overwrite_a=0, dtype=np.float64)
12. sgetrf(a, overwrite_a=0)
13. dgetri(lu, piv, lwork=0, overwrite_lu=0)
14. getri(lu, piv, lwork=0, overwrite_lu=0, dtype=np.float64)
15. sgetri(lu, piv, lwork=0, overwrite_lu=0)
16. dgetrs(lu, piv, b, trans=0, overwrite_b=0)
17. getrs(lu, piv, b, trans=0, overwrite_b=0, dtype=np.float64)
18. sgetrs(lu, piv, b, trans=0, overwrite_b=0)
```

39.4 DESCRIPTION

In Frovedis, the linalg module contains wrapper for ScaLAPACK (Scalable LAPACK: the LAPACK routines distributed in nature) routines, whereas scipy.linalg contains wrapper for LAPACK routines. Here, the method names, arguments, purpose etc. in frovedis.linalg wrapper routines are similar to scipy.linalg wrapper routines.
In order to use dgetrf() wrapper present in linalg module of both frovedis and scipy:

In frovedis:

```
from frovedis.linalg.scalapack import dgetrf
```

In scipy:

```
from scipy.linalg.lapack import dgetrf
```

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 perform the supported ScaLAPACK operation requested by the python client on that matrix. After request is completed, frovedis server sends back the resultant matrix and it can then create equivalent python data.

We have supported 'overwrite' option in frovedis routine wrappers.

However, unlike scipy.linalg, if 'overwrite' option is enabled and input is a valid ndarray or an instance of FrovedisBlockcyclicMatrix, we would always overwrite (copy-back) to the same (irrespective of its dtype).

This will slow down the overall computation. This should be enabled only when it's wanted to check the intermediate results (like LU factor etc.)

39.4.1 Detailed Description

39.4.1.1 1. dgels(a, b, trans = 'N', lwork = 0, overwrite_a = 0, overwrite_b = 0)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having double (float64) type values.
b: It accepts a python array-like input or right hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having double (float64) type values. It should have number of rows >= max(M,N) and at least 1 column.
trans: It accepts a string object parameter like 'N' or 'T' which specifies if transpose of 'a' is needed to be computed before solving linear equation. If set to 'T', then transpose is computed. (Default: 'N')
lwork: This is an ununsed parameter. (Default: 0)
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise, 'a' would be overwritten with QR or LQ factor. (Default: 0)
overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise, 'b' would be overwritten with the solution matrix. (Default: 0)

Purpose
This method solves overdetermined or underdetermined real linear systems involving a left hand side matrix 'a' or its transpose, using a QR or LQ factorization of 'a'. It is assumed that the matrix 'a' has full rank. It returns the LQ or QR factor, solution matrix with double (float64) precision.

The parameter: "lwork" is simply kept in to to make the interface uniform to the scipy.linalg.lapack.dgels() module. They are not used anywhere within the frovedis implementation.

This method internally uses Scalapack.gels() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# dgels() demo
from frovedis.linalg.scalapack import dgels
rf = dgels(mat1,mat2)
print(rf)
```

Output

```
(array([[-1.41421356,  3.53553391, -1.41421356],
   [-0.41421356, -4.63680925,  4.74464202],
   [ 0.        ,  0.36709178, -7.77742709]]), array([[-7.54901961,  2.68627451, -1.        ],
   [-1.50980392,  1.1372549 ,  0.        ],
   [-0.7254902 ,  0.15686275,  0.        ]]), 0)
```

For example,

```
# dgels() demo and using transpose of a
from frovedis.linalg.scalapack import dgels
rf = dgels(mat1,mat2,trans='T')
print(rf)
```

Output

```
(array([[-1.41421356,  3.53553391, -1.41421356],
   [-0.41421356, -4.63680925,  4.74464202],
   [ 0.        ,  0.36709178, -7.77742709]]), array([[-7.82352941,  3.29411765, -0.70588235],
   [ 1.17647059,  0.29411765,  0.29411765],
   [-1.96078431,  0.50980392, -0.15686275]]), 0)
```

For example,

```
# dgels() demo and overwriting of a with QR or LQ factor
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import dgels
rf = dgels(mat1,mat2,overwrite_a=1)
print('overwritten matrix with LQ or QF factor: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix with LQ or QF factor:
 [[-1.41421356  3.53553391 -1.41421356]
 [-0.41421356 -4.63680925  4.74464202]
 [ 0.          0.36709178 -7.77742709]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, then LQ or QR factor would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# dgels() demo and overwriting of b with solution matrix
print('original matrix mat2: ')
print(mat2)
from frovedis.linalg.scalapack import dgels
rf = dgels(mat1,mat2,overwrite_b=1)
print('overwritten matrix with solution matrix: ')
print(mat2)
```

Output

```
original matrix mat2:
[[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]

overwritten matrix with solution matrix:
[[-7.54901961  2.68627451 -1.        ]
 [-1.50980392  1.1372549   0.        ]
 [-0.7254902   0.15686275  0.        ]]
```

Here, if the input 'mat2' is double (float64) type and overwite is enabled, then solution matrix would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)
bcm2 = FrovedisBlockcyclicMatrix(mat2)

# dgels() demo and a and b as an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import dgels
rf = dgels(bcm1,bcm2)

# Unpacking the tuple
rf[0].debug_print()
rf[1].debug_print()
print(rf[2])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -1.41421 -0.414214 0 3.53553 -4.63681 0.367092 -1.41421 4.74464 -7.77743
matrix:
num_row = 3, num_col = 3
node 0
```

```
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Return Value
1. **If 'a' and 'b' are python inputs such as numpy matrices**:
- It returns a tuple **(lqr, x, stat)** where,
- **lqr**: It is a numpy matrix having double (float64) type values (by default) and containing the QR or LQ factor of input matrix 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **x**: It is also a numpy matrix having double (float64) type values (by default) and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' and 'b' are instances of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lqr, x, stat)** where,
- **lqr**: It returns instance of FrovedisBlockcyclicMatrix containing the QR or LQ factor of input matrix 'a'.
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
Here, the original inputs 'a' and 'b' are not overwritten even when overwrite_a/overwrite_b is enabled.

39.4.1.2 2. gels(a, b, trans = 'N', lwork = 0, overwrite_a = 0, overwrite_b = 0, dtype = np.float64)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
b: It accepts a python array-like input or right hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values. It should have number of rows >= max(M,N) and at least 1 column.
trans: It accepts a string object parameter like 'N' or 'T' which specifies if transpose of 'a' is needed to be computed before solving linear equation. If set to 'T', then transpose is computed. (Default: 'N')
lwork: This is an ununsed parameter. (Default: 0)
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise, 'a' would be overwritten with QR or LQ factor. (Default: 0)
overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise, 'b' would be overwritten with the solution matrix. (Default: 0)
dtype: It specifies the datatype to be used for setting the precision level (single for float32 / double for float64) for the values returned by this method. (Default: np.float64)
Currently, it supports float (float32) or double (float64) datatypes.

Purpose
This method solves overdetermined or underdetermined real linear systems involving a left hand side matrix 'a' or its transpose, using a QR or LQ factorization of 'a'. It is assumed that the matrix 'a' has full rank. It returns the LQ or QR factor, solution matrix for the system of linear equations with single (float32) or double (float64) precision depending on the 'dtype' parameter provided by user.

The parameter: "lwork" is simply kept in to to make the interface uniform with other modules in frovedis. They are not used anywhere within the frovedis implementation.

This method is present only in frovedis.

This method internally uses Scalapack.gels() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
```

```
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# gels() demo
from frovedis.linalg.scalapack import gels
rf = gels(mat1,mat2)
print(rf)
```

Output

```
(array([[-1.41421356,  3.53553391, -1.41421356],
   [-0.41421356, -4.63680925,  4.74464202],
   [ 0.        ,  0.36709178, -7.77742709]]), array([[-7.54901961,  2.68627451, -1.        ],
   [-1.50980392,  1.1372549 ,  0.        ],
   [-0.7254902 ,  0.15686275,  0.        ]]), 0)
```

For example,

```
# gels() demo and using transpose of a
from frovedis.linalg.scalapack import gels
rf = gels(mat1,mat2,trans='T')
print(rf)
```

Output

```
(array([[-1.41421356,  3.53553391, -1.41421356],
   [-0.41421356, -4.63680925,  4.74464202],
   [ 0.        ,  0.36709178, -7.77742709]]), array([[-7.82352941,  3.29411765, -0.70588235],
   [ 1.17647059,  0.29411765,  0.29411765],
   [-1.96078431,  0.50980392, -0.15686275]]), 0)
```

For example,

```
# gels() demo and overwriting of a with QR or LQ factor
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import gels
rf = gels(mat1,mat2,overwrite_a=1)
print('overwritten matrix with LQ or QF factor: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix with LQ or QF factor:
 [[-1.41421356  3.53553391 -1.41421356]
 [-0.41421356 -4.63680925  4.74464202]
 [ 0.          0.36709178 -7.77742709]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, then LQ or QR factor would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# gels() demo and overwriting of b with solution matrix
```

```
print('original matrix mat2: ')
print(mat2)
from frovedis.linalg.scalapack import gels
rf = gels(mat1,mat2,overwrite_b=1)
print('overwritten matrix with solution matrix: ')
print(mat2)
```

Output

```
original matrix mat2:
[[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]

overwritten matrix with solution matrix:
[[-7.54901961  2.68627451 -1.        ]
 [-1.50980392  1.1372549   0.        ]
 [-0.7254902   0.15686275  0.        ]]
```

Here, if the input 'mat2' is double (float64) type and overwite is enabled, then solution matrix would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# gels() demo and specifying the dtypes of output matrices LQ or QR factor and solution matrix
from frovedis.linalg.scalapack import gels
rf = gels(mat1,mat2,dtype=np.float32)
print(rf)
```

Output

```
(array([[-1.4142135 ,  3.535534  , -1.4142137 ],
       [-0.41421354, -4.6368093 ,  4.744642  ],
       [ 0.        ,  0.36709177, -7.777427  ]], dtype=float32),
       array([[-7.5490184 ,  2.686274  , -1.0000001 ],
       [-1.5098035 ,  1.1372546 ,  0.        ],
       [-0.7254901 ,  0.15686269,  0.        ]], dtype=float32), 0)
```

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)
bcm2 = FrovedisBlockcyclicMatrix(mat2)

# gels() demo and a and b as an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import gels
rf = gels(bcm1,bcm2)

# Unpacking the tuple
rf[0].debug_print()
rf[1].debug_print()
print(rf[2])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -1.41421 -0.414214 0 3.53553 -4.63681 0.367092 -1.41421 4.74464 -7.77743
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Note:- 'dtype' for the wrapper function and FrovedisBlockcyclicMatrix instance must be same during computation. Otherwise, it will raise an excpetion.

Return Value
1. **If 'a' and 'b' are python inputs such as numpy matrices and dtype = np.float64**:
- It returns a tuple **(lqr, x, stat)** where,
- **lqr**: It is a numpy matrix having double (float64) type values (by default) and containing the QR or LQ factor of input matrix 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **x**: It is also a numpy matrix having double (float64) type values (by default) and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' and 'b' are python inputs such as numpy matrices and dtype = np.float32**:
- It returns a tuple **(lqr, x, stat)** where,
- **lqr**: It is a numpy matrix having float (float32) type values (by default) and containing the QR or LQ factor of input matrix 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **x**: It is also a numpy matrix having float (float32) type values (by default) and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
3. **If 'a' and 'b' are instances of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lqr, x, stat)** where,
- **lqr**: It returns instance of FrovedisBlockcyclicMatrix containing the QR or LQ factor of input matrix 'a'.
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.

39.4.1.3 3. sgels(a, b, trans = 'N', lwork = 0, overwrite_a = 0, overwrite_b = 0)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values.
b: It accepts a python array-like input or right hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values. It should have number of rows >= max(M,N) and at least 1 column.
trans: It accepts a string object parameter like 'N' or 'T' which specifies if transpose of 'a' is needed to be computed before solving linear equation. If set to 'T', then transpose is computed. (Default: 'N')
lwork: This is an ununsed parameter. (Default: 0)
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise, 'a' would be overwritten with QR or LQ factor. (Default: 0)
overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise, 'b' would be overwritten with the solution matrix. (Default: 0)

Purpose

This method solves overdetermined or underdetermined linear systems involving a left hand side matrix 'a' or its transpose, using a QR or LQ factorization of 'a'. It is assumed that the matrix 'a' has full rank. It returns the LQ or QR factor, solution matrix for the system of linear equations with single (float32) precision.

The parameter: "lwork" is simply kept in to to make the interface uniform to the scipy.linalg.lapack.sgels() module. They are not used anywhere within the frovedis implementation.

This method internally uses Scalapack.gels() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# sgels() demo
from frovedis.linalg.scalapack import sgels
rf = sgels(mat1,mat2)
print(rf)
```

Output

```
(array([[-1.4142135 ,  3.535534  , -1.4142137 ],
   [-0.41421354, -4.6368093 ,  4.744642  ],
   [ 0.        ,  0.36709177, -7.777427  ]], dtype=float32),
   array([[-7.5490184 ,  2.686274  , -1.0000001 ],
   [-1.5098035 ,  1.1372546 ,  0.        ],
   [-0.7254901 ,  0.15686269,  0.        ]], dtype=float32), 0)
```

For example,

```
# sgels() demo and using transpose of a
from frovedis.linalg.scalapack import sgels
rf = sgels(mat1,mat2,trans='T')
print(rf)
```

Output

```
(array([[-1.4142135 ,  3.535534  , -1.4142137 ],
   [-0.41421354, -4.6368093 ,  4.744642  ],
   [ 0.        ,  0.36709177, -7.777427  ]], dtype=float32),
   array([[-7.8235283 ,  3.2941177 , -0.7058824 ],
   [ 1.1764708 ,  0.29411745,  0.29411763],
   [-1.9607842 ,  0.509804  , -0.15686275]], dtype=float32), 0)
```

For example,

```
# sgels() demo and overwriting of a with QR or LQ factor
print('original matrix mat1: ', mat1)
from frovedis.linalg.scalapack import sgels
rf = sgels(mat1,mat2,overwrite_a=1)
print('overwritten matrix with LQ or QF factor: ', mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]
overwritten matrix with LQ or QF factor:
[[-1.41421354  3.53553391 -1.41421366]
```

```
 [-0.41421354 -4.63680935  4.74464178]
 [ 0.          0.36709177 -7.7774272 ]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, then LQ or QR factor would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# sgels() demo and overwriting of b with solution matrix
print('original matrix mat2: ')
print(mat2)
from frovedis.linalg.scalapack import sgels
rf = sgels(mat1,mat2,overwrite_b=1)
print('overwritten matrix with solution matrix: ')
print(mat2)
```

Output

```
original matrix mat2:
[[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]

overwritten matrix with solution matrix:
[[-7.54901838  2.68627405 -1.00000012]
 [-1.50980353  1.1372546   0.        ]
 [-0.72549009  0.15686269  0.        ]]
```

Here, if the input 'mat2' is double (float64) type and overwite is enabled, then solution matrix would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)
bcm2 = FrovedisBlockcyclicMatrix(mat2)

# sgels() demo and a and b as an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import sgels
rf = sgels(bcm1,bcm2)

# Unpacking the tuple
rf[0].debug_print()
rf[1].debug_print()
print(rf[2])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
```

```
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -1.41421 -0.414214 0 3.53553 -4.63681 0.367092 -1.41421 4.74464 -7.77743
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Return Value
1. **If 'a' and 'b' are python inputs such as numpy matrices**:
- It returns a tuple **(lqr, x, stat)** where,
- **lqr**: It is a numpy matrix having float (float32) type values (by default) and containing the QR or LQ factor of input matrix 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **x**: It is also a numpy matrix having float (float32) (by default) type values and containing the solution matrix. In case 'overwrite_b' is enabled then dtype for the matrix will depend on input 'b' dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' and 'b' are instances of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lqr, x, stat)** where,
- **lqr**: It returns an instance of FrovedisBlockcyclicMatrix containing the QR or LQ factor of input matrix 'a'.
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.

39.4.1.4 4. dgesv(a, b, overwrite_a = 0, overwrite_b = 0)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having double (float64) type values.
b: It accepts a python array-like input or right hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having double (float64) type values. It should have number of rows >= the number of rows in 'a' and at least 1 column in it.
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise, 'a' would be overwritten with LU factor. (Default: 0)
overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise, 'b' would be overwritten with the solution matrix. (Default: 0)

Purpose
It solves a system of linear equations, AX = B with a left hand side square matrix, 'a' by computing it's LU factors internally. It returns the LU factor, solution matrix for the system of linear equations with double (float64) precision.

This method internally uses Scalapack.gesv() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# dgesv() demo
from frovedis.linalg.scalapack import dgesv
rf = dgesv(mat1,mat2)
print(rf)
```

Output

```
(array([[  1. ,   0. ,   2. ],
   [ -1. ,   5. ,   2. ],
   [  0. ,   0.6, -10.2]]), <frovedis.matrix.results.GetrfResult object at 0x7f30f7e25208>,
   array([[-7.54901961,  2.68627451, -1.        ],
   [-1.50980392,  1.1372549 ,  0.        ],
   [-0.7254902 ,  0.15686275,  0.        ]]), 0)
```

This resultant tuple of dgesv() wrapper function contains GetrfResult instance in frovedis. Whereas in scipy, it contains an array.
Both here are responsible to hold pivot array information.

For example,

```
# dgesv() demo and overwriting of a with LU factor
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import dgesv
rf = dgesv(mat1,mat2,overwrite_a=1)
print('overwritten matrix with LU factor: ')
print(mat1)
```

Output

```
original matrix mat1:
 [[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix with LU factor:
[[  1.    0.    2. ]
 [ -1.    5.    2. ]
 [  0.    0.6 -10.2]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, then LU factor would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# dgesv() demo and overwriting of b with solution matrix
print('original matrix mat2: ')
print(mat2)
from frovedis.linalg.scalapack import dgesv
rf = dgesv(mat1,mat2,overwrite_b=1)
print('overwritten matrix with solution matrix: ')
print(mat2)
```

Output

```
original matrix mat2:
 [[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]

overwritten matrix with solution matrix:
[[-7.54901961  2.68627451 -1.        ]
 [-1.50980392  1.1372549   0.        ]
```

```
 [-0.7254902   0.15686275  0.        ]]
```

Here, if the input 'mat2' is double (float64) type and overwite is enabled, then solution matrix would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)
bcm2 = FrovedisBlockcyclicMatrix(mat2)

# dgesv() demo and a and b as an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import dgesv
rf = dgesv(bcm1,bcm2)

# Unpacking the tuple
rf[0].debug_print()
print(rf[1])
rf[2].debug_print()
print(rf[3])
```

Output

```
<frovedis.matrix.results.GetrfResult object at 0x7f7ce5798cf8>
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 1 -1 0 0 5 0.6 2 2 -10.2
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Return Value
1. **If 'a' and 'b' are python inputs such as numpy matrices**:
- It returns a tuple **(lu, piv, x, rs_stat)** where,
- **lu**: It is a numpy matrix having double (float64) type values (by default) and containing the LU factor of input 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **piv**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **x**: It is also a numpy matrix having double (float64) type values (by default) and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **rs_stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' and 'b' are instances of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lu, piv, x, rs_stat)** where,
- **lu**: It returns an instance of FrovedisBlockcyclicMatrix containing the LU factor of input 'a'.
- **piv**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **rs_stat**: It returns an integer containing status (info) of native scalapack dgetrs.

39.4.1.5 5. gesv(a, b, overwrite_a = 0, overwrite_b = 0, dtype = np.float64)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
b: It accepts a python array-like input or right hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values. It should have number of rows >= the number of rows in 'a' and at least 1 column in it.
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise, 'a' would be overwritten with LU factor. (Default: 0)
overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise, 'b' would be overwritten with the solution matrix. (Default: 0)
dtype: It specifies the datatype to be used for setting the precision level (single for float32 / double for float64) for the values returned by this method. (Default: np.float64)
Currently, it supports float (float32) or double (float64) datatypes.

Purpose
It solves a system of linear equations, AX = B with a left hand side square matrix, 'a' by computing it's LU factors internally. It returns the LU factor computed using getrf() and solution matrix computed using getrf() for the system of linear equations with float (float32) or double (float64) precision depending on the 'dtype' parameter provided by user.

This method is present only in frovedis.

This method internally uses Scalapack.gesv() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# gesv() demo
from frovedis.linalg.scalapack import gesv
rf = gesv(mat1,mat2)
print(rf)
```

Output

```
(array([[  1. ,   0. ,   2. ],
   [ -1. ,   5. ,   2. ],
   [  0. ,   0.6, -10.2]]), <frovedis.matrix.results.GetrfResult object at 0x7f30f7e25208>,
   array([[-7.54901961,  2.68627451, -1.        ],
   [-1.50980392,  1.1372549 ,  0.        ],
   [-0.7254902 ,  0.15686275,  0.        ]]), 0)
```

For example,

```
# dgesv() demo and overwriting of a with LU factor
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import dgesv
rf = dgesv(mat1,mat2,overwrite_a=1)
print('overwritten matrix with LU factor: ')
print(mat1)
```

Output

```
original matrix mat1:
 [[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix with LU factor:
[[  1.    0.    2. ]
 [ -1.    5.    2. ]
 [  0.    0.6 -10.2]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, then LU factor would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# dgesv() demo and overwriting of b with solution matrix
print('original matrix mat2: ')
print(mat2)
from frovedis.linalg.scalapack import dgesv
rf = dgesv(mat1,mat2,overwrite_b=1)
print('overwritten matrix with solution matrix: ')
print(mat2)
```

Output

```
original matrix mat2:
 [[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]

overwritten matrix with solution matrix:
[[-7.54901961  2.68627451 -1.        ]
 [-1.50980392  1.1372549   0.        ]
 [-0.7254902   0.15686275  0.        ]]
```

Here, if the input 'mat2' is double (float64) type and overwite is enabled, then solution matrix would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# gesv() demo and specifying the dtypes of output matrices LU factor and solution matrix
from frovedis.linalg.scalapack import gesv
rf = gesv(mat1,mat2,dtype=np.float32)
print(rf)
```

Output

```
(array([[  1. ,   0. ,   2. ],
   [ -1. ,   5. ,   2. ],
   [  0. ,   0.6, -10.2]], dtype=float32),
   <frovedis.matrix.results.GetrfResult object at 0x7f03d450ad30>,
   array([[-7.54902   ,  2.6862745 , -1.        ],
   [-1.509804  ,  1.137255  ,  0.        ],
   [-0.7254902 ,  0.15686277,  0.        ]], dtype=float32), 0)
```

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)
bcm2 = FrovedisBlockcyclicMatrix(mat2)

# dgesv() demo and a and b as an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import dgesv
rf = dgesv(bcm1,bcm2)

# Unpacking the tuple
rf[0].debug_print()
print(rf[1])
rf[2].debug_print()
print(rf[3])
```

Output

```
<frovedis.matrix.results.GetrfResult object at 0x7f7ce5798cf8>
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 1 -1 0 0 5 0.6 2 2 -10.2
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Note:- 'dtype' for the wrapper function and FrovedisBlockcyclicMatrix instance must be same during computation. Otherwise, it will raise an excpetion.

Return Value
1. **If 'a' and 'b' are python inputs such as numpy matrices and dtype = np.float32**:
- It returns a tuple **(lu, piv, x, rs_stat)** where,
- **lu**: It is a numpy matrix having float (float32) type values (by default) and containing the LU factor of input 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **piv**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **x**: It is also a numpy matrix having float (float32) type values (by default) and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **rs_stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' and 'b' are python inputs such as numpy matrices and dtype = np.float64**:
- It returns a tuple **(lu, piv, x, rs_stat)** where,
- **lu**: It is a numpy matrix having double (float64) type values (by default) and containing the LU factor of input 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **piv**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **x**: It is also a numpy matrix having double (float64) type values (by default) and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **rs_stat**: It returns an integer containing status (info) of native scalapack dgetrs.
3. **If 'a' and 'b' are instances of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lu, piv, x, rs_stat)** where,
- **lu**: It returns an instance of FrovedisBlockcyclicMatrix containing the LU factor of input 'a'.

- **piv**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **rs_stat**: It returns an integer containing status (info) of native scalapack dgetrs.

39.4.1.6 6. sgesv(a, b, overwrite_a = 0, overwrite_b = 0)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values.
b: It accepts a python array-like input or right hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values. It should have number of rows >= the number of rows in 'a' and at least 1 column in it.
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise, 'a' would be overwritten with LU factor. (Default: 0)
overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise, 'b' would be overwritten with the solution matrix. (Default: 0)

Purpose
It solves a system of linear equations, AX = B with a left hand side square matrix, 'a' by computing it's LU factors internally. It returns the LU factor, solution matrix for the system of linear equations with single (float32) precision.

This method internally uses Scalapack.gesv() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]],dtype = np.float32)
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]],dtype = np.float32)

# sgesv() demo
from frovedis.linalg.scalapack import sgesv
rf = sgesv(mat1,mat2)
print(rf)
```

Output

```
(array([[  1. ,   0. ,   2. ],
   [ -1. ,   5. ,   2. ],
   [  0. ,   0.6, -10.2]], dtype=float32),
   <frovedis.matrix.results.GetrfResult object at 0x7f49657f3cf8>,
   array([[-7.54902   ,  2.6862745 , -1.        ],
   [-1.509804  ,  1.137255  ,  0.        ],
   [-0.7254902 ,  0.15686277,  0.        ]], dtype=float32), 0)
```

This resultant tuple of sgesv() wrapper function contains GetrfResult instance in frovedis. Whereas in scipy, it contains an array.
Both here are responsible to hold pivot array information.

For example,

```
# sgesv() demo and overwriting of a with LU factor
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import sgesv
rf = sgesv(mat1,mat2,overwrite_a=1)
print('overwritten matrix with LU factor: ')
```

```
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix with LU factor:
[[  1.    0.    2. ]
 [ -1.    5.    2. ]
 [  0.    0.6 -10.2]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, then LU factor would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# sgesv() demo and overwriting of b with solution matrix
print('original matrix mat2: ')
print(mat2)
from frovedis.linalg.scalapack import sgesv
rf = sgesv(mat1,mat2,overwrite_b=1)
print('overwritten matrix with solution matrix: ')
print(mat2)
```

Output

```
original matrix mat2:
[[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]

overwritten matrix with solution matrix:
[[-7.54901981  2.68627453 -1.        ]
 [-1.50980401  1.13725495  0.        ]
 [-0.72549021  0.15686277  0.        ]]
```

Here, if the input 'mat2' is double (float64) type and overwite is enabled, then solution matrix would also be double (float64) type.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype=np.float32)
bcm2 = FrovedisBlockcyclicMatrix(mat2,dtype=np.float32)

# sgesv() demo and a and b as an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import sgesv
rf = sgesv(bcm1,bcm2)
```

```
# Unpacking the tuple
rf[0].debug_print()
print(rf[1])
rf[2].debug_print()
print(rf[3])
```

Output

```
<frovedis.matrix.results.GetrfResult object at 0x7f67ddeccd30>
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 1 -1 0 0 5 0.6 2 2 -10.2
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Return Value
1. **If 'a' and 'b' are python inputs such as numpy matrices**:
- It returns a tuple **(lu, piv, x, rs_stat)** where,
- **lu**: It is a numpy matrix having float (float32) type values (by default) and containing the LU factor of input 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **piv**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **x**: It is also a numpy matrix having float (float32) type values (by default) and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **rs_stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' and 'b' are instances of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lu, piv, x, rs_stat)** where,
- **lu**: It returns an instance of FrovedisBlockcyclicMatrix containing the LU factor of input 'a'.
- **piv**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **rs_stat**: It returns an integer containing status (info) of native scalapack dgetrs.

39.4.1.7 7. dgesvd(a, compute_uv = 1, full_matrices = 0, lwork = 0, overwrite_a = 0)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having double (float64) type values.
compute_uv: It accepts an integer parameter that specifies whether left singular matrix and right singular matrix will be computed. If it is set as '0', a numpy matrix of shape **(1,1)** and datatype same as 'a' will be assigned to the left and right singular matrix. (Default: 1)
full_matrices: It accepts an integer parameter. Currently, it can only be set as 0. Also, the left singular matrix and right singular matrix shapes are **(M, K)** and **(K, N)**, respectively, where **K = min(M, N)**, **M** is the number of rows and **N** is the number of columns of input matrix. (Default: 0)
lwork: This is an ununsed parameter. (Default: 0)
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise 'a' would be consumed internally and its contents will be destroyed. (Default: 0)

Purpose
It computes the singular value decomposition (SVD) of matrix 'a' with double (float64) precision.

The parameter: "lwork" is simply kept in to to make the interface uniform to the scipy.linalg.lapack.dgesvd() module. They are not used anywhere within the frovedis implementation.

This method internally uses Scalapack.gesvd() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# dgesvd() demo
from frovedis.linalg.scalapack import dgesvd
rf = dgesvd(mat1)
print(rf)
```

Output

```
(array([[ 0.19046167, -0.12914878, -0.97316234],
   [-0.20606538, -0.97448299,  0.08899413],
   [-0.95982364,  0.18358509, -0.21221475]]), array([9.83830146, 4.80016509, 1.0799257 ]),
   array([[ 0.04030442, -0.39740577,  0.91675744],
   [ 0.17610524, -0.9003148 , -0.39802035],
   [-0.98354588, -0.17748777, -0.03369857]]), 0)
```

For example,

```
# dgesvd() demo and compute_uv = 0
from frovedis.linalg.scalapack import dgesvd
rf = dgesvd(mat1,compute_uv=0)
print(rf)
```

Output

```
(array([[0.]]), array([9.83830146, 4.80016509, 1.0799257 ]), array([[0.]]), 0)
```

For example,

```
# dgesvd() demo and overwriting of a
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import dgesvd
rf = dgesvd(mat1,overwrite_a=1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix:
[[-1.41421356 -3.80788655 -0.1925824 ]
 [-0.41421356  6.71975985  5.52667534]
 [ 0.          0.64659915 -5.36662718]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwitten with double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)

# dgesvd() demo and a is an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import dgesvd
rf = dgesvd(bcm1)

# Unpacking the tuple
rf[0].debug_print()
rf[1].debug_print()
rf[2].debug_print()
```

Output

```
vector:
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.190462 -0.206065 -0.959824 -0.129149 -0.974483 0.183585 -0.973162 0.0889941 -0.212215
[9.83830146 4.80016509 1.0799257 ]
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.0403044 -0.397406 0.916757 0.176105 -0.900315 -0.39802 -0.983546 -0.177488 -0.0336986
```

Return Value
1. **If 'a' is a python input such as numpy matrix**:
- It returns a tuple **(u, s, vt, stat)** where,
- **u**: It is a numpy matrix having double (float64) type values and containing the left singular matrix.
- **s**: It is a numpy matrix having double (float64) type values and containing the singular matrix.
- **vt**: It is a numpy matrix having double (float64) type values and containing the right singular matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(u, s, vt, stat)** where,
- **u**: It returns an instance of FrovedisBlockcyclicMatrix containing the left singular matrix.
- **s**: It is a numpy matrix having double (float64) type values and containing the singular matrix.
- **vt**: It returns an instance of FrovedisBlockcyclicMatrix containing the right singular matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.

39.4.1.8 8. gesvd(a, compute_uv = 1, full_matrices = 0, lwork = 0, overwrite_a = 0, dtype = np.float64)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
compute_uv: It accepts an integer parameter that specifies whether left singular matrix and right singular matrix will be computed. If it is set as '0', a numpy matrix of shape **(1,1)** and datatype same as 'a' will be

assigned to the left and right singular matrix. (Default: 1)
full_matrices: It accepts an integer parameter. Currently, it can only be set as 0. Also, the left singular matrix and right singular matrix shapes are **(M, K)** and **(K, N)**, respectively, where **K = min(M, N)**, **M** is the number of rows and **N** is the number of columns of input matrix. (Default: 0)
lwork: This is an ununsed parameter. (Default: 0)
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise 'a' would be consumed internally and its contents will be destroyed. (Default: 0)
dtype: It specifies the datatype to be used for setting the precision level (single for float32 / double for float64) for the values returned by this method. (Default: np.float64)
Currently, it supports float (float32) or double (float64) datatypes.

Purpose
It computes the singular value decomposition (SVD) of matrix 'a' with single (float32) or double (float64) precision depending on the 'dtype' parameter provided by user.

The parameter: "lwork" is simply kept in to to make the interface uniform to the scipy.linalg.lapack.dgesvd() module. They are not used anywhere within the frovedis implementation.

This method is present only in frovedis.

This method internally uses Scalapack.gesvd() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# gesvd() demo
from frovedis.linalg.scalapack import gesvd
rf = gesvd(mat1)
print(rf)
```

Output

```
(array([[ 0.19046167, -0.12914878, -0.97316234],
   [-0.20606538, -0.97448299,  0.08899413],
   [-0.95982364,  0.18358509, -0.21221475]]),
   array([9.83830146, 4.80016509, 1.0799257 ]),
   array([[ 0.04030442, -0.39740577,  0.91675744],
   [ 0.17610524, -0.9003148 , -0.39802035],
   [-0.98354588, -0.17748777, -0.03369857]]), 0)
```

For example,

```
# gesvd() demo and compute_uv = 0
from frovedis.linalg.scalapack import gesvd
rf = gesvd(mat1,compute_uv=0)
print(rf)
```

Output

```
(array([[0.]]), array([9.83830146, 4.80016509, 1.0799257 ]), array([[0.]]), 0)
```

For example,

```
# gesvd() demo and overwriting of a
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import gesvd
rf = gesvd(mat1,overwrite_a=1)
print('overwritten matrix: ')
```

```
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix:
[[-1.41421356 -3.80788655 -0.1925824 ]
 [-0.41421356  6.71975985  5.52667534]
 [ 0.          0.64659915 -5.36662718]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwritten with double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# gesvd() demo and specifying the dtypes of output matrices
from frovedis.linalg.scalapack import gesvd
rf = gesvd(mat1,dtype=np.float32)
print(rf)
```

Output

```
(array([[ 0.19046175, -0.12914878, -0.9731621 ],
       [-0.20606534, -0.97448295,  0.08899409],
       [-0.9598237 ,  0.1835851 , -0.2122148 ]], dtype=float32),
       array([9.838303 , 4.800165 , 1.0799255], dtype=float32),
       array([[ 0.04030442, -0.3974058 ,  0.91675735],
       [ 0.17610519, -0.90031475, -0.39802024],
       [-0.98354584, -0.17748773, -0.03369856]], dtype=float32), 0)
```

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)

# gesvd() demo and a is an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import gesvd
rf = gesvd(bcm1)

# Unpacking the tuple
rf[0].debug_print()
rf[1].debug_print()
rf[2].debug_print()
```

Output

```
vector:
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.190462 -0.206065 -0.959824 -0.129149 -0.974483 0.183585 -0.973162 0.0889941 -0.212215
```

```
[9.83830146 4.80016509 1.0799257 ]
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.0403044 -0.397406 0.916757 0.176105 -0.900315 -0.39802 -0.983546 -0.177488 -0.0336986
```

Note:- 'dtype' for the wrapper function and FrovedisBlockcyclicMatrix instance must be same during computation. Otherwise, it will raise an excpetion.

Return Value
1. **If 'a' is a python input such as numpy matrix and dtype = np.float32**:
- It returns a tuple **(u, s, vt, stat)** where,
- **u**: It is a numpy matrix having float (float32) type values and containing the left singular matrix.
- **s**: It is a numpy matrix having float (float32) type values and containing the singular matrix.
- **vt**: It is a numpy matrix having float (float32) type values and containing the right singular matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' is a python input such as numpy matrix and dtype = np.float64**:
- It returns a tuple **(u, s, vt, stat)** where,
- **u**: It is a numpy matrix having double (float64) type values and containing the left singular matrix.
- **s**: It is a numpy matrix having double (float64) type values and containing the singular matrix.
- **vt**: It is a numpy matrix having double (float64) type values and containing the right singular matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
3. **If 'a' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(u, s, vt, stat)** where,
- **u**: It returns an instance of FrovedisBlockcyclicMatrix containing the left singular matrix.
- **s**: It is a numpy matrix having double (float64) type values and containing the singular matrix.
- **vt**: It returns an instance of FrovedisBlockcyclicMatrix containing the right singular matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.

39.4.1.9 9. sgesvd(a, compute_uv = 1, full_matrices = 0, lwork = 0, overwrite_a = 0)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values.
compute_uv: It accepts an integer parameter that specifies whether left singular matrix and right singular matrix will be computed. If it is set as '0', a numpy matrix of shape **(1,1)** and datatype same as 'a' will be assigned to the left and right singular matrix. (Default: 1)
full_matrices: It accepts an integer parameter. Currently, it can only be set as 0. Also, the left singular matrix and right singular matrix shapes are **(M, K)** and **(K, N)**, respectively, where **K = min(M, N)**, **M** is the number of rows and **N** is the number of columns of input matrix. (Default: 0)
lwork: This is an ununsed parameter. (Default: 0)
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise 'a' would be consumed internally and its contents will be destroyed. (Default: 0)

Purpose
It computes the singular value decomposition (SVD) of matrix 'a' with single (float32) precision.

The parameter: "lwork" is simply kept in to to make the interface uniform to the scipy.linalg.lapack.sgesvd() module. They are not used anywhere within the frovedis implementation.

This method internally uses Scalapack.gesvd() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
```

```
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# sgesvd() demo
from frovedis.linalg.scalapack import sgesvd
rf = sgesvd(mat1)
print(rf)
```

Output

```
(array([[ 0.19046175, -0.12914878, -0.9731621 ],
   [-0.20606534, -0.97448295,  0.08899409],
   [-0.9598237 ,  0.1835851 , -0.2122148 ]], dtype=float32),
   array([9.838303 , 4.800165 , 1.0799255], dtype=float32),
   array([[ 0.04030442, -0.3974058 ,  0.91675735],
   [ 0.17610519, -0.90031475, -0.39802024],
   [-0.98354584, -0.17748773, -0.03369856]], dtype=float32), 0)
```

For example,

```
# sgesvd() demo and compute_uv = 0
from frovedis.linalg.scalapack import sgesvd
rf = sgesvd(mat1,compute_uv=0)
print(rf)
```

Output

```
(array([[0.]], dtype=float32), array([9.838302 , 4.800165 , 1.0799258], dtype=float32),
array([[0.]], dtype=float32), 0)
```

For example,

```
# sgesvd() demo and overwriting of a
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import sgesvd
rf = sgesvd(mat1,overwrite_a=1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix:
[[-1.41421354 -3.80788684 -0.19258241]
 [-0.41421354  6.71975994  5.52667427]
 [ 0.          0.64659911 -5.36662769]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwitten with double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
```

```
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype = np.float32)

# sgesvd() demo and a is an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import sgesvd
rf = sgesvd(bcm1)

# Unpacking the tuple
rf[0].debug_print()
rf[1].debug_print()
rf[2].debug_print()
```

Output

```
vector:
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.190462 -0.206065 -0.959824 -0.129149 -0.974483 0.183585 -0.973162 0.0889941 -0.212215
[9.838303  4.800165  1.0799255]
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.0403044 -0.397406 0.916757 0.176105 -0.900315 -0.39802 -0.983546 -0.177488 -0.0336986
```

Return Value
1. **If 'a' is a python input such as numpy matrices**:
- It returns a tuple **(u, s, vt, stat)** where,
- **u**: It is a numpy matrix having float (float32) type values and containing the left singular matrix.
- **s**: It is a numpy matrix having float (float32) type values and containing the singular matrix.
- **vt**: It is a numpy matrix having float (float32) type values and containing the right singular matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'a' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(u, s, vt, stat)** where,
- **u**: It returns an instance of FrovedisBlockcyclicMatrix containing the left singular matrix.
- **s**: It is a numpy matrix having float (float32) type values and containing the singular matrix.
- **vt**: It returns an instance of FrovedisBlockcyclicMatrix containing the right singular matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.

39.4.1.10 10. dgetrf(a, overwrite_a = 0)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise 'a' will be overwritten with 'L' and 'U' factors. (Default: 0)

Purpose
It computes LU factorization of matrix 'a' with double (float64) precision.

It computes an LU factorization of matrix 'a', using partial pivoting with row interchanges.

This method internally uses Scalapack.getrf() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# dgetrf() demo
from frovedis.linalg.scalapack import dgetrf
rf = dgetrf(mat1)
print(rf)
```

Output

```
(array([[  1. ,   0. ,   2. ],
       [ -1. ,   5. ,   2. ],
       [  0. ,   0.6, -10.2]]),
       <frovedis.matrix.results.GetrfResult object at 0x7f80614aacf8>, 0)
```

This resultant tuple of dgetrf() wrapper function contains GetrfResult instance in frovedis. Whereas in scipy, it contains an array.
Both here are responsible to hold pivot array information.

For example,

```
# dgetrf() demo and overwriting of a
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import dgetrf
rf = dgetrf(mat1,overwrite_a=1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix:
[[  1.    0.    2. ]
 [ -1.    5.    2. ]
 [  0.    0.6 -10.2]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwritten with double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype = np.float64)

# dgetrf() demo and bcm1 is an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import dgetrf
rf = dgetrf(bcm1)

# Unpacking the tuple
rf[0].debug_print()
```

```
print(rf[1])
print(rf[2])
```

Output

```
<frovedis.matrix.results.GetrfResult object at 0x7f785d244978>
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 1 -1 0 0 5 0.6 2 2 -10.2
```

Return Value
1. **If 'a' is a python inputs such as numpy matrix**:
- It returns a tuple **(lu, res, stat)** where,
- **lu**: It is a numpy matrix having double (float64) type values (by default) and containing the LU factor of input matrix 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **res**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrf.
2. **If 'a' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lu, res, stat)** where,
- **lu**: It returns an instance of FrovedisBlockcyclicMatrix containing the LU factor of input matrix 'a'.
- **res**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrf.

39.4.1.11 11. getrf(a, overwrite_a = 0, dtype = np.float64)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise 'a' will be overwritten with 'LU'. (Default: 0)
dtype: It specifies the datatype to be used for setting the precision level (single for float32 / double for float64) for the values returned by this method. (Default: np.float64)
Currently, it supports float (float32) or double (float64) datatypes.

Purpose
It computes the LU factorization of matrix 'a' with single (float32) or double (float64) precision depending on 'dtype' parameter provided by user.

It computes an LU factorization of matrix 'a', using partial pivoting with row interchanges.

This method is present only in frovedis.

This method internally uses Scalapack.getrf() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# getrf() demo
from frovedis.linalg.scalapack import getrf
rf = getrf(mat1)
print(rf)
```

Output

```
(array([[  1. ,   0. ,   2. ],
       [ -1. ,   5. ,   2. ],
       [  0. ,   0.6, -10.2]]),
       <frovedis.matrix.results.GetrfResult object at 0x7f80614aacf8>, 0)
```

For example,

```
# getrf() demo and overwriting of a
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import getrf
rf = getrf(mat1,overwrite_a=1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix:
[[  1.    0.    2. ]
 [ -1.    5.    2. ]
 [  0.    0.6 -10.2]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwitten with double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
# getrf() demo and specifying the dtype of output matrix LU factor
from frovedis.linalg.scalapack import getrf
rf = getrf(mat1,dtype=np.float32)
print(rf)
```

Output

```
(array([[  1. ,   0. ,   2. ],
       [ -1. ,   5. ,   2. ],
       [  0. ,   0.6, -10.2]], dtype=float32),
       <frovedis.matrix.results.GetrfResult object at 0x7f7c0bdbbd30>, 0)
```

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype = np.float32)

# getrf() demo and bcm1 is an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import getrf
rf = getrf(bcm1,dtype = np.float32)

# Unpacking the tuple
```

```
rf[0].debug_print()
print(rf[1])
print(rf[2])
```

Output

```
<frovedis.matrix.results.GetrfResult object at 0x7f56091379b0>
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 1 -1 0 0 5 0.6 2 2 -10.2
```

Note:- 'dtype' for the wrapper function and FrovedisBlockcyclicMatrix instance must be same during computation. Otherwise, it will raise an excpetion.

Return Value
1. **If 'a' is a python input such as numpy matrix and dtype = np.float32**:
- It returns a tuple **(lu, res, stat)** where,
- **lu**: It is a numpy matrix having float (float32) type values (by default) and containing the LU factor of input matrix 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **res**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrf.
1. **If 'a' is a python input such as numpy matrix and dtype = np.float64**:
- It returns a tuple **(lu, res, stat)** where,
- **lu**: It is a numpy matrix having double (float64) type values (by default) and containing the LU factor of input matrix 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **res**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrf.
3. **If 'a' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lu, res, stat)** where,
- **lu**: It returns an instance of FrovedisBlockcyclicMatrix containing the LU factor of input matrix 'a'.
- **res**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrf.

39.4.1.12 12. sgetrf(a, overwrite_a = 0)

Parameters
a: It accepts a python array-like input or left hand side numpy matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values.
overwrite_a: It accepts an integer parameter, if set to 0, then 'a' will remain unchanged. Otherwise 'a' will be overwritten with 'LU'. (Default: 0)

Purpose
It computes the LU factorization of matrix 'a' with single (float32) precision.

This method internally uses Scalapack.getrf() **(present in frovedis.matrix module)**.

It computes an LU factorization of matrix 'a', using partial pivoting with row interchanges.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# sgetrf() demo
```

```
from frovedis.linalg.scalapack import sgetrf
rf = sgetrf(mat1)
print(rf)
```

Output

```
(array([[  1. ,   0. ,   2. ],
       [ -1. ,   5. ,   2. ],
       [  0. ,   0.6, -10.2]], dtype=float32),
       <frovedis.matrix.results.GetrfResult object at 0x7f1498750cf8>, 0)
```

This resultant tuple of sgetrf() wrapper function contains GetrfResult instance in frovedis. Whereas in scipy, it contains an array.
Both here are responsible to hold pivot array information.

For example,

```
# sgetrf() demo and overwriting of a
print('original matrix mat1: ')
print(mat1)
from frovedis.linalg.scalapack import sgetrf
rf = sgetrf(mat1,overwrite_a=1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]
overwritten matrix:
[[  1.          0.          2.        ]
 [ -1.          5.          2.        ]
 [  0.          0.60000002 -10.19999981]]
```

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwitten with double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype = np.float32)

# sgetrf() demo and bcm1 is an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import getrf
rf = getrf(bcm1,dtype = np.float32)

# Unpacking the tuple
rf[0].debug_print()
print(rf[1])
print(rf[2])
```

Output

```
<frovedis.matrix.results.GetrfResult object at 0x7f6199f28908>
```

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 1 -1 0 0 5 0.6 2 2 -10.2
```

Return Value
1. **If 'a' is a python input such as numpy matrix**:
- It returns a tuple **(lu, res, stat)** where,
- **lu**: It is a numpy matrix having float (float32) type values (by default) and containing the LU factor of input matrix 'a'. In case 'overwrite_a' is enabled, then dtype for the matrix will depend on input 'a' dtype.
- **res**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrf.
2. **If 'a' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(lu, res, stat)** where,
- **lu**: It returns an instance of FrovedisBlockcyclicMatrix containing the LU factor of input matrix 'a'.
- **res**: It returns an instance of GetrfResult containing server side pointer of pivot array.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrf.

39.4.1.13 13. dgetri(lu, piv, lwork = 0, overwrite_lu = 0)

Parameters
lu: It accepts a python array-like input or numpy matrix having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having double (float64) type values computed using dgetrf().
piv: It accepts an instance of GetrfResult containing server side pointer of pivot array computed using dgetrf().
lwork: This is an unused parameter. (Default: 0)
overwrite_lu: It accepts an integer parameter, if set to 0, then 'lu' will remain unchanged. Otherwise 'lu' will be overwritten with inverse matrix. (Default: 0)

Purpose
It computes the inverse matrix with double (float64) precision.

The parameter: "lwork" is simply kept in to to make the interface uniform to the scipy.linalg.lapack.dgetri() module. They are not used anywhere within the frovedis implementation.

This method internally uses Scalapack.getri() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# dgetrf() demo
from frovedis.linalg.scalapack import dgetrf,dgetri

# compute lu and piv using dgetrf()
rf = dgetrf(mat1)

# dgetri() demo
ri = dgetri(rf[0],rf[1])
print(ri)
```

Output

```
(array([[ 0.88235294, -0.11764706,  0.19607843],
   [ 0.17647059,  0.17647059,  0.03921569],
   [ 0.05882353,  0.05882353, -0.09803922]]), 0)
```

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# dgetrf() demo
from frovedis.linalg.scalapack import dgetrf,dgetri

# compute lu and piv using dgetrf()
rf = dgetrf(mat1,overwrite_a = 1)

# dgetri() demo and overwriting of lu
print('original matrix mat1: ')
print(mat1)
ri = dgetri(rf[0],rf[1], overwrite_lu = 1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]
overwritten matrix:
[[ 0.88235294 -0.11764706  0.19607843]
 [ 0.17647059  0.17647059  0.03921569]
 [ 0.05882353  0.05882353 -0.09803922]]
```

The input matrix is only processed by dgetrf() first to generate LU factor. The LU factor and ipiv information is then used with dgetri().

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwitten with inverse matrix having double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)
bcm2 = FrovedisBlockcyclicMatrix(mat2)

# dgetri() demo and bcm1 and bcm2 are an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import dgetrf,dgetri
rf = dgetrf(bcm1)
ri = dgetri(rf[0],rf[1],bcm2)

# Unpacking the tuple
ri[0].debug_print()
print(ri[1])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.882353 0.176471 0.0588235 -0.117647 0.176471 0.0588235 0.196078 0.0392157 -0.0980392
```

Return Value
1. **If 'lu' is python input such as numpy matrix**:
- It returns a tuple **(inv_a, stat)** where,
- **inv_a**: It is a numpy matrix having double (float64) type values (by default) and containing the inverse matrix. In case 'overwrite_lu' is enabled, then dtype for the matrix will depend on intermediate input 'LU' factor dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.
2. **If 'lu' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(inv_a, stat)** where,
- **inv_a**: It returns an instance of FrovedisBlockcyclicMatrix containing the inverse matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.

39.4.1.14 14. getri(lu, piv, lwork = 0, overwrite_lu = 0, dtype = np.float64)

Parameters
lu: It accepts a python array-like input or numpy matrix having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values.
piv: It accepts an instance of GetrfResult containing server side pointer of pivot array.
lwork: This is an unused parameter. (Default: 0)
overwrite_lu: It accepts an integer parameter, if set to 0, then 'lu' will remain unchanged. Otherwise 'lu' will be overwritten with inverse matrix. (Default: 0)
dtype: It specifies the datatype to be used for setting the precision level (single for float32 / double for float64) for the values returned by this method. (Default: np.float64)
Currently, it supports float (float32) or double (float64) datatypes.

Purpose
It computes the inverse matrix with single (float32) or double (float64) precision depending on 'dtype' parameter provided by user.

The parameter: "lwork" is simply kept in to to make the interface uniform to the scipy.linalg.lapack.dgetri() module. They are not used anywhere within the frovedis implementation.

This method is present only in frovedis.

This method internally uses Scalapack.getri() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# getrf() demo
from frovedis.linalg.scalapack import getrf,getri

# compute lu and piv using getrf()
rf = getrf(mat1)
```

```
# getri() demo
ri = getri(rf[0],rf[1])
print(ri)
```

Output

```
(array([[ 0.88235294, -0.11764706,  0.19607843],
       [ 0.17647059,  0.17647059,  0.03921569],
       [ 0.05882353,  0.05882353, -0.09803922]]), 0)
```

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# getrf() demo
from frovedis.linalg.scalapack import getrf,getri

# compute lu and piv using getrf()
rf = getrf(mat1, overwrite_a = 1)

# getri() demo and overwriting of lu
print('original matrix mat1: ')
print(mat1)
ri = getri(rf[0],rf[1], overwrite_lu = 1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]
overwritten matrix:
[[ 0.88235294 -0.11764706  0.19607843]
 [ 0.17647059  0.17647059  0.03921569]
 [ 0.05882353  0.05882353 -0.09803922]]
```

The input matrix is only processed by getrf() first to generate LU factor. The LU factor and ipiv information is then used with getri().

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwitten with inverse matrix having double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# getrf() demo
from frovedis.linalg.scalapack import getrf,getri

# compute lu and piv using getrf()
rf = getrf(mat1)
```

```
# getri() demo and specifying the dtype of output LU matrix
from frovedis.linalg.scalapack import getri
rf = getri(rf[0],rf[1],dtype=np.float32)
print(rf)
```

Output

```
(array([[ 0.88235295, -0.11764707,  0.19607843],
   [ 0.1764706 ,  0.1764706 ,  0.03921569],
   [ 0.05882353,  0.05882353, -0.09803922]], dtype=float32), 0)
```

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype = np.float32)
bcm2 = FrovedisBlockcyclicMatrix(mat2,dtype = np.float32)

# sgetri() demo and bcm1 and bcm2 are an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import getrf,getri
rf = getrf(bcm1,dtype = np.float32)
ri = getri(rf[0],rf[1],bcm2, dtype = np.float32)

# Unpacking the tuple
ri[0].debug_print()
print(ri[1])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.882353 0.176471 0.0588235 -0.117647 0.176471 0.0588235 0.196078 0.0392157 -0.0980392
```

Note:- 'dtype' for the wrapper function and FrovedisBlockcyclicMatrix instance must be same during computation. Otherwise, it will raise an excpetion.

Return Value
1. **If 'lu' is python input such as numpy matrix and dtype = np.float32**:
- It returns a tuple **(inv_a, stat)** where,
- **inv_a**: It is a numpy matrix having float (float32) type values (by default) and containing the inverse matrix. In case 'overwrite_lu' is enabled, then dtype for the matrix will depend on intermediate input 'LU' factor dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.
2. **If 'lu' is python input such as numpy matrix and dtype = np.float64**:
- It returns a tuple **(inv_a, stat)** where,
- **inv_a**: It is a numpy matrix having double (float64) type values (by default) and containing the inverse matrix. In case 'overwrite_lu' is enabled, then dtype for the matrix will depend on intermediate input 'LU' factor dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.
3. **If 'lu' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(inv_a, stat)** where,
- **inv_a**: It returns an instance of FrovedisBlockcyclicMatrix containing the inverse matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.

39.4.1.15 15. sgetri(lu, piv, lwork = 0, overwrite_lu = 0)

Parameters
lu: It accepts a python array-like input or numpy matrix having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values.
piv: It accepts an instance of GetrfResult containing server side pointer of pivot array.
lwork: This is an unused parameter. (Default: 0)
overwrite_lu: It accepts an integer parameter, if set to 0, then 'lu' will remain unchanged. Otherwise 'lu' will be overwritten with inverse matrix. (Default: 0)

Purpose
It computes the inverse matrix with single (float32) precision.

The parameter: "lwork" is simply kept in to to make the interface uniform to the scipy.linalg.lapack.dgetri() module. They are not used anywhere within the frovedis implementation.

This method internally uses Scalapack.getri() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# sgetrf() demo
from frovedis.linalg.scalapack import sgetrf,sgetri

# compute lu and piv using sgetrf()
rf = sgetrf(mat1)

# sgetri() demo
ri = sgetri(rf[0],rf[1])
print(ri)
```

Output

```
(array([[ 0.88235295, -0.11764707,  0.19607843],
   [ 0.1764706 ,  0.1764706 ,  0.03921569],
   [ 0.05882353,  0.05882353, -0.09803922]], dtype=float32), 0)
```

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# sgetrf() demo
from frovedis.linalg.scalapack import sgetrf,sgetri

# compute lu and piv using sgetrf()
rf = sgetrf(mat1, overwrite_a = 1)

# sgetri() demo and overwriting of lu
print('original matrix mat1: ')
print(mat1)
ri = sgetri(rf[0],rf[1], overwrite_lu = 1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat1:
[[ 1.  0.  2.]
 [-1.  5.  0.]
 [ 0.  3. -9.]]

overwritten matrix:
[[ 0.88235295 -0.11764707  0.19607843]
 [ 0.17647059  0.17647059  0.03921569]
 [ 0.05882353  0.05882353 -0.09803922]]
```

The input matrix is only processed by sgetrf() first to generate LU factor. The LU factor and ipiv information is then used with sgetri().

Here, if the input 'mat1' is double (float64) type and overwite is enabled, it's overwitten with inverse matrix having double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype = np.float32)
bcm2 = FrovedisBlockcyclicMatrix(mat2,dtype = np.float32)

# sgetri() demo and bcm1 and bcm2 are an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import sgetrf,sgetri
rf = sgetrf(bcm1)
ri = sgetri(rf[0],rf[1],bcm2)

# Unpacking the tuple
ri[0].debug_print()
print(ri[1])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = 0.882353 0.176471 0.0588235 -0.117647 0.176471 0.0588235 0.196078 0.0392157 -0.0980392
```

Return Value
1. **If 'lu' is python input such as numpy matrix**:
- It returns a tuple **(inv_a, stat)** where,
- **inv_a**: It is a numpy matrix having float (float32) type values (by default) and containing the inverse matrix. In case 'overwrite_lu' is enabled, then dtype for the matrix will depend on intermediate input 'LU' factor dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.
2. **If 'lu' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(inv_a, stat)** where,
- **inv_a**: It returns an instance of FrovedisBlockcyclicMatrix containing the inverse matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.

39.4.1.16 16. dgetrs(lu, piv, b, trans = 0, overwrite_b = 0)

Parameters
lu: It accepts a python array-like input or numpy matrix having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having double (float64) type values computed using dgetrf.
piv: It accepts an instance of GetrfResult containing server side pointer of pivot array computed using dgetrf().
b: It accepts a python array-like input or right hand side matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having double (float64) type values. It should have number of rows >= the number of rows in 'a' and at least 1 column in it.
trans: It accepts an integer parameter indicating if transpose of 'lu' needs to be computed before solving linear equation. If it is not 0, then the transpose is computed. (Default: 0)
overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise 'b' will be overwritten with the solution matrix. (Default: 0)

Purpose
It solves a system of linear equations, AX = B with matrix 'a' using the LU factorization computed by getrf(). Thus before calling this function, it is required to obtain the factored matrix 'lu' (along with piv information) by calling getrf().

It computes the solution matrix for the system of linear equations with double (float64) precision.

This method internally uses Scalapack.getrs() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# dgetrf() demo
from frovedis.linalg.scalapack import dgetrf,dgetrs

# compute lu and piv using dgetrf()
rf = dgetrf(mat1)

# dgetrs() demo
ri = dgetrs(rf[0],rf[1],mat2)
print(ri)
```

Output

```
(array([[-7.54901961,  2.68627451, -1.        ],
   [-1.50980392,  1.1372549 ,  0.        ],
   [-0.7254902 ,  0.15686275,  0.        ]]), 0)
```

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# dgetrf() demo
from frovedis.linalg.scalapack import dgetrf,dgetrs

# compute lu and piv using dgetrf()
rf = dgetrf(mat1, overwrite_a = 1)
```

```
# dgetrs() demo and overwriting of b
print('original matrix mat2: ')
print(mat2)
ri = dgetrs(rf[0],rf[1],mat2,overwrite_b = 1)
print('overwritten matrix: ')
print(mat1)
```

Output

```
original matrix mat2:
[[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]
overwritten matrix:
[[-7.54901961  2.68627451 -1.        ]
 [-1.50980392  1.1372549   0.        ]
 [-0.7254902   0.15686275  0.        ]]
```

Here, if the inputs 'mat2' is double (float64) type and overwite is enabled in dgetrs(), it's overwritten with solution having double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1)
bcm2 = FrovedisBlockcyclicMatrix(mat2)

# dgetrs() demo and bcm1 and bcm2 are an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import dgetrf,dgetrs
rf = dgetrf(bcm1)
ri = dgetrs(rf[0],rf[1],bcm2)

# Unpacking the tuple
ri[0].debug_print()
print(ri[1])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Return Value

1. **If 'b' is python input such as numpy matrix**:

- It returns a tuple **(x, stat)** where,
- **x**: It is a numpy matrix having double (float64) type values (by default) and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.

2. **If 'b' is an instance of FrovedisBlockcyclicMatrix:**

- It returns a tuple **(x, stat)** where,
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.

39.4.1.17 17. getrs(lu, piv, b, trans = 0, overwrite_b = 0, dtype = np.float64)

Parameters
lu: It accepts a python array-like input or numpy matrix having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values computed using dgetrf.
piv: It accepts an instance of GetrfResult containing server side pointer of pivot array computed using dgetrf().
b: It accepts a python array-like input or right hand side matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) or double (float64) type values. It should have number of rows >= the number of rows in 'a' and at least 1 column in it.
trans: It accepts an integer parameter indicating if transpose of 'lu' needs to be computed before solving linear equation. If it is not 0, then the transpose is computed. (Default: 0)
overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise 'b' will be overwritten with the solution matrix. (Default: 0)
dtype: It specifies the datatype to be used for setting the precision level (single for float32 / double for float64) for the values returned by this method. (Default: np.float64)
Currently, it supports float (float32) or double (float64) datatypes.

Purpose
It solves a system of linear equations, AX = B with matrix 'a' using the LU factorization computed by getrf(). Thus before calling this function, it is required to obtain the factored matrix 'lu' by calling getrf().

It computes the solution matrix for the system of linear equations with float or double (float64) precision depending on 'dtype' parameter provided by user.

This method is present only in frovedis.

This method internally uses Scalapack.getrs() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# getrf() demo
from frovedis.linalg.scalapack import getrf,getrs

# compute lu and piv using getrf()
rf = getrf(mat1)

# getrs() demo
ri = getrs(rf[0],rf[1],mat2)
print(ri)
```

Output

```
(array([[-7.54901961,  2.68627451, -1.        ],
   [-1.50980392,  1.1372549 ,  0.        ],
   [-0.7254902 ,  0.15686275,  0.        ]]), 0)
```

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# getrf() demo
from frovedis.linalg.scalapack import getrf,getrs

# compute lu and piv using getrf()
rf = getrf(mat1, overwrite_a = 1)

# getrs() demo and overwriting of b
print('original matrix mat2: ')
print(mat2)
ri = getrs(rf[0],rf[1], mat2, overwrite_b = 1)
print('overwritten matrix: ')
print(mat2)
```

Output

```
original matrix mat2:
[[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]
overwritten matrix:
[[-7.54901961  2.68627451 -1.        ]
 [-1.50980392  1.1372549   0.        ]
 [-0.7254902   0.15686275  0.        ]]
```

Here, if the inputs 'mat2' is double (float64) type and overwite is enabled in dgetrs(), it's overwritten with solution having double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# getrf() demo
from frovedis.linalg.scalapack import getrf,getrs

# compute lu and piv using getrf()
rf = getrf(mat1,dtype=np.float32)

# getrs() demo and specifying the dtype of output LU matrix
from frovedis.linalg.scalapack import getrs
rf = getrs(rf[0],rf[1],mat2,dtype=np.float32)
print(rf)
```

Output

```
(array([[-7.54902   ,  2.6862745 , -1.        ],
   [-1.509804  ,  1.137255  ,  0.        ],
   [-0.7254902 ,  0.15686277,  0.        ]], dtype=float32), 0)
```

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
```

```
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype = np.float32)
bcm2 = FrovedisBlockcyclicMatrix(mat2, dtype = np.float32)

# getrs() demo and bcm1 and bcm2 are an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import getrf,getrs
rf = getrf(bcm1,dtype = np.float32)
ri = getrs(rf[0],rf[1],bcm2,dtype = np.float32)

# Unpacking the tuple
ri[0].debug_print()
print(ri[1])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Note:- 'dtype' for the wrapper function and FrovedisBlockcyclicMatrix instance must be same during computation. Otherwise, it will raise an excpetion.

Return Value
1. **If 'b' is python input such as numpy matrix and dtype = np.float32**:
- It returns a tuple **(x, stat)** where,
- **x**: It is a numpy matrix having float (float32) type values (by default) containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'b' is python input such as numpy matrix and dtype = np.float64**:
- It returns a tuple **(x, stat)** where,
- **x**: It is a numpy matrix having double (float64) type values (by default) containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
3. **If 'b' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(x, stat)** where,
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.

39.4.1.18 18. sgetrs(lu, piv, b, trans = 0, overwrite_b = 0)

Parameters
lu: It accepts a python array-like input or numpy matrix having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values computed using dgetrf.
piv: It accepts an instance of GetrfResult containing server side pointer of pivot array computed using dgetrf().
b: It accepts a python array-like input or right hand side matrix of the linear equation having int, float (float32) or double (float64) type values. It also accepts FrovedisBlockcyclicMatrix instance having float (float32) type values. It should have number of rows >= the number of rows in 'a' and at least 1 column in it.
trans: It accepts an integer parameter indicating if transpose of 'lu' needs to be computed before solving linear equation. If it is not 0, then the transpose is computed. (Default: 0)

overwrite_b: It accepts an integer parameter, if set to 0, then 'b' will remain unchanged. Otherwise 'b' will be overwritten with the solution matrix. (Default: 0)

Purpose
It solves a system of linear equations, AX = B with the matrix 'a' using the LU factorization computed by sgetrf().
Thus before calling this function, it is required to obtain the factored matrix 'lu' by calling sgetrf().

It computes the solution matrix for the system of linear equations with float precision.

This method internally uses Scalapack.getrs() **(present in frovedis.matrix module)**.

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])

# sgetrf() demo
from frovedis.linalg.scalapack import sgetrf,sgetrs

# compute lu and piv using sgetrf()
rf = sgetrf(mat1)

# sgetrs() demo
ri = sgetrs(rf[0]rf[1],mat2)
print(ri)
```

Output

```
(array([[-7.54902   ,  2.6862745 , -1.        ],
   [-1.509804  ,  1.137255  ,  0.        ],
   [-0.7254902 ,  0.15686277,  0.        ]], dtype=float32), 0)
```

For example,

```
import numpy as np
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])

# sgetrf() demo
from frovedis.linalg.scalapack import sgetrf,sgetrs

# compute lu and piv using sgetrf()
rf = sgetrf(mat1)

# sgetrs() demo and overwriting of b
print('original matrix mat2: ')
print(mat2)
ri = sgetrs(rf[0],rf[1], mat2, overwrite_b = 1)
print('overwritten matrix: ')
print(mat2)
```

Output

```
original matrix mat2:
[[-9.  3. -1.]
 [ 0.  3.  1.]
 [ 2.  2.  0.]]
overwritten matrix:
```

```
[[-7.54901981  2.68627453 -1.        ]
 [-1.50980401  1.13725495  0.        ]
 [-0.72549021  0.15686277  0.        ]]
```

Here, if the inputs 'mat2' is double (float64) type and overwite is enabled in dgetrs(), it's overwritten with solution having double (float64) type values.

Same applies for other types (int, float (float32)) when input is a numpy matrix/array and overwrite is enabled.

For example,

```
from frovedis.matrix.dense import FrovedisBlockcyclicMatrix
mat1 = np.array([[1.,0.,2.],[-1.,5.,0.],[0.,3.,-9.]])
mat2 = np.array([[-9.,3.,-1.],[0.,3.,1.],[2.,2.,0.]])
bcm1 = FrovedisBlockcyclicMatrix(mat1,dtype = np.float32)
bcm2 = FrovedisBlockcyclicMatrix(mat2,dtype = np.float32)

# sgetrs() demo and bcm1 and bcm2 are an instance of FrovedisBlockcyclicMatrix
from frovedis.linalg.scalapack import sgetrf,sgetrs
rf = sgetrf(bcm1)
ri = sgetrs(rf[0],rf[1],bcm2)

# Unpacking the tuple
ri[0].debug_print()
print(ri[1])
```

Output

```
0
matrix:
num_row = 3, num_col = 3
node 0
node = 0, local_num_row = 3, local_num_col = 3, type = 2, descriptor = 1 1 3 3 3 3 0 0 3 3 3
val = -7.54902 -1.5098 -0.72549 2.68627 1.13725 0.156863 -1 0 0
```

Return Value
1. **If 'b' is python input such as numpy matrix**:
- It returns a tuple **(x, stat)** where,
- **x**: It is a numpy matrix having float (float32) type value and containing the solution matrix. In case 'overwrite_b' is enabled, then dtype for the matrix will depend on input 'b' dtype.
- **stat**: It returns an integer containing status (info) of native scalapack dgetrs.
2. **If 'b' is an instance of FrovedisBlockcyclicMatrix:**
- It returns a tuple **(x, stat)** where,
- **x**: It returns an instance of FrovedisBlockcyclicMatrix containing the solution matrix.
- **stat**: It returns an integer containing status (info) of native scalapack dgetri.

39.5 SEE ALSO

- **Linalg Functions**

Chapter 40

Graph

40.1 NAME

Graph - This class implements a directed or undirected frovedis graph.

40.2 SYNOPSIS

```
frovedis.graph.graph.Graph(nx_graph=None)
```

40.2.1 Public Member Functions

load(nx_graph)
load_csr(smat)
debug_print()
release()
clear()
number_of_edges()
number_of_nodes()
save(fname)
load_text(fname)
to_networkx_graph()

40.3 DESCRIPTION

Base class for undirected and directed frovedis graphs. A Frovedis Graph stores nodes and edges with optional data. Here, the graph data, edge to edge information is internally stored as and adjacency matrix data.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis public method interface is almost same as NetworkX Graph interface, but it doesn't have any dependency with NetworkX. It can be used simply even if the system doesn't have NetworkX 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 Graph() on the frovedis server. Once the graph is loaded for the given input networkx or scipy csr_matrix graph data at the frovedis server, it returns a frovedis graph instance to the client python program.

40.3.1 Detailed Description

40.3.1.1 1. Graph()

Parameters
nx_graph: Data to initialize graph. Here data provided is a scipy sparse csr_matrix or a networkx.Graph instance. It loads such data to create a frovedis graph. (Default: None)
When it is None, an empty frovedis graph is created.

Purpose
It initializes a Frovedis Graph object having nodes and edges. It can create both directed or undirected graph.

For example,

a) When loading an undirected networkx.Graph as input,

```
import frovedis.graph as fnx
import networkx as gnx
net_graph = gnx.read_edgelist('input/cit-Patents_10.txt')
frov_graph = fnx.Graph(net_graph)
```

b) When loading a directed networkx.Graph as input,

```
import frovedis.graph as fnx
import networkx as gnx
net_graph = gnx.read_edgelist('input/cit-Patents_10.txt', create_using = gnx.DiGraph())
frov_graph = fnx.Graph(net_graph)
```

c) When loading a scipy sparse csr_matrix as input to create undirected frovedis graph,

```
#Here it creates an undirected graph
import frovedis.graph as fnx
from scipy.sparse import csr_matrix
data = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
row = [0,0,0,0,0,1,2,3,4,5,6,6,6,6,7,8,9,10,11,12]
col = [1,2,3,4,5,0,0,0,0,0,7,8,9,10,6,6,6,6,12,11]
csr_mat = csr_matrix((data, (row, col)), shape = (13, 13))
frov_graph = fnx.Graph(csr_mat)
```

d) Again, when loading a scipy sparse csr_matrix as input to create directed frovedis graph,

```
#Here it creates a directed graph
import frovedis.graph as fnx
from scipy.sparse import csr_matrix
data = [1,1,1,1,1,1,1,1,1,1]
row = [0,0,0,0,0,6,6,6,6,11]
col = [1,2,3,4,5,7,8,9,10,12]
csr_mat = csr_matrix((data, (row, col)), shape = (13, 13))
frov_graph = fnx.Graph(csr_mat)
```

Return Value
It simply returns “self” reference.

40.3.1.2 2. load(nx_graph)

Parameters
nx_graph: Data to initialize graph. Here graph data provided is a networkx.Graph instance.

Purpose
It loads a networkx graph to create a frovedis graph. Here, the loaded graph can be directed or undirected.

For example,

```
import frovedis.graph as fnx
import networkx as gnx
net_graph = gnx.read_edgelist('input/cit-Patents_10.txt')
frov_graph = fnx.Graph().load(nx_graph = net_graph)
```

Return Value
It simply returns "self" reference.

40.3.1.3 3. load_csr(smat)

Parameters
smat: Data to initialize graph. Here graph data provided is a scipy sparse csr_matrix.

Purpose
It loads frovedis graph from a scipy csr_matrix. Here, depending on the scipy csr_matrix data, the final graph maybe directed or undirected.

For example,

```
#Loading a scipy csr_matrix to create an undirected frovedis graph
import frovedis.graph as fnx
from scipy.sparse import csr_matrix
data = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
row = [0,0,0,0,0,1,2,3,4,5,6,6,6,6,7,8,9,10,11,12]
col = [1,2,3,4,5,0,0,0,0,0,7,8,9,10,6,6,6,6,12,11]
csr_mat = csr_matrix((data, (row, col)), shape = (13, 13))
frov_graph = fnx.Graph().load_csr(smat = csr_mat)

#Loading a scipy csr_matrix to create a directed frovedis graph
import frovedis.graph as fnx
from scipy.sparse import csr_matrix
data = [1,1,1,1,1,1,1,1,1,1]
row = [0,0,0,0,0,6,6,6,6,11]
col = [1,2,3,4,5,7,8,9,10,12]
csr_mat = csr_matrix((data, (row, col)), shape = (13, 13))
frov_graph = fnx.Graph().load_csr(smat = csr_mat)
```

Return Value
It simply returns "self" reference.

40.3.1.4 4. debug_print()

Purpose
It shows graph information on the client side and server side user terminal. It is mainly used for debugging purpose.

For example,

```
frov_graph.debug_print()
```

Output on client side

```
Num of edges:  20
Num of vertices:  13
```

It displays information such as number of edges and vertices.

Output on server side

```
Num of edges:  20
Num of vertices:  13
is directed: 0
is weighted: 0
in-degree:
5 1 1 1 1 1 4 1 1 1 1 1 1
out-degree:
5 1 1 1 1 1 4 1 1 1 1 1 1
adjacency matrix:
0 1 1 1 1 1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0
```

It displays information such as number of edges and vertices, directed ('0' for No, '1' for Yes) , incoming links, outgoing links and an adjacency matrix. Currently, it shows information about an undirected frovedis graph.

Return Value
It returns nothing.

40.3.1.5 5. release()

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

For example,

```
frov_graph.release()
```

This will remove the graph model, model-id present on server, along with releasing server side memory.

Return Value
It returns nothing.

40.3.1.6 6. clear()

Purpose
It can be used to release the in-memory model at frovedis server. This method is an alias to release().

For example,

```
frov_graph.clear()
```

This will remove the graph model, model-id present on server, along with releasing server side memory.

Return Value
It returns nothing.

40.3.1.7 7. number_of_edges()

Purpose
It is used to fetch the number of edges present in the directed or undirected frovedis graph.

For example,

```
edge_count = frov_graph.number_of_edges()
print('Number of edges: ', edge_count)
```

Output

```
Number of edges: 20
```

Return Value
It returns a long (int64) type value specifying the number of edges.

40.3.1.8 8. number_of_nodes()

Purpose
It is used to fetch the number of nodes/vertices present in the directed or undirected frovedis graph.

For example,

```
nodes_count = frov_graph.number_of_nodes()
print('Number of nodes: ', nodes_count)
```

Output

```
Number of nodes: 13
```

Return Value
It returns a long (int64) type value specifying the number of node/vertices.

40.3.1.9 9. save(fname)

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

Purpose
On success, it writes the adjacency matrix information in the specified file. Otherwise, it throws an exception.

For example,

```
# To save the frovedis graph
frov_graph.save("./out/FrovGraph")
```

This will save the frovedis grpah on the path '/out/FrovGraph'. It would raise exception if the directory already exists with same name.

Return Value
It returns nothing.

40.3.1.10 10. load_text(fname)

Parameters
fname: A string object containing the name of the file having adjacency matrix information to be loaded.

Purpose
It loads a frovedis graph from the specified file path (having adjacency matrix data).

For example,

```
# To load the frovedis graph
frov_graph.load_text("./out/FrovGraph")
```

Return Value
It simply returns "self" reference.

40.3.1.11 11. to_networkx_graph()

Purpose
It is used to convert a frovedis graph into a networkx graph.

For example,

```
nx_graph = frov_graph.to_networkx_graph()
print(nx_graph.adj)
```

Output

```
{0: {1: {'weight': 1.0}, 2: {'weight': 1.0}, 3: {'weight': 1.0}, 4: {'weight': 1.0},
5: {'weight': 1.0}}, 1: {0: {'weight': 1.0}}, 2: {0: {'weight': 1.0}},
3: {0: {'weight': 1.0}}, 4: {0: {'weight': 1.0}}, 5: {0: {'weight': 1.0}},
6: {7: {'weight': 1.0}, 8: {'weight': 1.0}, 9: {'weight': 1.0}, 10: {'weight': 1.0}},
7: {6: {'weight': 1.0}}, 8: {6: {'weight': 1.0}}, 9: {6: {'weight': 1.0}},
10: {6: {'weight': 1.0}}, 11: {12: {'weight': 1.0}}, 12: {11: {'weight': 1.0}}}
```

Return Value
It returns a networkx.Graph instance.

40.4 SEE ALSO

- **Graph in Frovedis**
- **Breadth First Search in Frovedis**
- **Connected Components in Frovedis**
- **PageRank in Frovedis**
- **Single Source Shortest Path in Frovedis**

Chapter 41

pagerank()

41.1 NAME

pagerank() - It computes a ranking of the nodes in the graph 'G' based on the structure of the incoming links. It was originally designed as an algorithm to rank web pages.

41.2 SYNOPSIS

```
frovedis.graph.Pagerank.pagerank(G, alpha=0.85, personalization=None, max_iter=100, tol=1.0e-6,
                                 nstart=None, weight='weight', dangling=None, verbose=0)
```

41.3 DESCRIPTION

The PageRank algorithm ranks the nodes in a graph by their relative importance or influence. PageRank determines each node's ranking by identifying the number of links to the node and the quality of the links. The quality of a link is determined by the importance (PageRank) of the node that presents the outbound link.

The PageRank algorithm was designed at first to measure the importance of a webpage by analyzing the quantity and quality of the links that point to it.

Mathematically, PageRank (PR) is defined as:

```
PR(A) = (1 - d) + d(PR(Ti)/C(Ti)+ ... + PR(Tn)/C(Tn))
```

where **Page A** has pages **T1 to Tn** which point to it.
d is a damping factor. It corresponds to the probability that an imaginary web surfer will suddenly visit a random page on the web instead of following one of the outbound links prescribed by the web page that the surfer is currently visiting. This is useful for accounting for sinks (web pages that have inbound links but no outbound links).
Also, the **(1 - d)** in the preceding formula dampens the contribution of the incoming PageRanks from the adjacent vertices. It is as if imaginary outbound edges are added from all sink vertices to every other vertex in the graph, and to keep things fair, this same thing is done to the non-sink vertices as well.
C(A) is defined as the number of links going out of page A.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis public method interface is almost same as NetworkX pagerank public method interface, but

it doesn't have any dependency with NetworkX. It can be used simply even if the system doesn't have NetworkX 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 pagerank() on the frovedis server. Once the PageRanks are computed for the given input graph data at the frovedis server, it returns a dictionary of nodes with PageRank as value to the client python program.

41.3.1 Detailed Description

41.3.1.1 1. pagerank()

Parameters
G: An instance of networkx graph or frovedis graph on which pagerank is to be computed. The graph can be directed or undirected.
alpha: A positive integer parameter that is referred as **damping factor**. It must be in range 0 and 1. It represents a sort of minimum PageRank value. (Default: 0.85)
personalization: An unused parameter. (Default: None)
max_iter: A positive integer parameter that specifies the maximum number of iterations to run this method. Convergence may occur before max_iter value, in that case the iterations will stop. (Default: 100)
tol: A positive double (float64) parameter specifying the convergence tolerance. The PageRank iterations stop if the sum of the error values for all nodes is below this value. (Default: 1.0e-6)
nstart: An unused parameter. (Default: None)
weight: An unused parameter. (Default: 'weight')
dangling: An unused parameter. (Default: None)
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.

Purpose
This method ranks the nodes present in the graph with the given parameters.

The parameters: "personalization", "nstart", "weight" and "dangling" are simply kept to make the method uniform to NetworkX pagerank method. They are not used anywhere within the frovedis implementation.

For example,

When loading an undirected networkx.graph as input from an edgelist file,

FILE: cit-Patents.txt

1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13

Here, the above file contains a list of edges between the nodes of graph G.

```
# An undirected graph loaded from edgelist file
import networkx as gnx
```

```
networkx_graph = gnx.read_edgelist('input/cit-Patents.txt')
ranks = gnx.pagerank(networkx_graph)
print ("NX PR: ", ranks)
```

Output

```
[rank 0] pagerank: converged in 8 iterations!
NX PR:  {1: 0.11705684590144233, 2: 0.06889632312740385, 3: 0.06889632312740385,
4: 0.06889632312740385, 5: 0.06889632312740385, 6: 0.06889632312740385, 7: 0.10702340365685098,
8: 0.06939799523963341, 9: 0.06939799523963341, 10: 0.06939799523963341,
11: 0.06939799523963341, 12: 0.07692307692307693, 13: 0.07692307692307693}
```

When loading an undirected frovedis graph as input from an edgelist file,

FILE: cit-Patents.txt (same file)

```
# An undirected graph loaded from edgelist file
import frovedis.graph as fnx
frov_graph = fnx.read_edgelist('input/cit-Patents.txt')
ranks = fnx.pagerank(frov_graph)
print ("FROV PR: ", ranks)
```

Output

```
[rank 0] pagerank: converged in 8 iterations!
FROV PR:  {1: 0.11705684590144233, 2: 0.06889632312740385, 3: 0.06889632312740385,
4: 0.06889632312740385, 5: 0.06889632312740385, 6: 0.06889632312740385, 7: 0.10702340365685098,
8: 0.06939799523963341, 9: 0.06939799523963341, 10: 0.06939799523963341,
11: 0.06939799523963341, 12: 0.07692307692307693, 13: 0.07692307692307693}
```

When loading an directed networkx.graph as input from an edgelist file,

FILE: cit-Patents.txt

```
1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13
```

Here, the above file contains a list of edges between the nodes of graph G.

```
# A directed graph loaded from edgelist file
import networkx as gnx
networkx_graph = gnx.read_edgelist('input/cit-Patents.txt', create_using = gnx.DiGraph())
ranks = gnx.pagerank(networkx_graph)
print ("NX PR: ", ranks)
```

Output

```
[rank 0] pagerank: converged in 3 iterations!
NX PR:  {1: 0.06538461538461539, 2: 0.06734615384615385, 3: 0.06734615384615385,
4: 0.06734615384615385, 5: 0.06734615384615385, 6: 0.06734615384615385, 7: 0.06538461538461539,
8: 0.06783653846153846, 9: 0.06783653846153846, 10: 0.06783653846153846,
11: 0.06783653846153846, 12: 0.06538461538461539, 13: 0.0751923076923077}
```

When loading an directed frovedis graph as input from an edgelist file,

FILE: cit-Patents.txt (same file)

```
# A directed graph loaded from edgelist file
import frovedis.graph as fnx
import networkx
frov_graph = fnx.read_edgelist('input/cit-Patents.txt', create_using = networkx.DiGraph())
ranks = fnx.pagerank(frov_graph)
print ("FROV PR: ", ranks)
```

Output

```
[rank 0] pagerank: converged in 3 iterations!
FROV PR:  {1: 0.06538461538461539, 2: 0.06734615384615385, 3: 0.06734615384615385,
4: 0.06734615384615385, 5: 0.06734615384615385, 6: 0.06734615384615385, 7: 0.06538461538461539,
8: 0.06783653846153846, 9: 0.06783653846153846, 10: 0.06783653846153846,
11: 0.06783653846153846, 12: 0.06538461538461539, 13: 0.0751923076923077}
```

Return Value
It returns a dictionary of nodes with PageRank as value.

41.4 SEE ALSO

- **Graph in Frovedis**
- **Breadth First Search in Frovedis**
- **Connected Components in Frovedis**
- **Single Source Shortest Path in Frovedis**

Chapter 42

Breadth First Search

42.1 NAME

Breadth First Search - Basic algorithms for breadth-first searching the nodes of a graph.

42.2 SYNOPSIS

```
frovedis.graph.traversal.bfs(G, source, depth_limit=None, opt_level=1, hyb_threshold=0.4,
                             verbose=0)

frovedis.graph.traversal.bfs_edges(G, source, reverse=False, depth_limit=None,
                                   sort_neighbors=None, opt_level=1, hyb_threshold=0.4,
                                   verbose=0)

frovedis.graph.traversal.bfs_tree(G, source, reverse=False, depth_limit=None,
                                  sort_neighbors=None, opt_level=1, hyb_threshold=0.4,
                                  verbose=0)

frovedis.graph.traversal.bfs_predecessors(G, source, depth_limit=None, sort_neighbors=None,
                                          opt_level=1, hyb_threshold=0.4, verbose=0)

frovedis.graph.traversal.bfs_successors(G, source, depth_limit=None, sort_neighbors=None,
                                        opt_level=1, hyb_threshold=0.4, verbose=0)

frovedis.graph.traversal.descendants_at_distance(G, source, distance, opt_level=1,
                                                 hyb_threshold=0.4, verbose=0)
```

42.3 DESCRIPTION

Breadth-first search is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis breadth first search public method interfaces are almost same as NetworkX breadth first serach

public method interface, but it doesn't have any dependency on NetworkX. It can be used simply even if the system doesn't have NetworkX 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.

42.3.1 Detailed Description

42.3.1.1 1. bfs()

Parameters
G: An instance of networkx graph or frovedis graph. The graph can be directed or undirected.
source: A positive integer parameter that specifies the starting node for breadth-first search. This method iterates over only those edges in the component, reachable from this node. It must be in range **[1, G.num_vertices]**.
depth_limit: A positive integer parameter that specifies the maximum search depth. (Default: None)
When it is None (not specified explicitly), it will be set as maximum value for int64 datatype.
opt_level: Zero or a positive integer parameter that must be in range 0 to 2. It is an optimization parameter that is used for reducing computation times during breadth first search. (Default: 1)
- **When opt_level = 0**: this should only be used where systems have memory constraints. It is slowest.
- **When opt_level = 1**: this is fastest. It uses comparatively large amount of memory.
- **When opt_level = 2**: this is much better than **'opt_level = 0'** but slightly slower than **opt_level = 1**. It optimizes the memory usage over **'opt_level = 1'**.

hyb_threshold: A double (float64) parameter that specifies a threshold value which performs optimization during breadth first search when the number of remaining nodes to be visited becomes less then this value. It optimizes the execution time. This parameter works only with **'opt_level = 2'**. It must be within the range 0 to 1. (Default: 0.4)
For example, if it is 0.5, then optimization starts when number of remaining nodes to be visited is less than 50%.
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.

Purpose
This method computes bfs traversal path of a graph.

This method is not present in Networkx. It is only provided in frovedis.

For example,

FILE: cit-Patents.txt

1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13

Here, the above file contains a list of edges between the nodes of graph G.

```
# A directed graph loaded from edgelist file
import numpy as np
```

```
import networkx as nx
import frovedis.graph as fnx
G = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                      create_using = nx.DiGraph())
print ("Frovedis BFS : ", list(fnx.bfs(G, source = 1, depth_limit = 1)))
```

Output

```
Frovedis BFS :  [{1: [1]}, {2: [1, 2]}, {3: [1, 3]}, {4: [1, 4]}, {5: [1, 5]}, {6: [1, 6]}]
```

In case, we started from node 7 i.e source = 7 in the directed graph 'G'

```
print ("Frovedis BFS : ", list(fnx.bfs(G, source = 7, depth_limit = 1)))
```

Output

```
Frovedis BFS :  [{7: [7]}, {8: [7, 8]}, {9: [7, 9]}, {10: [7, 10]}, {11: [7, 11]}]
```

Return Value
It returns a dictionary with keys as destination node-id and values as corresponding traversal path from the source. In case any node in input graph not reachable from the source, it would not be included in the resultant dictionary.

42.3.1.2 2. bfs_edges()

Parameters
G: An instance of networkx graph or frovedis graph. The graph can be directed or undirected.
source: A positive integer parameter that specifies the starting node for breadth-first search. This method iterates over only those edges in the component, reachable from this node. It must be in range [**1, G.num_vertices**].
reverse: A boolean parameter, in general, specifies the direction of traversal (forward or backward) if **'G'** is a directed graph. Currently, it can only traverse in forward direction **(reverse = True not supported yet)**. (Default: False)
depth_limit: A positive integer parameter that specifies the maximum search depth. (Default: None)
When it is None (not specified explicitly), it will be set as maximum value for int64 datatype.
sort_neighbors: An unused parameter. (Default: None)
opt_level: Zero or a positive integer parameter that must be in range 0 to 2. It is an optimization parameter that is used for reducing computation times during breadth first search. (Default: 1)
- **When opt_level = 0**: this should only be used where systems have memory constraints. It is slowest.
- **When opt_level = 1**: this is fastest. It uses comparatively large amount of memory.
- **When opt_level = 2**: this is much better than **'opt_level = 0'** but slightly slower than **opt_level = 1**. It optimizes the memory usage over **'opt_level = 1'**.

hyb_threshold: A double (float64) parameter that specifies a threshold value which performs optimization during breadth first search when the number of remaining nodes to be visited becomes less then this value. It optimizes the execution time. This parameter works only with **'opt_level = 2'**. It must be within the range 0 to 1. (Default: 0.4)
For example, if it is 0.5, then optimization starts when number of remaining nodes to be visited is less than 50%.
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.

Purpose
This method iterates over edges in a breadth-first-search starting at source.

The parameter: "sort_neighbors" is simply kept to make the method uniform to NetworkX bfs_edges method. It is not used anywhere within the frovedis implementation.

For example,

FILE: cit-Patents.txt

1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13

Here, the above file contains a list of edges between the nodes of graph G.

```
# A directed graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
G = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                      create_using = nx.DiGraph())
print("Frovedis BFS edges: ", list(fnx.bfs_edges(G, source = 1, depth_limit = 1)))
```

Output

```
Frovedis BFS edges:  [(1, 2), (1, 3), (1, 4), (1, 5), (1, 6)]
```

In case, we started from node 7 i.e source = 7 in the directed graph 'G'

```
print("Frovedis BFS edges: ", list(fnx.bfs_edges(G, source = 7, depth_limit = 1)))
```

Output

```
Frovedis BFS edges:  [(7, 8), (7, 9), (7, 10), (7, 11)]
```

Return Value
It yields edges resulting from the breadth-first search.

42.3.1.3 3. bfs_tree()

Parameters
G: An instance of networkx graph or frovedis graph. The graph can be directed or undirected.
source: A positive integer parameter that specifies the starting node for breadth-first search. This method iterates over only those edges in the component, reachable from this node. It must be in range **[1, G.num_vertices]**.
reverse: A boolean parameter, in general, specifies the direction of traversal (forward or backward) if **'G'** is a directed graph. Currently, it can only traverse in forward direction **(reverse = True not supported yet)**. (Default: False)
depth_limit: A positive integer parameter that specifies the maximum search depth. (Default: None)
When it is None (not specified explicitly), it will be set as maximum value for int64 datatype.
sort_neighbors: An unused parameter. (Default: None)
opt_level: Zero or a positive integer parameter that must be in range 0 to 2. It is an optimization parameter that is used for reducing computation times during breadth first search. (Default: 1)
- **When opt_level = 0**: this should only be used where systems have memory constraints. It is slowest.
- **When opt_level = 1**: this is fastest. It uses comparatively large amount of memory.
- **When opt_level = 2**: this is much better than **'opt_level = 0'** but slightly slower than **opt_level = 1**. It optimizes the memory usage over **'opt_level = 1'**.

hyb_threshold: A double (float64) parameter that specifies a threshold value which performs optimization during breadth first search when the number of remaining nodes to be visited becomes less then this value. It optimizes the execution time. This parameter works only with **'opt_level = 2'**. It must be within the range 0 to 1. (Default: 0.4)
For example, if it is 0.5, then optimization starts when number of remaining nodes to be visited is less than 50%.
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.

Purpose
This method constructs a directed tree from of a breadth-first-search starting at source.

The parameter: "sort_neighbors" is simply kept to make the method uniform to NetworkX bfs_tree method. It is not used anywhere within the frovedis implementation.

For example,

FILE: cit-Patents.txt

```
1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13
```

Here, the above file contains a list of edges between the nodes of graph G.

```
# A directed graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
G = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                      create_using = nx.DiGraph())
fnx.bfs_tree(G, source = 1, depth_limit = 1)

# To check edges in newly created networkx directed graph
print("Edges in Frovedis bfs_tree: ")
print(fnx.bfs_tree(G, source = 1, depth_limit = 1).number_of_edges())
```

Output

```
Edges in Frovedis bfs_tree:
5
```

In case, we started from node 7 i.e source = 7 in the directed graph 'G'

```
fnx.bfs_tree(G, source = 7, depth_limit = 1)

# To check edges in newly created networkx directed graph
print("Edges in Frovedis bfs_tree: ")
print(fnx.bfs_tree(G, source = 7, depth_limit = 1).number_of_edges())
```

Output

```
Edges in Frovedis bfs_tree:
```

4

Return Value
It returns a new Networkx.DiGraph instance.

42.3.1.4 4. bfs_predecessors()

Parameters
G: An instance of networkx graph or frovedis graph. The graph can be directed or undirected.
source: A positive integer parameter that specifies the starting node for breadth-first search. This method iterates over only those edges in the component, reachable from this node. It must be in range **[1, G.num_vertices]**.
depth_limit: A positive integer parameter that specifies the maximum search depth. (Default: None)
When it is None (not specified explicitly), it will be set as maximum value for int64 datatype.
sort_neighbors: An unused parameter. (Default: None)
opt_level: Zero or a positive integer parameter that must be in range 0 to 2. It is an optimization parameter that is used for reducing computation times during breadth first search. (Default: 1)
- **When opt_level = 0**: this should only be used where systems have memory constraints. It is slowest.
- **When opt_level = 1**: this is fastest. It uses comparatively large amount of memory.
- **When opt_level = 2**: this is much better than **'opt_level = 0'** but slightly slower than **opt_level = 1**. It optimizes the memory usage over **'opt_level = 1'**.

hyb_threshold: A double (float64) parameter that specifies a threshold value which performs optimization during breadth first search when the number of remaining nodes to be visited becomes less then this value. It optimizes the execution time. This parameter works only with **'opt_level = 2'**. It must be within the range 0 to 1. (Default: 0.4)
For example, if it is 0.5, then optimization starts when number of remaining nodes to be visited is less than 50%.
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.

Purpose
This method provides predecessors for each node in breadth-first-search from source.

The parameter: "sort_neighbors" is simply kept to make the method uniform to NetworkX bfs_tree method. It is not used anywhere within the frovedis implementation.

For example,

FILE: cit-Patents.txt

1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13

Here, the above file contains a list of edges between the nodes of graph G.

```
# A directed graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
```

```
G = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                      create_using = nx.DiGraph())
print("Frovedis bfs_predecessors: ")
print(list(fnx.bfs_predecessors(G, source = 1, depth_limit = 1)))
```

Output

```
Frovedis bfs_predecessors:
[(2, 1), (3, 1), (4, 1), (5, 1), (6, 1)]
```

Here, node 2, node 3, node 4, node 5 and node 6 were having source = 1 as thier predecessor.

In case, we started from node 7 i.e source = 7 in the directed graph 'G'

```
print("Frovedis bfs_predecessors: ")
print(list(fnx.bfs_predecessors(G, source = 7, depth_limit = 1)))
```

Output

```
Frovedis bfs_predecessors:
[(8, 7), (9, 7), (10, 7), (11, 7)]
```

Here, node 8, node 9, node 10 and node 11 were having source = 7 as thier predecessor.

Return Value
It yields predecessors for each node in form of **(node, predecessor)** iterator.

42.3.1.5 5. bfs_successors()

Parameters
G: An instance of networkx graph or frovedis graph. The graph can be directed or undirected.
source: A positive integer parameter that specifies the starting node for breadth-first search. This method iterates over only those edges in the component, reachable from this node. It must be in range **[1, G.num_vertices]**.
depth_limit: A positive integer parameter that specifies the maximum search depth. (Default: None)
When it is None (not specified explicitly), it will be set as maximum value for int64 datatype.
sort_neighbors: An unused parameter. (Default: None)
opt_level: Zero or a positive integer parameter that must be in range 0 to 2. It is an optimization parameter that is used for reducing computation times during breadth first search. (Default: 1)
- **When opt_level = 0**: this should only be used where systems have memory constraints. It is slowest.
- **When opt_level = 1**: this is fastest. It uses comparatively large amount of memory.
- **When opt_level = 2**: this is much better than **'opt_level = 0'** but slightly slower than **opt_level = 1**. It optimizes the memory usage over **'opt_level = 1'**.

hyb_threshold: A double (float64) parameter that specifies a threshold value which performs optimization during breadth first search when the number of remaining nodes to be visited becomes less then this value. It optimizes the execution time. This parameter works only with **'opt_level = 2'**. It must be within the range 0 to 1. (Default: 0.4)
For example, if it is 0.5, then optimization starts when number of remaining nodes to be visited is less than 50%.
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.

Purpose
This method provides successors for each node in breadth-first-search from source.

The parameter: "sort_neighbors" is simply kept to make the method uniform to NetworkX bfs_tree method. It is not used anywhere within the frovedis implementation.

For example,

FILE: cit-Patents.txt

1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13

Here, the above file contains a list of edges between the nodes of graph G.

```
# A directed graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
G = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                      create_using = nx.DiGraph())
print("Frovedis bfs_successors: ", list(fnx.bfs_successors(G, source = 1, depth_limit = 1)))
```

Output

```
Frovedis bfs_successors:  [(1, [2, 3, 4, 5, 6])]
```

Here, for source = 1, node 2, node 3, node 4, node 5 and node 6 were its successor.

In case, we started from node 7 i.e source = 7 in the directed graph 'G'

```
print("Frovedis bfs_successors: ", list(fnx.bfs_successors(G, source = 7, depth_limit = 1)))
```

Output

```
Frovedis bfs_successors:  [(7, [8, 9, 10, 11])]
```

Here, for source = 7, node 8, node 9, node 10 and node 11 were its successor.

Return Value
It yields successors for each node in form of **(node, successor)** iterator.

42.3.1.6 6. descendants_at_distance()

Parameters
G: An instance of networkx graph or frovedis graph. The graph can be directed or undirected.
source: A positive integer parameter that specifies the starting node for breadth-first search. This method iterates over only those edges in the component, reachable from this node. It must be in range [**1, G.num_vertices**].
distance: A positive integer parameter that specifies the distance of the wanted nodes from source.
opt_level: Zero or a positive integer parameter that must be in range 0 to 2. It is an optimization parameter is used for reducing computation times during breadth first search. (Default: 1)
- **When opt_level = 0**: this should only be used where systems have memory constraints. It is slowest.
- **When opt_level = 1**: this is fastest. It uses comparatively large amount of memory.
- **When opt_level = 2**: this is much better than **'opt_level = 0'** but slightly slower than **opt_level = 1**. It optimizes the memory usage over **'opt_level = 1'**.

hyb_threshold: A double (float64) parameter that specifies a threshold value which performs optimization during breadth first search when the number of remaining nodes to be visited becomes less then this value. It optimizes the execution time. This parameter works only with **'opt_level = 2'**. It must be within the

range 0 to 1. (Default: 0.4)
For example, if it is 0.5, then optimization starts when number of remaining nodes to be visited is less than 50%.
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.

Purpose
This method returns all nodes at a fixed distance from source in G.

For example,

FILE: cit-Patents.txt

1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13

Here, the above file contains a list of edges between the nodes of graph G.

```
# A directed graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
G = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                      create_using = nx.DiGraph())
print("Frovedis descendants at distance:")
print(fnx.descendants_at_distance(G, source = 1, distance = 1))
```

Output

```
Frovedis descendants at distance:
{2, 3, 4, 5, 6}
```

In case, we started from node 7 i.e source = 7 in the directed graph 'G'

```
print("Frovedis descendants at distance:")
print(fnx.descendants_at_distance(G, source = 7, distance = 1))
```

Output

```
Frovedis descendants at distance:
{8, 9, 10, 11}
```

Return Value
It returns an instance of Set having the descendants of source in graph 'G' at the given distance from source.

42.4 SEE ALSO

- **Graph in Frovedis**
- **Connected Components in Frovedis**

- **Single Source Shortest Path in Frovedis**
- **PageRank in Frovedis**

Chapter 43

single_source_shortest_path()

43.1 NAME

single_source_shortest_path() - finds the shortest path from source to all reachable nodes in graph 'G'. Here, graph 'G' maybe a directed or undirected graph.

43.2 SYNOPSIS

```
frovedis.graph.traversal.single_source_shortest_path(G, source, return_distance=False,
                                                     verbose=0)
```

43.3 DESCRIPTION

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis public single_source_shortest_path method interface is almost same as NetworkX single_source_shortest_path public method interface, but it doesn't have any dependency on NetworkX. It can be used simply even if the system doesn't have NetworkX 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 single_source_shortest_path() on the frovedis server. Once the shortest distances are computed for the given input graph data at the frovedis server, it returns a dictionary of lists containing shortest path from source to all other nodes to the client python program.

43.3.1 Detailed Description

43.3.1.1 1. single_source_shortest_path()

Parameters
G: An instance of networkx graph or frovedis graph. The graph can be directed or undirected.
source: A positive integer parameter that specifies the starting node for the path. It must be in the range

[1, G.num_vertices].
return_distance: A boolean parameter if set to True, will return the shortest distances from source to all nodes along with traversal path. Otherwise, it will return only the traversal path. (Default: False)
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.

Purpose
This method computes the shortest path between source and all other nodes reachable from source.

For example,

FILE: cit-Patents.txt

1 2
1 3
1 4
1 5
1 6
7 8
7 9
7 10
7 11
12 13

Here, the above file contains a list of edges between the nodes of graph G.

```
# A directed graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
frov_graph = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                     create_using = nx.DiGraph())
# use verbose = 1 for detailed inforrmation
path, dist = fnx.single_source_shortest_path(frov_graph, source = 1, return_distance=True
                                             verbose = 1)
print("Frovedis sssp traversal path: ")
print(list(path))
print("Frovedis sssp traversal distance from source: ")
print(dist)
```

Output

```
sssp computation time: 0.001 sec.
sssp res conversion time: 0.000 sec.
Frovedis sssp traversal path:
[{1: [1]}, {2: [1, 2]}, {3: [1, 3]}, {4: [1, 4]}, {5: [1, 5]}, {6: [1, 6]}]
Frovedis sssp traversal distance from source:
{1: 0.0, 2: 1.0, 3: 1.0, 4: 1.0, 5: 1.0, 6: 1.0}
```

Incase we had started with node 2 i.e source = 2, then,

```
# A directed graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
frov_graph = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                     create_using = nx.DiGraph())
path, dist = fnx.single_source_shortest_path(frov_graph, source = 2, return_distance=True)
print("Frovedis sssp traversal path: ")
```

```
print(list(path))
print("Frovedis sssp traversal distance from source: ")
print(dist)
```

Output

```
Frovedis sssp traversal path:
[{2: [2]}, {1: [2, 1]}, {3: [2, 1, 3]}, {4: [2, 1, 4]}, {5: [2, 1, 5]}, {6: [2, 1, 6]}]
Frovedis sssp traversal distance from source:
{2: 0.0, 1: 1.0, 3: 2.0, 4: 2.0, 5: 2.0, 6: 2.0}
```

When source = 2 and return_distances = False,

```
# A directed graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
frov_graph = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int32,
                      create_using = nx.DiGraph())
# return_distances = False, by default
path= fnx.single_source_shortest_path(frov_graph, source = 2)
print("Frovedis sssp traversal path: ")
print(list(path))
```

Output

```
Frovedis sssp traversal path:
[{2: [2]}, {1: [2, 1]}, {3: [2, 1, 3]}, {4: [2, 1, 4]}, {5: [2, 1, 5]}, {6: [2, 1, 6]}]
```

Return Value
It returns a dictionary of lists containing shortest path from source to all other nodes.

43.4 SEE ALSO

- **Graph in Frovedis**
- **Breadth First Search in Frovedis**
- **Connected Components in Frovedis**
- **PageRank in Frovedis**

Chapter 44

connected_components()

44.1 NAME

connected_components() - generates the connected components of an undirected graph 'G'.

44.2 SYNOPSIS

```
frovedis.graph.ConnectedComponents.connected_components(G, opt_level=2, hyb_threshold=0.4,
                                                        verbose=0, print_summary=False,
                                                        print_limit=5)
```

44.3 DESCRIPTION

In graph theory, a component of an undirected graph is a connected subgraph that is not part of any larger connected subgraph. The components of any graph partition its vertices into disjoint sets, and are the induced subgraphs of those sets. A graph that is itself connected has exactly one component, consisting of the whole graph. Components are also sometimes called connected components.

This module provides a client-server implementation, where the client application is a normal python program. The frovedis public connected_components method interface is almost same as NetworkX connected_components public method interface, but it doesn't have any dependency on NetworkX. It can be used simply even if the system doesn't have NetworkX 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 connected_components() on the frovedis server. Once the connected components are computed for the given input graph data at the frovedis server, it returns a dictionary with keys as root-nodeid for each component and values as list of pairs of nodeid with its distance from root of the component to which the node belongs.

44.3.1 Detailed Description

44.3.1.1 1. connected_components()

Parameters
G: An instance of networkx graph or frovedis graph. The graph must be undirected.
opt_level: Zero or a positive integer parameter that must be in range 0 to 2. It is an optimization parameter that is used for reducing computation time while generating the connected components. (Default: 1)
- **When opt_level = 0**: this should only be used where systems have memory constraints. It is slowest.
- **When opt_level = 1**: this is fastest. It uses comparatively large amount of memory.
- **When opt_level = 2**: this is much better than **'opt_level = 0'**, but slightly slower than **opt_level = 1**. It optimizes the memory usage over **'opt_level = 1'**.

hyb_threshold: A double (float64) parameter that specifies a threshold value which performs optimization in generating connected components when the number of remaining nodes to be visited becomes less then this value. It optimizes the execution time. This parameter works only with **'opt_level = 2'**. It must be within the range 0 to 1. (Default: 0.4)
For example, if it is 0.5, then optimization starts when number of remaining nodes to be visited is less than 50%.
verbose: An integer parameter specifying the log level to use. Its value is 0 by default (INFO level). But it can be set to 1 (DEBUG level) or 2 (TRACE level) for getting training time logs from frovedis server.
print_summary: A boolean parameter that specifies whether to print summary of connected components. (Default: False)
print_limit: An integer parameter that specifies the maximum number of nodes info to be printed. (Default: 5)

Purpose
This method computes connected components of an undirected graph.

For example,

```
# An undirected graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
frov_graph = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int64)
ret = fnx.connected_components(frov_graph)
for i in ret:
    print(i)
```

Output

```
{1, 2, 3, 4, 5, 6}
{7, 8, 9, 10, 11}
{12, 13}
```

When print_summary = True and print_limit = 5 (by default), then,

```
# An undirected graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
frov_graph = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int64)
ret = fnx.connected_components(frov_graph, print_summary = True)
for i in ret:
    print(i)
```

Output

```
Number of connected components: 3
Root with its count of nodes in each connected component: (root_id:count)
1:6
7:5
12:2
Nodes in which cc: (node_id:root_id)
1:1
2:1
3:1
4:1
5:1
...
Nodes dist: (node:level_from_root)
1:0
2:1
3:1
4:1
5:1
...
{1, 2, 3, 4, 5, 6}
{7, 8, 9, 10, 11}
{12, 13}
```

When print_limit = 10, then,

```
# An undirected graph loaded from edgelist file
import numpy as np
import networkx as nx
import frovedis.graph as fnx
frov_graph = fnx.read_edgelist('input/cit-Patents_10.txt', nodetype = np.int64)
ret = fnx.connected_components(frov_graph, print_summary = True, print_limit = 10)
for i in ret:
    print(i)
```

Output

```
Number of connected components: 3
Root with its count of nodes in each connected component: (root_id:count)
1:6
7:5
12:2
Nodes in which cc: (node_id:root_id)
1:1
2:1
3:1
4:1
5:1
6:1
7:7
8:7
9:7
10:7
...
Nodes dist: (node:level_from_root)
1:0
2:1
```

```
3:1
4:1
5:1
6:1
7:0
8:1
9:1
10:1
...
{1, 2, 3, 4, 5, 6}
{7, 8, 9, 10, 11}
{12, 13}
```

Return Value
It yields an output as Sets of nodes for each connected components.

44.4 SEE ALSO

- **Graph in Frovedis**
- **Breadth First Search in Frovedis**
- **Single Source Shortest Path in Frovedis**
- **PageRank in Frovedis**

Chapter 45

FrovedisDvector

45.1 NAME

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

45.2 SYNOPSIS

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

45.2.1 Public Member Functions

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

45.3 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.

45.3.1 Constructor Documentation

45.3.1.1 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.

45.3.2 Pubic Member Function Documentation

45.3.2.1 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.

45.3.2.2 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.

45.3.2.3 size()

Purpose
It returns the size of the dvector

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

45.3.2.4 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.

45.3.2.5 release()

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

Return Type
It returns nothing.

45.3.2.6 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 46

FrovedisCRSMatrix

46.1 NAME

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

46.2 SYNOPSIS

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

46.2.1 Public Member Functions

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

46.3 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.

46.3.1 Constructor Documentation

46.3.1.1 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.

46.3.2 Pubic Member Function Documentation

46.3.2.1 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.

46.3.2.2 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.

46.3.2.3 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.

46.3.2.4 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.

46.3.2.5 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.

46.3.2.6 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.

46.3.2.7 numRows()

Purpose
It returns the number of rows in the matrix

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

46.3.2.8 numCols()

Purpose
It returns the number of columns in the matrix

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

46.3.2.9 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.

46.3.2.10 release()

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

Return Type
It returns nothing.

46.3.2.11 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 47

FrovedisBlockcyclicMatrix

47.1 NAME

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

47.2 SYNOPSIS

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

47.2.1 Overloaded Operators

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

47.2.2 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()

47.3 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.

47.3.1 Constructor Documentation

47.3.1.1 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.

47.3.2 Overloaded Operators Documentation

47.3.2.1 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.

47.3.2.2 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.

47.3.2.3 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.

47.3.2.4 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.

47.3.2.5 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.

47.3.3 Pubic Member Function Documentation

47.3.3.1 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.

47.3.3.2 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.

47.3.3.3 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.

47.3.3.4 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.

47.3.3.5 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.

47.3.3.6 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.

47.3.3.7 transpose ()

Purpose

This function will compute the transpose of the given matrix.

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

47.3.3.8 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

47.3.3.9 numRows()

Purpose
It returns the number of rows in the matrix

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

47.3.3.10 numCols()

Purpose
It returns the number of columns in the matrix

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

47.3.3.11 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.

47.3.3.12 release()

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

Return Type
It returns nothing.

47.3.3.13 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 48

pblas_wrapper

48.1 NAME

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

48.2 SYNOPSIS

import frovedis.matrix.wrapper.PBLAS

48.2.1 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)

48.3 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".

48.3.1 Detailed Description

48.3.1.1 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.

48.3.1.2 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.

48.3.1.3 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.

48.3.1.4 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.

48.3.1.5 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.

48.3.1.6 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.

48.3.1.7 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.

48.3.1.8 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.

48.3.1.9 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.

48.3.1.10 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.

48.4 SEE ALSO

scalapack_wrapper, blockcyclic_matrix

Chapter 49

scalapack_wrapper

49.1 NAME

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

49.2 SYNOPSIS

import frovedis.matrix.wrapper.SCALAPACK

49.3 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)

49.4 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".

49.4.1 Detailed Description

49.4.1.1 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.

49.4.1.2 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.

49.4.1.3 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.

49.4.1.4 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.

49.4.1.5 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.

49.4.1.6 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.

49.5 SEE ALSO

blockcyclic_matrix, pblas_wrapper, arpack_wrapper, getrf_result, gesvd_result

Chapter 50

getrf_result

50.1 NAME

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

50.2 SYNOPSIS

import frovedis.matrix.results.GetrfResult

50.2.1 Public Member Functions

release()
ipiv()
stat()

50.3 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.

50.3.1 Public Member Function Documentation

50.3.1.1 release()

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

Return Type
It returns nothing.

50.3.1.2 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.

50.3.1.3 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 51

gesvd_result

51.1 NAME

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

51.2 SYNOPSIS

import frovedis.matrix.results.GesvdResult

51.2.1 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()

51.3 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.

51.3.1 Public Member Function Documentation

51.3.1.1 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.

51.3.1.2 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.

51.3.1.3 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.

51.3.1.4 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.

51.3.1.5 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.

51.3.1.6 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.

51.3.1.7 release()

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

Return Type
It returns nothing.

51.3.1.8 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.

51.3.1.9 getK()

Purpose
This function returns the number of singular values computed.

Return Type
An integer value.

Chapter 52

DataFrame Introduction

52.1 NAME

DataFrame - here refers to a Frovedis dataframe. It is a two-dimensional, size-mutable, potentially heterogeneous tabular data structure.

52.2 DESCRIPTION

Dataframe is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. This data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. It can be thought of as a dict-like container for Series objects or it can be thought of as an SQL table or a spreadsheet data representation.

Features of DataFrame:
- Columns can be of different types.
- DataFrame is mutable i.e. the number of rows and columns can be increased or decreased.
- DataFrame supports indexing and labeled columns name.
- Supports arithmetic operations on rows and columns.

Frovedis contains dataframe implementation over a client server architecture (which uses Vector Engine on server side to perform fast computations) where pandas dataframe will be converted to frovedis dataframe and then will be used to perform dataframe related operations.

In frovedis, currently dataframe can only support atomic data i.e. a column in a frovedis dataframe can only have primitve type values such as string, integer, float, double, boolean).

52.2.1 Detailed description

52.2.1.1 1. Creating frovedis DataFrame instance

a) Using DataFrame constructor

DataFrame(df = None)

Parameters
df: A pandas DataFrame instance or pandas Series instance. (Default: None)
When this parameter is not None (specified explicitly), it will load the pandas dataframe (or series) to perform conversion into frovedis dataframe.

Purpose
It is used to create a Frovedis dataframe from the given pandas dataframe or series.

Constructing frovedis DataFrame from pandas DataFrame:

For example,

```
# a pandas dictionary having key and values pairs
peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
           }

# creating a pandas dataframe from key value pair
pdf1 = pd.DataFrame(peopleDF)

# creating frovedis dataframe
fdf1 = fd.DataFrame(pdf1)

# display created frovedis dataframe
fdf1.show()
```

Output

```
index   Age     Country Ename   isMale
0       29      USA     Michael 0
1       30      England Andy    0
2       27      Japan   Tanaka  0
3       19      France  Raul    0
4       31      Japan   Yuta    1
```

Return Value
It returns self reference.

Attributes

columns: It returns a list of string having the column labels of the frovedis dataframe.

For example,

```
# fdf1 is same from above example
print(fdf1.columns)
```

Output

```
['Age', 'Country', 'Ename', 'isMale']
```

count: It returns an integer value having the number of rows in the frovedis dataframe

For example,

```
# fdf1 is same from above example
print(fdf1.count)
```

Output

```
5
```

dtypes: It returns the dtypes of all columns in the frovedis dataFrame. It returns a Series object with the data type of each column.

For example,

```
# fdf1 is same from above example
print(fdf1.dtypes)
```

Output

```
Age         int64
Country    object
Ename      object
isMale       bool
dtype: object
```

ndim: It returns an integer representing the number of axes / array dimensions.

For example,

```
# fdf1 is same from above example
print(fdf1.ndim)
```

Output

```
2
```

shape: It returns a tuple representing the dimensions of the frovedis dataframe in form: **(nrows, ncols)**

For example,

```
# fdf1 is same from above example
print(fdf1.shape)
```

Output

```
(5, 4)
```

values: It returns a numpy representation of the frovedis dataframe.

For example,

```
# fdf1 is same from above example
print(fdf1.values)
```

Output

```
[['Michael' '29' 'USA' '0']
['Andy' '30' 'England' '0']
['Tanaka' '27' 'Japan' '0']
['Raul' '19' 'France' '0']
['Yuta' '31' 'Japan' '1']]
```

b) By loading data using read_csv() method

read_csv(filepath_or_buffer, sep = ',', delimiter = None, header = "infer", names = None, index_col = None, usecols = None, squeeze = False, prefix = None, mangle_dupe_cols = True, dtype = None, na_values = None, verbose = False, comment = None, low_memory = True, rows_to_see = 1024, separate_mb = 1024, parse_dates=None, datetime_format=None, infer_datetime_format=False, date_parser=None, keep_date_col=False)

Note:- read_csv() requires pandas >= 1.2.0 as it utilizes 'guess_datetime_format' module for inferring date formats.

Parameters

filepath_or_buffer: It accepts any valid string value that contains the name of the file to access as parameter. The string path can be a URL as well. In case, a file is to be read, then the pathname can be absolute path or relative path to the current working directory of the file to be opened.

sep: It accepts string value as paramter that specifies delimiter to use. (Default: ',')
Currently, it supports only single character delimiter.
When it is None (if specified explicitly), it will use ',' as delimiter.
delimiter: It accepts string value as paramter. It is an alias for 'sep' parameter. (Default: None)
When it is None (not specified explicitly), it will set ',' as the delimiter.

header: It accepts integer or string value that specifies the row number(s) to use as the column names, and the start of the data. (Default: 'infer')
Currently, frovedis supports only 'infer', 0 or None as values for this parameter.
The default behavior is to **'infer'** the column names:

- **If no column names are passed**, then the behavior is identical to **header = 0** and column names are inferred from the first line of the file.

- **If column names are passed explicitly**, then the behavior is identical to **header = None** and column names are inferred from the explicitly specified list.

names: It accepts array like input as parameter that specifies the list of column names to use. (Default: None)
When it is None (not specified explicitly), it will get the column names from the first line of the file. Otherwise, it will override with the given list of column names. Since, **header = 0 or None is only supported in frovedis**, the first line of the file will be used as column names. **Duplicates in this list of column names are not allowed.**
index_col: It accepts boolean and integer type values as parameter. It specifies the column(s) to use as the row labels of the frovedis dataframe. **Currently, multi-index is not supported in frovedis.** (Default: None)

- If boolean type value is passed, then currently it can only be set as False. Also, when **index_col = False**, it can be used to force to not use the first column as the index. **Also, 'names' parameter must be provided**.

- It integer type value is passed, then it must be in the range **(0, nCols - 1)**, where **nCols** is the number of columns present in the input file.

usecols: It accepts currently a list-like input of integer values as parameter. It specifies the subset of the columns to be used while loading data into frovedis dataframe. (Default: None)
When it is None (not specified explicitly), it will use all columns while creating frovedis dataframe.
Also, if the column names are provided and do not match the number of columns in the input file, then the list of names must be same in length as the list of 'usecols' parameter values. Otherwise, it raises an exception.
squeeze: An unused parameter. (Default: False)
prefix: It accepts a string value as parameter. It specifies the prefix which needs to be added to the column numbers. (Default: None)

```
For example, 'X' for X0, X1, X2, ...
```

This is an optional paramter which will be used when column names are not provided and header is None (no header).
When it is None (not specified explicitly), it will create list of numbers as column labels of length N - 1, where **'N'** is the number of columns in the input file.
mangle_dupe_cols: It accepts boolean value as parameter that specifies the action taken for duplicate columns present in the input file. (Default: True)

- **When set to True (not specified explicitly) and if duplicate columns are present**, they will be specified as 'X', 'X.1', ... 'X.N', rather than 'X'... 'X'.

- **When set to False and duplicate columns are present**, it will raise an exeption **(not supported in frovedis yet)**.

dtype: It accepts datatype or dictionary of columns which is used for explicitly setting type for data or columns in frovedis dataframe. (Default: None)
For boolwan type, frovedis supports:

```
True/False, On/OFF, 0/1, yes/No, y/n
```

User must provide this, then frovedis can convert to boolean type. **Make sure user provides correct dtype , otherwise it will cause issues while creating frovedis dataframe**.
na_values: It accepts scalar value or array-like input as parameter that specifies the additional strings to recognize as missing values (NA/NaN). Currently, it does not accept dictionary as value for this parameter. (Default: None)
When it is None (not specified explicitly), it will interpret the list of following values as missing values:

```
['null', 'NULL', 'nan', '-nan', 'NaN', '-NaN', 'NA', 'N/A', 'n/a']
```

In case we want some other value to be interpreted as missing value, it is explicitly provided.
verbose: It accepts a bollean values as parameter that specifies the log level to use. Its value is **False by default (for INFO mode)** and it can be set to **True (for DEBUG mode)**. This is used for getting the loading time logs from frovedis server. It is useful for debugging purposes. (Default: False)
comment: This is an unused parameter. (Default: None)
low_memory: It accepts boolean value as parameter that specifies if the dataframe is to be loaded chunk wise. The chunk size then would be specified by the 'separate_mb' parameter. (Default: True)
rows_to_see: It accepts integer value as parameter. It is useful when creating frovedis dataframe by loading from large files. While using big files, only some rows would be used to infer the data types of columns. This paramter specifies the number of such rows which will used to infer data types. (Default: 1024)
separate_mb: It accepts double (float64) value as parameter. It is a memory optimization parameter. (Default: 1024)

- **If low_memory = True**, then internally frovedis dataframes of size **'separate_mb'** will be loaded separately and would be combined later into a single frovedis dataframe.

- **If low_memory = False**, then complete frovedis dataframe will be loaded at once.

parse_dates: List of columns names or positions to be parsed as datetime type. In order to combine multiple columns in file as a single datetime column, list of list is supported. (Default: None)

datetime_format: This is a frovedis specific parameter. The datetime format can be provided a string, list or dictionary for the respective columns. (Default: None)

infer_datetime_format: In case this parameter is set to True, the first hundred rows would be used to infer the datetime format. In case the datetime format cannot be inferred, the columns would be loaded unchanged as string type columns. (Default: False)

date_parser: This parameter is not supported. In order to specify the datetime format, **datetime_format** parameter can be used. (Default: None)

keep_date_col: In case this parameter is set to True and **parse_dates** specifies combining multiple columns then the original columns are kept in the resultant dataframe. (Default: False)

Purpose

This method reads a comma-separated values (csv) file into frovedis dataframe.

Note:- While loading file into frovedis dataframe, the file should not contain blank line at beginning.

The parameters: "squeeze" and "comment" are simply kept in to to make the interface uniform to the pandas read_csv() method. These are not used internally within the frovedis implementation.

Using read_csv() to load data from given csv file path into frovedis dataframe:

File: numbers.csv

10,10.23,F,0
12,12.20,nan,0
13,34.90,D,1
15,100.12,A,2

In order to load csv file will into frovedis dataframe,

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv")
df.show()
```

Output

```
index   10      10.23   F       0
0       12      12.2    NULL    0
1       13      34.8999 D       1
2       15      100.12  A       2
```

Here, by default it uses ',' delimiter while loading csv data into frovedis dataframe.

Also, since no column names were provided and header = 'infer', so the newly created frovedis dataframe contains column names inferred from the first line of input file.

Using read_csv() with header = None and no column name is provided:

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", header = None)
df.show()
```

Output

```
index   0       1       2       3
0       10      10.23   F       0
1       12      12.2    NULL    0
2       13      34.8999 D       1
3       15      100.12  A       2
```

Here, it creates list of column numbers which is used as column names in newly created frovedis dataframe.

Using read_csv() with header = None. Also, column names are provided:

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", names = ['one', 'two', 'three', 'four'], header = None)
df.show()
```

Output

```
index   one     two     three   four
0       10      10.23   F       0
1       12      12.2    NULL    0
2       13      34.8999 D       1
3       15      100.12  A       2
```

Here, given list of names will be used as column names in the newly generated frovedis dataframe.

Also, the input file contained no duplicate column names in its header.

Using read_csv() with prefix = True:

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", header = None, prefix = "X")
df.show()
```

Output

```
index   X0      X1      X2      X3
0       10      10.23   F       0
1       12      12.2    NULL    0
2       13      34.8999 D       1
3       15      100.12  A       2
```

Here, 'prefix' is an optional paremeter which will be in use when header = None and names = None.
It will use column numbers by default when prefix = None.

To load specific columns in frovedis dataframe, usecols parameter will be used:

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv",names=['one', 'two', 'three', 'four'], usecols = [1,2])
df.show()
```

Output

```
index   two     three
0       10.23   F
1       12.2    NULL
2       34.8999 D
3       100.12  A
```

Using read_csv() with dtype parameter:

File: numbers.csv

10,10.23,E,TRUE
12,12.20,nan,TRUE
13,34.90,D,FALSE
15,100.12,A,FALSE

In above data, **TRUE/FALSE** in python can be converted to boolean **(0/1)** type values using dtype parameter.

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", names = ['one', 'two', 'three', 'four'],
                  dtype = {'four': bool})
df.show()
```

Output

```
index   one     two     three   four
0       10      10.23   E       1
1       12      12.2    NULL    1
2       13      34.8999 D       0
3       15      100.12  A       0
```

In order to use second column as index, index_col parameter will be used:

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", names = ['one', 'two', 'three', 'four'], index_col = 1)
df.show()
```

Output

```
two     one     three   four
10.23   10      F       0
12.2    12      NULL    0
34.8999 13      D       1
100.12  15      A       2
```

When index_col = False, it will force to not use the first column as the index:

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", names = ['one', 'two', 'three', 'four'], index_col = False)
df.show()
```

Output

```
index   one     two     three   four
0       10      10.23   E       0
1       12      12.2    NULL    0
2       13      34.8999 D       1
3       15      100.12  A       2
```

Using mangle_dupe_cols = True, when input file contains duplicate columns in header:

File: numbers_dupl.csv

```
score,score,grade,pass
11.2,10.23,F,0
21.2,12.20,nan,0
43.6,34.90,D,1
75.1,100.12,A,1
```

The input file contains duplicate column labels. To make sure while loading data into frovedis dataframe, mangle_dupe_cols will be set as True.

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers_dupl.csv",mangle_dupe_cols = True)
df.show()
```

Output

```
index   score   score.1 grade   pass
0       10      10.23   F       0
1       12      12.2    NULL    0
2       13      34.8999 D       1
3       15      100.12  A       1
```

Using read_csv() with na_values parameter:

File: numbers.csv

```
11,10.23,E,0,1
12,12.20,F,0,1
10,34.90,D,1,0
15,10,A,10,0
```

10,10.23,B,0,1
10,10.23,B,0,1

When using na_values = None (by default), it will interpret the values nan, Nan, NULL, etc. as missing values (explained in parameter description)

```
# demo for read_csv() with na_values = None by default
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", na_values = None)
df.show()
```

Output

```
index   11      10.23   E       0       1
0       12      12.2    F       0       1
1       10      34.8999 D       1       0
2       15      10      A       10      0
3       10      10.23   B       0       1
4       10      10.23   B       0       1
```

When list of strings are provided to be set as missing values:

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", na_values = ['10','1'])
df.show()
```

Output

```
index   11      10.23   E       0       1
0       12      12.2    F       0       NULL
1       NULL    34.8999 D       NULL    0
2       15      NULL    A       NULL    0
3       NULL    10.23   B       0       NULL
4       NULL    10.23   B       0       NULL
```

It will interpret the list of values **['10','1']** as missing values (explicitly provided) while loading input file into frovedis dataframe.

When a string is to be set as missing value:

For example,

```
import frovedis.dataframe as fdf
df = fdf.read_csv("./input/numbers.csv", na_values = '10')
df.show()
```

Output

```
index   11      10.23   E       0       1
0       12      12.2    F       0       1
1       NULL    34.8999 D       1       0
2       15      NULL    A       NULL    0
3       NULL    10.23   B       0       1
4       NULL    10.23   B       0       1
```

It will interpret the string **'10'** as missing value (explicitly provided) while loading input file into frovedis dataframe.

Return Value
It returns a frovedis DataFrame instance.

52.2.1.2 2. Destructing the frovedis DataFrame instance

Using release() method

Purpose
This method acts like a destructor.
It is used to release dataframe pointer from server heap and it resets all its attributes to None.

Releasing dataframe pointers from frovedis:

For example,

```
# fdf1 is same from above example
fdf1.release()
```

Return Value
It returns nothing.

52.2.1.3 3. Public Member Functions of DataFrame

DataFrame provides a lot of utilities to perform various operations. For simplicity we have categorized them into three lists: ***Basic functions***, ***Application, groupby functions***, ***Aggregate functions*** and ***Binary operator functions***.

52.2.1.3.1 a) List of Basic Functions **Basic functions** are further categorized into two sub parts - ***conversion and sorting functions*** and ***selection and combination functions***.

In the basic functions, we will discuss the common and essential functionalities of dataframe like conversion of dataframes, sorting of data, selection of specified data and combining two or more data.

52.2.1.3.1.1 Conversion Functions: Conversion functions are essential part of DataFrame which are basically used to perform conversions to other types and to narrow down the data as per specification.

1. **asDF()** - Returns a Frovedis DataFrame after suitable conversion from other DataFrame types.
2. **to_dict()** - Convert the dataframe to a dictionary.
3. **to_numpy()** - Converts a frovedis dataframe to numpy array.
4. **to_pandas()** - Returns a pandas dataframe object from frovedis dataframe.
5. **to_frovedis_rowmajor_matrix()** - Converts a frovedis dataframe to FrovedisRowmajorMatrix.
6. **to_frovedis_colmajor_matrix()** - Converts a frovedis dataframe to FrovedisColmajorMatrix.
7. **to_frovedis_crs_matrix()** - Converts a frovedis dataframe to FrovedisCRSMatrix.
8. **to_frovedis_crs_matrix_using_info()** - Converts a frovedis dataframe to FrovedisCRSMatrix provided an info object of df_to_sparse_info class.

52.2.1.3.1.2 Sorting Functions: Sorting functions are essential part of DataFrame.

1. **nlargest()** - Return the first n rows ordered by columns in descending order.

2. **nsmallest()** - Return the first n rows ordered by columns in ascending order.
3. **sort()** - Sort by the values on a column.
4. **sort_index()** - Sort dataframes according to index.
5. **sort_values()** - Sort by the values along either axis.

52.2.1.3.1.3 Generic Functions: DataFrame provides various facilities to easily select and combine together specified values and support join/merge operations.

1. **add_index()** - Adds index column to the dataframe in-place.
2. **append()** - Union of dataframes according to rows.
3. **apply()** - Apply a function along an axis of the DataFrame.
4. **astype()** - Cast a selected column to a specified dtype.
5. **between()** - Filters rows according to the specified bound over a single column at a time.
6. **copy()** - Make a copy of the indices and data of this object.
7. **countna()** - Count NA values for each column/row.
8. **describe()** - Generate descriptive statistics.
9. **drop()** - Drop specified labels from rows or columns.
10. **drop_duplicates()** - Return DataFrame with duplicate rows removed.
11. **dropna()** - Remove missing values.
12. **fillna()** - Fill NA/NaN values using specified values.
13. **filter()** - Subset the dataframe rows or columns according to the specified index labels.
14. **get_index_loc()** - Returns integer location, slice or boolean mask for specified value in index column.
15. **head()** - Return the first n rows.
16. **insert()** - Insert column into DataFrame at specified location.
17. **isna()** - Detect missing values.
18. **isnull()** - Is an alias of isna().
19. **join()** - Join columns of another DataFrame.
20. **merge()** - Merge dataframes according to specified parameters.
21. **rename()** - Used to rename column.
22. **rename_index()** - Renames index field (inplace).

23. **reset_index()** - Reset the index.
24. **set_index()** - Set the DataFrame index using existing columns.
25. **show()** - Displays the selected dataframe values on console.
26. **tail()** - Return the last n rows.
27. **update_index()** - Updates/sets index values.

52.2.1.3.2 b) List of Aggregate Functions Aggregate functions of dataframe help to perform computations on the specified values and helps with efficient summarization of data. The calculated values gives insight into the nature of potential data.

1. **agg()** - Aggregate using the specified functions and columns.
2. **cov()** - Returns the pairwise covariance of columns, excluding missing values.
3. **mad()** - Returns the mean absolute deviation of the values over the requested axis.
4. **max()** - Returns the maximum of the values over the requested axis.
5. **mean()** - Returns the mean of the values over the requested axis.
6. **median()** - Returns the median of the values over the requested axis.
7. **min()** - Returns the minimum of the values over the requested axis.
8. **mode()** - Returns the mode(s) of each element along the selected axis.
9. **sem()** - Returns the unbiased standard error of the mean over requested axis.
10. **std()** - Returns the sample standard deviation over requested axis.
11. **sum()** - Returns the sum of the values over the requested axis.
12. **var()** - Returns unbiased variance over requested axis.

52.2.1.3.3 c) List of Math Functions DataFrame has methods for carrying out binary operations like add(), sub(), etc and related functions like radd(), rsub(), etc. for carrying out reverse binary operations.

1. **abs()** - Return a DataFrame with absolute numeric value of each element.
2. **add()** - Get addition of dataframe and other specified value. It is equivalent to dataframe + other.
3. **div()** - Get floating division of dataframe and other specified value. It is equivalent to dataframe / other.
4. **floordiv()** - Get integer division of dataframe and other specified value. It is equivalent to dataframe // other.
5. **mod()** - Get modulo of dataframe and other specified value. It is equivalent to dataframe % other.

6. **mul()** - Get multiplication of dataframe and other specified value. It is equivalent to dataframe * other.
7. **pow()** - Get exponential power of dataframe and other specified value. It is equivalent to dataframe ** other.
8. **sub()** - Get subtraction of dataframe and other specified value. It is equivalent to dataframe - other.
9. **truediv()** - Get floating division of dataframe and other specified value. It is equivalent to dataframe / other.
10. **radd()** - Get addition of other specified value and dataframe. It is equivalent to other + dataframe.
11. **rdiv()** - Get floating division of other specified value and dataframe. It is equivalent to other / dataframe.
12. **rfloordiv()** - Get integer division of other specified value and dataframe. It is equivalent to other // dataframe.
13. **rmod()** - Get modulo of other specified value and dataframe. It is equivalent to other % dataframe.
14. **rmul()** - Get multiplication of other specified value and dataframe. It is equivalent to other * dataframe.
15. **rpow()** - Get exponential power of other specified value and dataframe. It is equivalent to other ** dataframe.
16. **rsub()** - Get subtraction of other specified value and dataframe. It is equivalent to other - dataframe.
17. **rtruediv()** - Get floating division of other specified value and dataframe. It is equivalent to other / dataframe.

52.3 SEE ALSO

- **DataFrame - Data Extraction Methods**
- **DataFrame - Indexing Operations**
- **DataFrame - Conversion Functions**
- **DataFrame - Sorting Functions**
- **DataFrame - Math Functions**
- **DataFrame - Aggregate Functions**

Chapter 53

DataFrame Indexing Operations

53.1 NAME

Indexing in frovedis Dataframe - it means simply selecting particular rows and columns of data from a dataframe.

53.2 DESCRIPTION

There are a lot of ways to pull the rows and columns from a dataframe. Frovedis DataFrame class provides some indexing methods which help in filtering data from a dataframe. These indexing methods appear very similar but behave very differently. **Currently, frovedis supports the below mentioned types of indexing**. They are as follows:

- **Dataframe.**[ ]: This function is also known as **getitem**. It helps in filtering rows and columns from a dataframe.
- **Dataframe.loc**[]: It is primarily label based, but may also be used with a boolean array.
- **Dataframe.iloc**[]: It is primarily integer position based, but may also be used with a boolean array.
- **Dataframe.at**[]: Similar to loc, both provide label-based lookups. It is used when only needed to get a single value in a DataFrame.
- **Dataframe.iat**[]: Similar to iloc, both provide integer-based lookups. It is used when only needed to get a single value in a DataFrame.
- **Dataframe.take()**: This function is used to return the elements in the given positional indices along an axis.

53.2.1 Detailed description

1. Indexing using [ ]

Indexing operator is used to refer to the square brackets following an object.

In order to select a single column, we simply put the name of the column in-between the brackets.

Selecting a single column:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a pandas dataframe from key value pair
pdf2 = pd.DataFrame({ "Last Bonus": [5, 2, 2, 4],
                      "Bonus": [5, 2, 2, 4],
                      "Last Salary": [58, 59, 63, 58],
                      "Salary": [60, 60, 64, 59]
                    }, index= ["John", "Marry", "Sam", "Jo"]
                   )
# creating frovedis dataframe
fdf1 = fdf.DataFrame(pdf2)

# display created frovedis dataframe
fdf1.show()

print('selecting single column: ')
fdf1['Bonus'].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
Marry   2            2       59           60
Sam     2            2       63           64
Jo      4            4       58           59

selecting single column:
index   Bonus
John    5
Marry   2
Sam     2
Jo      4
```

Note:- Frovedis also supports use of attribute operators to filter/select column(s) in dataframe

In previous example, **Bonus** column can also be selected from the dataframe as follows:

For example,

```
fdf1.Bonus
```

This returns a FrovedisColumn instance.

To select multiple columns:

For example,

```
# selecting multiple columns
fdf1[['Bonus','Salary']].show()
```

Output

```
index   Bonus   Salary
John    5       60
Marry   2       60
Sam     2       64
```

```
Jo      4       59
```

Here, list of columns are passed in the indexing operator.

Filtering dataframe using slice operation with row numbers:

For example,

```
fdf1[1:2].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
Marry   2            2       59            60
```

Filtering dataframe using slice operation with row labels:

For example,

```
fdf1['John':'Sam'].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
Marry   2            2       59            60
```

Filtering can be done with help of attribute operators:

For example,

```
# filtering data using given condition
fdf1[fdf1.Bonus == 2].show()
```

Output

```
index   Last Bonus  Bonus  Last Salary  Salary
Marry   2           2      59           60
Sam     2           2      63           64
```

For example,

```
# filtering data using '>' operator
fdf1[fdf1.Bonus > 2].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
John    5            5       58            60
Jo      4            4       58            59
```

For example,

```
# filtering data using '<' operator
fdf1[fdf1.Bonus < 5].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
Marry   2            2       59            60
Sam     2            2       63            64
Jo      4            4       58            59
```

For example,

```
# filtering data using '!=' operator
fdf1[fdf1.Bonus != 2].show()
```

Output

```
index   Last Bonus    Bonus   Last Salary    Salary
John    5             5       58             60
Jo      4             4       58             59
```

For example,

```
# using '&' operation to filter data
fdf1[(fdf1.Bonus == 5) & (fdf1.Salary == 60)].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
John    5            5       58            60
```

For example,

```
# using '|' operation to filter data
fdf1[(fdf1.Bonus == 5) | (fdf1.Salary == 60)].show()
```

Output

```
index   Last Bonus    Bonus   Last Salary   Salary
John    5             5       58            60
Marry   2             2       59            60
```

For example,

```
# using '~' operation to filter data
fdf1[~(fdf1.Bonus == 5)].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
Marry   2            2       59            60
Sam     2            2       63            64
Jo      4            4       58            59
```

The above filtering expressions can also be done with the help of indexing operator.

For example,

```
# filtering data using given condition
fdf1[fdf1["Bonus"] == 2].show()
```

Output

```
index   Last Bonus  Bonus  Last Salary  Salary
Marry   2           2      59           60
Sam     2           2      63           64
```

For example,

```
# filtering data using '>' operator
fdf1[fdf1["Bonus"] > 2].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
John    5            5       58            60
Jo      4            4       58            59
```

2. Indexing a DataFrame using .loc[ ] :

This function selects data by the label of the rows and columns.

Allowed inputs are:

- A single label, e.g. 5 or 'a'. Here, 5 is interpreted as a label of the index, and not as an integer position along the index.
- A list or array of labels, e.g. ['a', 'b', 'c'].
- A slice object with labels, e.g. 'a':'f'. Here while using slices, both the start and the stop are included.
- A boolean array of the same length as number of rows, e.g. [True, False, True].

Currently, .loc[] cannot be used to set values for items in dataframe.

For example,

```
fdf1.loc['0',:] = 'Jai'              #not supported
fdf1.loc['0',:] = 12                 #not supported
```

The above expression will give an error.

Also, it cannot be used to filter data using given condition.

For example,

```
fdf1.loc[fdf1['Bonus'] > 2].show()   #not supported
```

The above expression will give an error.

Selecting a single column:

In order to select a single row using .loc[], we put a single row label in a .loc function.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a pandas dataframe from key value pair
pdf2 = pd.DataFrame({ "Last Bonus": [5, 2, 2, 4],
                      "Bonus": [5, 2, 2, 4],
                      "Last Salary": [58, 59, 63, 58],
                      "Salary": [60, 60, 64, 59]
                    }, index= ["John", "Marry", "Sam", "Jo"]
                   )
# creating frovedis dataframe
fdf1 = fdf.DataFrame(pdf2)

# display created frovedis dataframe
fdf1.show()

print('selecting single column: ')
fdf1.loc['John'].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
Marry   2            2       59           60
Sam     2            2       63           64
Jo      4            4       58           59
```

```
selecting single column:
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
```

To select multiple columns:

For example,

```
# selecting multiple columns
fdf1.loc[['Marry','Sam','Jo']].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
Marry   2            2       59           60
Sam     2            2       63           64
Jo      4            4       58           59
```

Filtering dataframe using slice operation with row labels:

For example,

```
fdf1.loc['Marry':'Sam'].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
Marry   2            2       59           60
Sam     2            2       63           64
```

Selecting two rows and three columns:

For example,

```
fdf1.loc['Marry':'Sam', ['Bonus', 'Last Salary', 'Salary']]
```

Output

```
index   Bonus    Last Salary   Salary
Marry   2        59            60
Sam     2        63            64
```

Selecting all rows and some columns:

For example,

```
fdf1.loc[:, ['Bonus', 'Last Salary', 'Salary']]
```

Output

```
index   Bonus    Last Salary   Salary
John    5        58            60
Marry   2        59            60
Sam     2        63            64
Jo      4        58            59
```

To select data using boolean list:

For example,

```
# selecting using boolean list
fdf1.loc[[True, False, True, False]].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
John    5            5       58            60
Sam     2            2       63            64
```

In case a dataframe having boolean indices is used, then .loc[] must be used with such boolean labels only.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a pandas dataframe from key value pair
pdf2 = pd.DataFrame({ "Last Bonus": [5, 2, 2, 4],
                      "Bonus": [5, 2, 2, 4],
                      "Last Salary": [58, 59, 63, 58],
                      "Salary": [60, 60, 64, 59]
                    }, index= [True, False, False, True]
                   )
# creating frovedis dataframe
fdf1 = fdf.DataFrame(pdf2)

# display created frovedis dataframe
fdf1.show()

fdf1.loc[[False, False, True, False]].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
1       5            5       58            60
0       2            2       59            60
0       2            2       63            64
1       4            4       58            59

index   Last Bonus   Bonus   Last Salary   Salary
0       2            2       63            64
```

3. Indexing a DataFrame using .iloc[ ] :

This function allows us to retrieve rows and columns by position.

The df.iloc indexer is very similar to df.loc but only uses integer locations to make its selections.

Allowed inputs are:

- An integer, e.g. 5.
- A list or array of integers, e.g. [4, 3, 0].
- A slice object with ints, e.g. 1:7.
- A boolean array of the same length as number of rows, e.g. [True, False, True].

Currently, .iloc[] cannot be used to set values for items in dataframe.

For example,

```
fdf1.iloc[0,:] = 'Jai'              #not supported
fdf1.iloc[0,:] = 12                 #not supported
```

The above expression will give an error.

Also, it cannot be used to filter data using given condition.

For example,

```
fdf1.iloc[fdf1['Bonus'] > 2].show()  #not supported
```

The above expression will give an error.

Selecting a single column:

In order to select a single row using .iloc[], we can pass a single integer to .iloc[] function.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a pandas dataframe from key value pair
pdf2 = pd.DataFrame({ "Last Bonus": [5, 2, 2, 4],
                      "Bonus": [5, 2, 2, 4],
                      "Last Salary": [58, 59, 63, 58],
                      "Salary": [60, 60, 64, 59]
                    }, index= ["John", "Marry", "Sam", "Jo"]
                   )
# creating frovedis dataframe
fdf1 = fdf.DataFrame(pdf2)

# display created frovedis dataframe
fdf1.show()

print('selecting single column: ')
fdf1.iloc[3].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
Marry   2            2       59           60
Sam     2            2       63           64
Jo      4            4       58           59

selecting single column:
index   Last Bonus   Bonus   Last Salary  Salary
Jo      4            4       58           59
```

To select multiple columns:

For example,

```
# selecting multiple columns
fdf1.iloc[[0, 1 ,3]].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
Marry   2            2       59           60
Jo      4            4       58           59
```

Filtering dataframe using slice operation with row labels:

For example,

```
fdf1.iloc[1:3].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
Marry   2            2       59           60
Sam     2            2       63           64
```

Selecting two rows and two columns:

For example,

```
fdf1.iloc[[1, 3],[2, 3]].show()
```

Output

```
index   Last Salary  Salary
Marry   59           60
Jo      58           59
```

Selecting all rows and some columns:

For example,

```
fdf1.iloc[:, [2, 3]].show()
```

Output

```
index   Last Salary   Salary
John    58            60
Marry   59            60
Sam     63            64
Jo      58            59
```

To select data using boolean list:

For example,

```
# selecting using boolean list
fdf1.iloc[[True, False, True, False]].show()
```

Output

```
index   Last Bonus   Bonus   Last Salary   Salary
John    5            5       58            60
Sam     2            2       63            64
```

4. Indexing a DataFrame using .at[ ] :

This function is used to access single value for a row/column label pair.

This method works in a similar way to .loc[ ] but .at[ ] is used to return an only single value and hence it works faster than .loc[].

Currently, .at[] cannot be used to set values for items in dataframe.

For example,

```
fdf1.at['John', 'Bonus'] = 52       #not supported
```

The above expression will give an error.

Accesing a single value:

In order to access a single value, we can pass a pair of row and column labels to .at[] function.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a pandas dataframe from key value pair
pdf2 = pd.DataFrame({ "Last Bonus": [5, 2, 2, 4],
                      "Bonus": [5, 2, 2, 4],
                      "Last Salary": [58, 59, 63, 58],
                      "Salary": [60, 60, 64, 59]
                    }, index= ["John", "Marry", "Sam", "Jo"]
                   )
# creating frovedis dataframe
fdf1 = fdf.DataFrame(pdf2)

# display created frovedis dataframe
fdf1.show()

print('selecting single value: ')
fdf1.at['John', 'Bonus']
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
Marry   2            2       59           60
Sam     2            2       63           64
Jo      4            4       58           59

selecting single value:
5
```

In case the given row/column label is not found then, it will give KeyError.

Also, it does not allow to perform slicing and will give an error.

5. Indexing a DataFrame using .iat[ ] :

This function is used to access single value for a row/column pair by integer position.

This method works in a similar way to .iloc[ ] but .iat[ ] is used to return an only single value and hence it works faster than .iloc[].

Currently, .iat[] cannot be used to set values for items in dataframe.

For example,

```
fdf1.iat[0, 2] = 52       #not supported
```

The above expression will give an error.

Accesing a single value:

In order to access a single value, we can pass a pair of row and column index positions to .iat[] function.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a pandas dataframe from key value pair
pdf2 = pd.DataFrame({ "Last Bonus": [5, 2, 2, 4],
                      "Bonus": [5, 2, 2, 4],
```

```
                      "Last Salary": [58, 59, 63, 58],
                      "Salary": [60, 60, 64, 59]
                    }, index= ["John", "Marry", "Sam", "Jo"]
                   )
# creating frovedis dataframe
fdf1 = fdf.DataFrame(pdf2)

# display created frovedis dataframe
fdf1.show()

print('selecting single value: ')
fdf1.iat[0, 2]
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
Marry   2            2       59           60
Sam     2            2       63           64
Jo      4            4       58           59

selecting single value:
58
```

In case the given row/column index position is not found then, it will give IndexError.

Also, it does not allow to perform slicing and will give an error.

6. Indexing a DataFrame using take() :

DataFrame.take(indices, axis = 0, is_copy = None, **kwargs)

Parameters
indices: An array of integers indicating which positions to take.
axis: It accepts an integer or string object as parameter. It is used to select elements. (Default: 0)
- 0 or 'index': used to select rows.
- 1 or 'columns': used to select columns.

is_copy: This is an unused parameter. (Default: None)
****kwargs***: This is an unused parameter.

Purpose
It returns the elements in the given positional indices along an axis.

Here, indexing is not according to actual values in the index attribute of the object rather indexing is done according to the actual position of the element in the object.

Selecting elements along axis = 0 (default) :

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a pandas dataframe from key value pair
pdf2 = pd.DataFrame({ "Last Bonus": [5, 2, 2, 4],
                      "Bonus": [5, 2, 2, 4],
                      "Last Salary": [58, 59, 63, 58],
                      "Salary": [60, 60, 64, 59]
                    }, index= ["John", "Marry", "Sam", "Jo"]
```

```
                    )
# creating frovedis dataframe
fdf1 = fdf.DataFrame(pdf2)

# display created frovedis dataframe
fdf1.show()

print('selecting along axis = 0: ')
fdf1.take(indices = [0, 2], axis = 0)
```

Output

```
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
Marry   2            2       59           60
Sam     2            2       63           64
Jo      4            4       58           59

selecting along axis = 0:
index   Last Bonus   Bonus   Last Salary  Salary
John    5            5       58           60
Sam     2            2       63           64
```

Selecting elements along axis = 1 (column selection) :

For example,

```
fdf1.take(indices = [1, 2], axis = 1)
```

Output

```
index   Bonus   Last Salary
John    5       58
Marry   2       59
Sam     2       63
Jo      4       58
```

Selecting elements from the end along axis = 0:

For example,

```
fdf1.take(indices = [-1, -2], axis = 0)
```

Output

```
index   Last Bonus  Bonus   Last Salary    Salary
Jo      4           4       58             59
Sam     2           2       63             64
```

Return Value
It returns a frovedis DataFrame instance or FrovedisColumn instance.

53.3 SEE ALSO

- **DataFrame - Introduction**
- **DataFrame - Generic Functions**

- **DataFrame - Conversion Functions**
- **DataFrame - Sorting Functions**
- **DataFrame - Math Functions**
- **DataFrame - Aggregate Functions**

Chapter 54

DataFrame Generic Functions

54.1 NAME

DataFrame Generic Functions - it contalins list of all generally used functions with frovedis dataframe.

54.1.1 DESCRIPTION

Frovedis dataframe provides various functions which are generally/frequently used with it to perform operations:
- Selecting / Filtering / Modifying data in frovedis dataframe.
- Combining two frovedis dataframes.
- Dropping data from rows or columns ina frovedis dataframe.
- Display frovedis dataframe (full/partially)

54.1.2 Public Member Functions

```
1. append(other, ignore_index = False, verify_integrity = False, sort = False)
2. apply()
3. astype(dtype, copy = True, errors = 'raise', check_bool_like_string = False)
4. between(left, right, inclusive="both")
5. copy(deep = True)
6. countna()
7. describe()
8. drop(labels = None, axis = 0, index = None, columns = None, level = None,
        inplace = False, errors = 'raise')
9. drop_duplicates(subset = None, keep = 'first', inplace = False, ignore_index = False)
10. dropna(axis = 0, how = 'any', thresh = None, subset = None, inplace = False)
11. fillna(value = None, method = None, axis = None, inplace = False, limit = None, downcast = None)
12. filter(items = None, like = None, regex = None, axis = None)
13. first_element(col_name, skipna = None)
14. get_index_loc(value)
15. head(n = 5)
16. insert(loc, column, value, allow_duplicates = False)
17. isna()
18. isnull()
19. join(right, on, how = 'inner', lsuffix = '_left', rsuffix = '_right',
```

```
        sort = False, join_type = 'bcast')
20. last_element(col_name, skipna = None)
21. merge(right, on = None, how = 'inner', left_on = None, right_on = None,
          left_index = False, right_index = False, sort = False, suffixes = ('_x', '_y'),
          copy = True, indicator = False, join_type = 'bcast')
22. rename(columns, inplace = False)
23. reset_index(drop = False, inplace = False)
24. set_index(keys, drop = True, append = False, inplace = False, verify_integrity = False)
25. show()
26. tail(n = 5)
```

54.1.3 Detailed Description

54.1.3.1 1. DataFrame.append(other, ignore_index = False, verify_integrity = False, sort = False)

Parameters
other: It accepts a Frovedis DataFrame instance or a Pandas DataFrame instance or a list of Frovedis DataFrame instances which are to be appended.
ignore_index: It accepts a boolean type parameter. If True, old index axis is ignored and a new index axis is added with values 0 to n - 1, where n is the number of rows in the DataFrame. (Default: False)
verify_integrity: It accepts a boolean type as parameter. If it is set to True, it checks 'index' label for duplicate entries before appending and when there are duplicate entries in the DataFrame, it does not append. Otherwise duplicate entries in the 'index' label will be appended. It will also append duplicate entries when 'ignore_index' = True. (Default: False)
sort: It accepts a boolean type as parameter. It sorts the columns, if the columns of the given DataFrame and other DataFrame are not aligned. (Default: False)

Purpose
It is used to append entries of dataframe at the end of another dataframe. The columns of other DataFrame instance that are not in the calling DataFrame instance are added as new columns.

Creating two frovedis dataframes:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df1 = pd.DataFrame([[1, 2], [3, 4]], columns = list('AB'), index = ['x', 'y'])

# create a frovedis dataframe
fd_df1 = fdf.DataFrame(pd_df1)

# display frovedis dataframe
fd_df1.show()

# create another pandas dataframe
pd_df2 = pd.DataFrame([[5, 6], [7, 8]], columns = list('AB'), index = ['x', 'y'])

# create another frovedis dataframe
fd_df2 = fdf.DataFrame(pd_df2)

# display other frovedis dataframe
```

```
fd_df2.show()
```

Output

```
index   A       B
x       1       2
y       3       4

index   A       B
x       5       6
y       7       8
```

Appending fd_df2 to fd_df1:

For example,

```
# example of using append method with default values
# appending fd_df2 to fd_df1
fd_df1.append(fd_df2).show()
```

Output

```
index   A       B
x       1       2
y       3       4
x       5       6
y       7       8
```

With ignore_index set to True:

For example,

```
# append() demo With ignore_index = True
fd_df1.append(fd_df2, ignore_index = True).show()
```

Output

```
index   A       B
0       1       2
1       3       4
2       5       6
3       7       8
```

Creating two frovedis dataframes with no common columns:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df1 = pd.DataFrame([[1, 2], [3, 4]], columns = list('AB'), index = ['x', 'y'])

# create a frovedis dataframe
fd_df1 = fdf.DataFrame(pd_df1)

# display frovedis dataframe
fd_df1.show()

# create another pandas dataframe
pd_df2 = pd.DataFrame([[5, 6], [7, 8]], columns = list('CD'), index = ['x', 'y'])
```

```
# create another frovedis dataframe
fd_df2 = fdf.DataFrame(pd_df2)

# display other frovedis dataframe
fd_df2.show()
```

Output

```
index   A       B
x       1       2
y       3       4

index   C       D
x       5       6
y       7       8
```

Append fd_df2 to fd_df1 when there are no common columns in frovedis dataframes:

For example,

```
fd_df1.append(fd_df2).show()
```

Output

```
index   A       B       D       C
x       1       2       NULL    NULL
y       3       4       NULL    NULL
x       NULL    NULL    6       5
y       NULL    NULL    8       7
```

Creating two frovedis dataframes where both have few duplicate entries:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df1 = pd.DataFrame([[1, 2], [3, 4]], columns=list('AB'), index=['x', 'y'])

# creating a frovedis dataframe
fd_df1 = fdf.DataFrame(pd_df1)

# display a frovedis dataframe
fd_df1.show()

# creating another pandas dataframe
pd_df2 = pd.DataFrame([[1, 2], [7, 8]], columns=list('AB'), index=['u', 'v'])

# creating another frovedis dataframe
fd_df2 = fdf.DataFrame(pd_df2)

# display other frovedis dataframe
fd_df2.show()
```

Output

```
index   A       B
x       1       2
```

```
y       3       4

index   A       B
u       1       2
v       7       8
```

Appending fd_df2 to fd_df1 with verify_integrity = True:

For example,

```
fd_df1.append(fd_df2, verify_integrity = True).show()
```

Output

```
index   A       B
x       1       2
y       3       4
u       1       2
v       7       8
```

Creating two frovedis dataframes to perform append and using sort=True:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df1 = pd.DataFrame([[1, 2], [3, 4]], columns=list('YZ'), index=['x', 'y'])

# creating a frovedis dataframe
fd_df1 = fdf.DataFrame(pd_df1)

# display a frovedis dataframe
fd_df1.show()

# creating another pandas dataframe
pd_df2 = pd.DataFrame([[1, 2], [7, 8]], columns=list('AB'), index=['x', 'y'])

# creating another frovedis dataframe
fd_df2 = fdf.DataFrame(pd_df2)

# display other frovedis dataframe
fd_df2.show()
```

Output

```
index   Y       Z
x       1       2
y       3       4

index   A       B
x       1       2
y       7       8
```

Appending fd_df2 with fd_df1 with sort = True:

For example,

```
fd_df1.append(fd_df2, sort = True).show()
```

Output

```
index   A       B       Y       Z
x       NULL    NULL    1       2
y       NULL    NULL    3       4
x       1       2       NULL    NULL
y       7       8       NULL    NULL
```

Return Value
It returns a new Frovedis DataFrame consisting of the rows of original DataFrame object and the rows of other DataFrame object.

54.1.3.2 2. DataFrame.apply(func, axis = 0, raw = False, result_type = None, args = (), **kwds)

Parameters
func: Names of functions to be applied on the data. The input to be used with the function must be a frovedis DataFrame instance having atleast one numeric column.
Accepted combinations for this parameter are:
- A string function name such as 'max', 'min', etc.
- list of functions and/or function names, For example, ['max', 'mean'].
- dictionary with keys as column labels and values as function name or list of such functions.
For Example, {'Age': ['max','min','mean'], 'Ename': ['count']}

axis: It accepts an integer as parameter. It is used to decide whether to perform aggregate operation along the columns or rows. (Default: 0)
raw: It accepts boolean as parameter. When set to True, the row/column will be passed as an ndarray. (Default: False)
result_type: It accepts string object as parameter. It specifies how the result will be returned. (Default: None)
These only act when **axis = 1 (columns)**:
- **expand** : list-like results will be turned into columns.
- **reduce** : returns a Series if possible rather than expanding list-like results. This is the opposite of 'expand'.
- **broadcast** : results will be broadcast to the original shape of the DataFrame, the original index and columns will be retained.

The default behaviour (None) depends on the return value of the applied function. List-like results will be returned as a Series of those. However if the apply function returns a Series these are expanded to columns.
args: Positional arguments to pass to 'func'. (Default: ())
*****kwds***: This is an unused parameter.

Purpose
Apply a function along an axis of the DataFrame.

The parameter: "**kwds" is simply kept in to make the interface uniform to the pandas DataFrame.apply(). This is not used anywhere within the frovedis implementation.

It actually uses pandas.apply() internally.

Basically, it converts the pandas dataframe into frovedis dataframe and after performing apply(), it returns the result back as pandas dataframe.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf
```

```
# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Apply function along the rows (by default) and where func is string name:

For example,

```
print(fdf1.apply('max'))
```

Output

```
Name             Princi
Age                  36
City             Nagpur
Qualification       Phd
Score                52
dtype: object
```

Apply function along the columns and where func is string name:

For example,

```
print(fdf1.apply('max', axis = 1))
```

Output

```
index
0    27.0
1    34.0
2    35.0
3    45.0
```

```
4    33.0
5    50.0
6    52.0
7    32.0
dtype: float64
```

Apply multiple functions along the rows (by default) and where func is a list of functions:

For example,

```
print(fdf1.apply(['max','min']))
```

Output

```
      Name    Age  City        Qualification  Score
max   Princi  36   Nagpur      Phd            52.0
min   Abhi    22   Allahabad   B.Tech         23.0
```

Return Value

1. **If only one 'func' provided:**
   - It returns a pandas Series instance with numeric column(s) only, after aggregation function is completed.

2. **If more than one 'func' provided:**
   - It returns a pandas DataFrame instance with numeric column(s) only, after aggregation function is completed.

54.1.3.3 3. DataFrame.astype(dtype, copy = True, errors = 'raise', check_bool_like_string = False)

Parameters
dtype: It accepts a string, numpy.dtype or a dict of column labels to cast entire DataFrame object to same type or one or more columns to column-specific types.
copy: It accepts a boolean parameter and returns a new DataFrame object when it is True. Currently this parameter will always return a copy. The original DataFrame is not modified. (Default: True)
errors: This is an unused parameter. (Default: 'raise')
check_bool_like_string: A boolean parameter which when set to True will cast string columns having boolean like case-insensitive strings (True, False, yes, No, On, Off, Y, N, T, F) to boolean columns. (Default: False)

Note:- astype() requires pandas >= 1.2.0 as it utilizes 'guess_datetime_format' module for inferring date formats.

Purpose
It cast an entire Frovedis DataFrame or selected columns of Frovedis DataFrame to the specified dtype.

The parameter: "errors" is simply kept in to make the interface uniform to the pandas DataFrame.astype(). This is not used anywhere within the frovedis implementation.

Creating a frovedis DataFrame from pandas DataFrame and display all column dtypes:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df =  pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
```

```
# creating a frovedis dataframe
fd_df1 = fdf.DataFrame(pf1)

# displaying a frovedis dataframe dtype
print(fd_df1.dtypes)
```

Output

```
col1    int64
col2    int64
dtype: object
```

Cast all columns to int32 type:

For example,

```
fd_df2 = fd_df1.astype('int32')

# display dataframe after conversion
print(fd_df2.dtypes)
```

Output

```
col1    int32
col2    int32
dtype: object
```

Creating a new frovedis DataFrame and display all column dtypes:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
    'Age' : [29, 30, 27, 19, 31],
    'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
    'isMale': [False, False, False, False, True]
          }
# converting to pandas dataframe
pd_df = pd.DataFrame(peopleDF)

# converting to frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display the datatype of object
print(fd_df.dtypes)
```

Output

```
Ename      object
Age         int64
Country    object
isMale       bool
dtype: object
```

Cast 'Age' column to int32 type using a dictionary:

For example,

```
print(fd_df.astype({'Age':'int32'}).dtypes)
```

Output

```
Ename      object
Age         int32
Country    object
isMale      int32
dtype: object
```

Creating a new frovedis DataFrame and using check_bool_like_string parameter:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
    'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
    'Age' : [29, 30, 27, 19, 31],
    'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
    'isMale': ['F', 'No', 'Off', False, 'Y']
       }
# converting to pandas dataframe
pd_df = pd.DataFrame(peopleDF)

# converting to frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display the dataframes
fd_df.show()

# display the datatype
print(fd_df.dtypes)
```

Output

```
index   Ename   Age     Country isMale
0       Michael 29      USA     F
1       Andy    30      England No
2       Tanaka  27      Japan   Off
3       Raul    19      France  False
4       Yuta    31      Japan   Y

Ename      object
Age         int64
Country    object
isMale     object
dtype: object
```

With check_bool_like_string = True:

For example,

```
# astype() demo with check_bool_like_string = True
fd_df.astype({'isMale':'bool'}, check_bool_like_string = True).show()

# display the datatype
```

```
print(fd_df.astype({'isMale':'bool'}, check_bool_like_string = True).dtypes)
```

Output

```
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
4       Yuta    31      Japan   1

Ename      object
Age         int64
Country    object
isMale       bool
dtype: object
```

Return Value
It returns a new DataFrame instance with dtype converted as specified.

54.1.3.4 4. DataFrame.between(left, right, inclusive = "both")

Parameters
left: It accepts scalar values as parameter.
right: It accepts scalar values as parameter.
inclusive: It accepts string object as parameter tha specifies which boundaries to include, whether to set bounds as open or closed. (Default: 'both')
- **'left'**: left boundary value is included.
- **'right'**: right boundary value is included.
- **'both'**: boundary values are included.
- **'neither'**: boundary values are excluded.

Purpose
This method performs filtering of rows according to the specified bound over a single column at a time.

Currently, this method filters data for numeric column data only.

Creating a frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
           }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)
```

```
# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification   Score
0       Jai     27      Nagpur     B.Tech          23
1       Anuj    24      Kanpur     Phd             34
2       Jai     22      Allahabad  B.Tech          35
3       Princi  32      Kannuaj    Phd             45
4       Gaurav  33      Allahabad  Phd             NULL
5       Anuj    36      Kanpur     B.Tech          50
6       Princi  27      Kanpur     Phd             52
7       Abhi    32      Kanpur     B.Tech          NULL
```

Using between function where both boundaries are provided:

For example,

```
# between() demo used with given column of dataframe
# Also, both boundaries are included
fdf1['Score'].between(40,50)
```

Unlike pandas, it returns an instance of dfoperator **(present in frovedis only)** which contains masked boolean data as result of between operation. In order to unmask the data, **to_mask_array()** must be used.

```
fdf1['Score'].between(40,50).to_mask_array
```

Output

```
[False False False  True False  True False False]
```

This returns a boolean array containing True wherever the corresponding element is between the boundary values 'left' and 'right'. The missing values are treated as False.

Also, this method can be used in the form of given expression below:

For example,

```
# between() demo used with given column of dataframe
# Also, both boundaries are included
res = fdf1[fdf1['Score'].between(40,50)]
print(res)
```

Output

```
index   Name    Age     City    Qualification   Score
3       Princi  32      Kannuaj Phd     45
5       Anuj    36      Kanpur  B.Tech  50
```

It can also be expressed as follows:

For example,

```
print(fdf1[fdf1.Score.between(40,50)])
```

Output

```
index   Name    Age     City    Qualification   Score
3       Princi  32      Kannuaj Phd     45
```

```
5       Anuj    36      Kanpur  B.Tech  50
```

With inclusive='left':

For example,

```
# between() demo and only left boundary included
print(fdf1['Score'].between(40,50,inclusive = 'left').to_mask_array())
```

Output

```
[False False False  True False False False False]
```

With inclusive='right':

For example,

```
# between() demo and only right boundary included
print(fdf1['Score'].between(40,50,inclusive = 'right').to_mask_array())
```

Output

```
[False False False  True False  True False False]
```

With inclusive='neither':

For example,

```
# between() demo and both boundaries excluded
print(fdf1['Score'].between(40,50,inclusive = 'neither').to_mask_array())
```

Output

```
[False False False  True False False False False]
```

Return Value
It returns a dfoperator instance.

54.1.3.5 5. DataFrame.copy(deep = True)

Parameters
deep: A boolean parameter to decide the type of copy operation. (Default: True)
When it is True (not specified explicitly), it creates a deep copy i.e. the copy includes copy of data and indices of the original DataFrame. **Currently this parameter does not support shallow copy (deep = False).**

Purpose
It creates a deep copy of the Frovedis DataFrame object.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

peopleDF = {
    'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
    'Age' : [29, 30, 27, 19, 31],
    'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
    'isMale': [False, False, False, False, True]
       }
# creating a pandas dataframe
```

```
pd_df = pd.DataFrame(peopleDF)

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

print('displaying original dataframe object')
fd_df.show()
```

Output

```
displaying original dataframe object
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
4       Yuta    31      Japan   1
```

Creating a deep copy:

For example,

```
fd_df_copy = fd_df.copy()

print('displaying copied dataframe object')
fd_df_copy.show()
```

Output

```
displaying copied dataframe object
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
4       Yuta    31      Japan   1
```

In order to check changes done in copy are reflected in original, we can do the following:

For example,

```
# changing column Age to Age2 for the copied object
fd_df_copy.rename({'Age':'Age2', inplace = True}

print('displaying copied dataframe object')
fd_df_copy.show()
```

Output

```
displaying copied dataframe object
index   Ename   Age2     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
4       Yuta    31      Japan   1
```

NOTE: changes are reflected only in copied DataFrame instance but not in original DataFrame instance.

Return Value
It returns a deep copy of the DataFrame instance of the same type.

54.1.3.6 6. DataFrame.countna(axis = 0)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to count missing values along the indices or by column labels. (Default: 0)
- **0 or 'index'**: count missing values along the indices.
- **1 or 'columns'**: count missing values along the columns.

Purpose
It counts number of missing values in the given axis.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Counting number of missing values along the rows:

For example,

```
# countna() demo
fdf1.countna().show()
```

Output

```
index   count
Name    0
Age     0
City    0
Qualification   0
Score   2
```

Counting number of missing values along the columns:

For example,

```
# countna() demo using axis = 1
fdf1.countna(axis = 1).show()
```

Output

```
index   count
0       0
1       0
2       0
3       0
4       1
5       0
6       0
7       1
```

Return Value
It returns a frovedis DataFrame instance.

54.1.3.7 7. DataFrame.describe()

Purpose
It generates descriptive statistics. Descriptive statistics include count, mean, median, etc, excluding missing values.

Currenlty, it only numeric fields are returned.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }
```

```
# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Describing a DataFrame. Only numeric fields are returned:

For example,

```
print(fdf1.describe())
```

Output

```
              Age       Score
count     8.000000    6.000000
mean     29.125000   39.833333
median   29.500000   40.000000
var      23.553571  123.766667
mad       4.125000    9.166667
std       4.853202   11.125047
sem       1.715866    4.541781
sum     233.000000  239.000000
min      22.000000   23.000000
max      36.000000   52.000000
```

Return Value
It returns a pandas DataFrame instance with the result of the specified aggregate operation.

54.1.3.8 8. DataFrame.drop(labels = None, axis = 0, index = None, columns = None, level = None, inplace = False, errors = 'raise')

Parameters
labels: It takes an integer or string type as argument. It represents column labels and integer represent index values of rows to be dropped. If any of the label is not found in the selected axis, it will raise an exception. (Default: None)
When it is None (not specified explicitly), 'index' or 'columns' parameter must be provided. Otherwise it will drop specific rows of the DataFrame.
axis: It accepts an integer type parameter where the value indicates the direction of drop operation according to below conditions:

- 0 : The corresponding index labels will be dropped.
- 1 : The corresponding columns labels will be dropped.

index: It accepts an integer or string object as parameter. It is equivalent to dropping indices along the axis = 0. (Default: None)
When it is None (not specified explicitly), 'columns' or 'labels' must be provided.
columns: It accepts a string object or list of strings. It is equivalent to dropping columns along the axis = 1. (Default: None)
When it is None (not specified explicitly), 'index' or 'labels' must be provided.
level: This is an unsed parameter. (Default: None)
inplace: It accepts a boolean as a parameter. It return a copy of DataFrame instance by default but when explicitly set to True, it performs operation on original DataFrame. (Default: False)
errors: This is an unsed parameter. (Default: 'raise')

Purpose
It is used to drop specified labels from rows or columns.
Rows or columns can be removed by specifying label names and corresponding axis, or by specifying index or column names.

Creating a frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

pd_df = pd.DataFrame(np.arange(12).reshape(3, 4), columns = ['A', 'B', 'C', 'D'])

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df).

# display frovedis dataframe
fd_df.show()
```

Output

```
index   A       B       C       D
0       0       1       2       3
1       4       5       6       7
2       8       9       10      11
```

Drop columns using axis parameter:

For example,

```
# drop() demo with label and axis parameter
fd_df.drop(labels = ['B', 'C'], axis = 1).show()
```

Output

```
index   A       D
0       0       3
1       4       7
2       8       11
```

Other way to drop columns:

For example,

```
# drop() demo with columns parameter
```

```
fd_df.drop(columns = ['B', 'C']).show()
```

Output

```
index   A       D
0       0       3
1       4       7
2       8       11
```

Drop a row by index:

For example,

```
fd_df.drop(labels = [0, 1]).show()
```

Output

```
index   A       B       C       D
2       8       9       10      11
```

Return Value

- It returns a new of Frovedis DataFrame having remaining entries.
- It returns None when parameter 'inplace' = True.

54.1.3.9 9. DataFrame.drop_duplicates(subset = None, keep = 'first', inplace = False, ignore_index = False)

Parameters
subset: It accepts a string object or a list of strings which only consider certain columns for identifying duplicates. (Default: None)
When it is None (not specified explicitly), it will consider all of the columns.
keep: It accepts a string object which is used to determine which duplicates values to keep. (Default: 'first')
1. **'first'** : Drop duplicates except for the first occurrence.
2. **'last'** : Drop duplicates except for the last occurrence.

inplace: It accepts a boolean as a parameter. It return a copy of DataFrame instance by default but when explicitly set to True, it performs operation on original DataFrame. (Default: False)
ignore_index: It accepts a boolean type parameter. If True, old index axis is ignored and a new index axis is added with values 0 to n - 1, where n is the number of rows in the DataFrame. (Default: False)

Purpose
It is used to remove duplicate rows.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# Consider dataset containing ramen rating.
pd_df = pd.DataFrame({
        'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],
        'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
        'rating': [4, 4, 3.5, 15, 5]})

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)
```

```
# display frovedis dataframe
fd_df.show()
```

Output

```
index   brand   style   rating
0       Yum Yum cup     4
1       Yum Yum cup     4
2       Indomie cup     3.5
3       Indomie pack    15
4       Indomie pack    5
```

Drop duplicates along the rows (by default):

For example,

```
# drop_duplicates() demo
fd_df.drop_duplicates()
```

Output

```
index   brand   style   rating
0       Yum Yum cup     4
2       Indomie cup     3.5
3       Indomie pack    15
4       Indomie pack    5
```

By default, it removes duplicate rows based on all columns

To remove duplicates on specific column(s), use subset parameter:

For example,

```
# drop_duplicates() demo with subset parameter
fd_df.drop_duplicates(subset = ['brand']).show()
```

Output

```
index   brand   style   rating
0       Yum Yum cup     4
2       Indomie cup     3.5
```

To remove duplicates and keep last occurrences, use keep parameter:

For example,

```
# drop_duplicates() demo with parameters: subset and keep = 'last'
fd_df.drop_duplicates(subset = ['brand', 'style'], keep = 'last').show()
```

Output

```
index   brand   style   rating
1       Yum Yum cup     4
2       Indomie cup     3.5
4       Indomie pack    5
```

Return Value

- It returns a new of Frovedis DataFrame having remaining entries.
- It returns None when parameter 'inplace' = True.

54.1.3.10 10. DataFrame.dropna(axis = 0, how = 'any', thresh = None, subset = None, inplace = False)

Parameters
axis: It accepts an integer value that can be 0 or 1. This parameter is used to determine whether rows or columns containing missing values are to be removed. (Default: 0)

- 0 : Drop rows which contain missing values.
- 1 : Drop columns which contain missing values.

how: It accepts a string object to determine if row or column is removed from DataFrame, when we have at least one 'NaN' or all 'NaN'. (Default: 'any')

- 'any' : If any NaN values are present, drop that row or column.
- 'all' : If all values are NaN, drop that row or column.

thresh: It accepts an integer as parameter which is the number of NaN values required for rows/columns to be dropped. (Default: None)
subset: It accepts a python ndarray. It is the name of the labels along the other axis which is being considered.
For example, if you are dropping rows, then these would be a list of columns. (Default: None)
inplace: This parameter accepts a boolean value. When it is set to True, then it performs operation on the original Frovedis DataFrame object itself otherwise operation is performed on a new Frovedis DataFrame object. (Default: False)

Purpose
It is used to remove missing values from the Frovedis DataFrame.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],
              "toy": [np.nan, 'Batmobile', 'Bullwhip'],
          "born": [np.nan, "1940-04-25", np.nan]})

# creating a frovedis dataframe from pandas dataframe
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
print("Before dropping nan values")
fd_df.show()
```

Output

```
Before dropping nan values
index   name            toy             born
0       Alfred          NULL            NULL
1       Batman          Batmobile       1940-04-25
2       Catwoman        Bullwhip        NULL
```

Drop the rows where at least one missing value is present:

For example,

```
print("After dropping nan values")
fd_df.dropna().show()
```

Output

```
After dropping nan values
index   name    toy             born
1       Batman  Batmobile       1940-04-25
```

Drop the columns where at least one missing value is present:

For example,

```
# display frovedis dataframe
print("Before dropping nan values")
fd_df.show()

print("After dropping nan values")
fd_df.dropna(axis=1)
```

Output

```
Before dropping nan values
index   name            toy             born
0       Alfred          NULL            NULL
1       Batman          Batmobile       1940-04-25
2       Catwoman        Bullwhip        NULL

After dropping nan values
index   name
0       Alfred
1       Batman
2       Catwoman
```

Drop the rows where all elements are missing:

For example,

```
# display frovedis dataframe
print("Before dropping nan values")
fd_df.show()

# drop the rows where all elements are missing (how='all')
fd_df.dropna(how='all').show()
```

Output

```
Before dropping nan values
index   name            toy             born
0       NULL            NULL            NULL
1       Batman          Batmobile       1940-04-25
2       Catwoman        Bullwhip        NULL

index   name            toy             born
1       Batman          Batmobile       1940-04-25
2       Catwoman        Bullwhip        NULL
```

Keep only the rows with at least 2 non-NA values:

For example,

```
# display frovedis dataframe
print("Before dropping nan values")
fd_df.show()

# example of using drop with parameter 'thresh'
# dropna() demo with only rows with at least 2 non-NA values (thresh = 2) to keep
fd_df.dropna(thresh=2).show()
```

Output

```
Before dropping nan values
index   name            toy             born
0       Alfred          NULL            NULL
1       Batman          Batmobile       1940-04-25
2       Catwoman        Bullwhip        NULL

index   name            toy             born
1       Batman          Batmobile       1940-04-25
2       Catwoman        Bullwhip        NULL
```

NOTE: Since 0th index had 2 missing values, so it was dropped, but other rows are not dropped.

To specify the columns to look for missing values:

For example,

```
# display frovedis dataframe
print("Before dropping nan values")
fd_df.show()

# to drop missing values only from specified columns
fd_df.dropna(subset=[1, 2], axis = 1).show()
```

Output

```
Before dropping nan values
index   name            toy             born
0       Alfred          NULL            NULL
1       Batman          Batmobile       1940-04-25
2       Catwoman        Bullwhip        NULL

index   name            toy
0       Alfred          NULL
1       Batman          Batmobile
2       Catwoman        Bullwhip
```

To keep the dataframe with valid entries in the same variable:

For example,

```
# display frovedis dataframe
print("Before dropping nan values")
fd_df.show()

# dropna() demo with inplace = True
fd_df.dropna(inplace=True)
fd_df.show()
```

Output

```
Before dropping nan values
index   name            toy     born
0       Alfred          NULL    NULL
1       Batman          Batmobile       1940-04-25
2       Catwoman        Bullwhip        NULL

index   name    toy             born
1       Batman  Batmobile       1940-04-25
```

Return Value

- If inplace = False, it returns a new Frovedis DataFrame with NA entries dropped.
- If inplace = True, it returns None.

54.1.3.11 11. DataFrame.fillna(value = None, method = None, axis = None, inplace = False, limit = None, downcast = None)

Parameters
value: It accepts a numeric type as parameter which is used to replace all NA values (e.g. 0, NULL). (Default: None)
When it is None (not specified explicitly), it must be NaN value or numeric, non-numeric value otherwise it will raise an exception.
method: This is an unused parameter. (Default: None)
axis: It accepts an integer or string object as parameter. It decides the axis along which missing values will be filled. (Default: None)
- 0 or 'index': Operation will be carried out on rows. **Currently only axis = 0 is supported in Frovedis DataFrame**.
inplace: This parameter accepts a boolean value. When it is set to True, then it performs operation on the original Frovedis DataFrame object itself otherwise operation is performed on a new Frovedis DataFrame object. (Default: False)
limit: This is an unused parameter. (Default: None)
downcast: This is an unused parameter. (Default: None)

Purpose
It replaces NA/NaN values with the specified value provided in 'value' parameter in the Frovedis DataFrame.

The parameters: "method", "limit" and "downcast" are simply kept in to make the interface uniform to the pandas DataFrame.fillna().
This is not used anywhere within the frovedis implementation.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                      [3, 4, np.nan, 1],
                      [np.nan, np.nan, np.nan, 5],
                      [np.nan, 3, np.nan, 4]],
                     columns=list("ABCD"))

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)
```

```
# display frovedis dataframe
fd_df.show()
```

Output

```
index   A       B       C       D
0       NULL    2       NULL    0
1       3       4       NULL    1
2       NULL    NULL    NULL    5
3       NULL    3       NULL    4
```

Replace all missing values with 0:

For example,

```
fd_df.fillna(0).show()
```

Output

```
index   A       B       C       D
0       0       2       0       0
1       3       4       0       1
2       0       0       0       5
3       0       3       0       4
```

Replace all missing values with 1 and keep the dataframe in the same variable:

For example,

```
# replace all NaN elements with -1s and inplace = True
fd_df.fillna(-1, inplace = True)

# display after modifying the original object
fd_df.show()
```

Output

```
index   A       B       C       D
0       NULL    2       NULL    0
1       3       4       NULL    1
2       NULL    NULL    NULL    5
3       NULL    3       NULL    4

index   A       B       C       D
0       -1      2       -1      0
1       3       4       -1      1
2       -1      -1      -1      5
3       -1      3       -1      4
```

Return Value

- It returns a Frovedis DataFrame object with missing values replaced when 'inplace' parameter is False.
- It returns None when 'inplace' parameter is set to True.

54.1.3.12 12. DataFrame.filter(items = None, like = None, regex = None, axis = None)

Parameters
items: It accepts a list of string as parameter. It filters only those labels which are mentioned. (Default:

None)
When it is None (not specified explicitly), 'like' or 'regex' must be provided.
like: It accepts a string object parameter. It keeps the column labels if 'like in label == True'. (Default: None)
When it is None (not specified explicitly), 'items' or 'regex' must be provided.
regex: It accepts a regular expression as a string parameter. It keeps the column labels if 're.search(regex, label) == True'. (Default: None)
When it is None (not specified explicitly), 'items' or 'like' must be provided.
axis: It accepts an integer or string type parameter. It specifies the axis on which filter operation will be performed. (Default: None)
- 0 or 'index' : The corresponding index labels will be filtered. **Currently this is not supported in Frovedis**.
- 1 or 'columns': The corresponding columns labels will be filtered.

When it is None (not specified explicitly), it will filter along axis = 1 or 'columns'.

Purpose
It is used to filter the DataFrame according to the specified column labels.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

pd_df = pd.DataFrame(np.array(([1, 2, 3], [4, 5, 6])),
                  index=['mouse', 'rabbit'],
                  columns=['one', 'two', 'three'])
fd_df = fdf.DataFrame(pd_df)

# display the frovedis dataframe
fd_df.show()
```

Output

```
index   one     two     three
mouse   1       2       3
rabbit  4       5       6
```

Select columns by name:

For example,

```
# filter() demo with items parameter
# select columns by name
fd_df.filter(items = ['one', 'three']).show()
```

Output

```
index   one     three
mouse   1       3
rabbit  4       6
```

Select columns by regular expression:

For example,

```
fd_df.filter(regex='e$', axis=1).show()
```

Output

```
index   one     three
mouse   1       3
rabbit  4       6
```

The output displayed only those columns whose label ends with 'e'.

Select column containing 'hre':

For example,

```
fd_df.filter(like='hre', axis=1)
```

Output

```
index   three
mouse   3
rabbit  6
```

Return Value
It returns a new Frovedis DataFrame instance with the column labels that matches the given conditions.

54.1.3.13 13. DataFrame.first_element(col_name, skipna = None)

Parameters
col_name: It accepts a string parameter as column name.
skipna: It accepts boolean as parameter. When set to True, it will exclude the missing values while fetching the first element from dataframe. (Default: None)
When it is None (not specified explicitly), it excludes missing values while fetching the first element.

Purpose
It fetches the first element from the given column (numeric/non-numeric) in a dataframe.

By default, it will exclude missing values while fetching first element from the given column.

It is present only in frovedis.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd',
                             'B.Tech', 'Phd', 'B.Tech'],
            'Score': [np.nan, 34, 35, 45, np.nan, 50, 52, np.nan]
           }

pd_df = pd.DataFrame(peopleDF)
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   Name    Age     City       Qualification   Score
0       Jai     27      Nagpur     B.Tech          NULL
1       Anuj    24      Kanpur     Phd             34
2       Jai     22      Allahabad  B.Tech          35
3       Princi  32      Kannuaj    Phd             45
4       Gaurav  33      Allahabad  Phd             NULL
5       Anuj    36      Kanpur     B.Tech          50
6       Princi  27      Kanpur     Phd             52
7       Abhi    32      Kanpur     B.Tech          NULL
```

Fetching the first element from 'Score' column (numeric column) :

For example,

```
fdf1.first_element(col_name='Score').show()
```

Output

```
34
```

Also, the above call can be expressed in two ways as follows,

For example,

```
fdf1.Score.first_element().show()
```

Output

```
34
```

For example,

```
fdf1["Score"].first_element().show()
```

Output

```
34
```

Fetching the first element from 'Name' column (non-numeric column) :

For example,

```
fdf1.first_element(col_name='Name').show()
```

Output

```
Jai
```

Using skipna=False in order to fetch first element (including Nan) from 'Score' column:

For example,

```
fdf1.first_element(col_name='Score', skipna=False).show()
```

Output

```
nan
```

Return Value
It returns a scalar value.

54.1.3.14 14. DataFrame.get_index_loc(value)

Parameters
value: It accepts an integer or string parameter. It is the index value whose location is to be determined.

Purpose
It provides integer location, slice or boolean mask for requested label.

It is present only in frovedis.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
           }
pd_df = pd.DataFrame(peopleDF)
pd_df.index = ['a', 'b', 'c', 'd', 'e']
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   Ename   Age     Country isMale
a       Michael 29      USA     0
b       Andy    30      England 0
c       Tanaka  27      Japan   0
d       Raul    19      France  0
e       Yuta    31      Japan   1
```

Get index location when there are unique indices:

For example,

```
# getting index location of 'd' index value
fd_df.get_index_loc('d').show()
```

Output

```
3
```

Creating a new frovedis DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
           }
# creating a pandas dataframe
pd_df = pd.DataFrame(peopleDF)
pd_df.index = ['a', 'a', 'd', 'd', 'e']
```

```
# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display a frovedis dataframe
fd_df.show()
```

Output

```
index   Ename   Age     Country isMale
a       Michael 29      USA     0
a       Andy    30      England 0
d       Tanaka  27      Japan   0
d       Raul    19      France  0
e       Yuta    31      Japan   1
```

Get index location when there are consecutive duplicate indices:

For example,

```
# getting index location of 'd' index value
fd_df.get_index_loc('d').show()
```

Output

```
slice(2, 4, None)
```

Creating another frovedis DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
           }
# creating a pandas dataframe
pd_df = pd.DataFrame(peopleDF)
pd_df.index = ['a', 'b', 'c', 'd', 'a']

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display a frovedis dataframe
fd_df.show()
```

Output

```
index   Ename   Age     Country isMale
a       Michael 29      USA     0
b       Andy    30      England 0
c       Tanaka  27      Japan   0
d       Raul    19      France  0
a       Yuta    31      Japan   1
```

Get index location when there are duplicate indices randomly:

For example,

```
# getting index location of 'a' index value
fd_df.get_index_loc('a').show()
```

Output

```
[ True False False False  True]
```

Return Value
It returns the following values:
1. **integer:** when there is a unique index.
2. **slice:** when there is a monotonic index i.e. repetitive values in index.
3. **mask:** it returns a list of boolean values.

54.1.3.15 15. DataFrame.head(n = 5)

Parameters
n: It accepts an integer parameter which represents the number of rows to select. (Default: 5)

Purpose
This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

For negative values of n, this function returns all rows except the last n rows, equivalent to df[:-n].

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating the dataframe
pd_df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',
                      'monkey', 'parrot', 'shark', 'whale', 'zebra']
                     })

#creating frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   animal
0       alligator
1       bee
2       falcon
3       lion
4       monkey
5       parrot
6       shark
7       whale
8       zebra
```

Viewing the first 5 lines (by default n=5):

For example,

```
# head() demo with default n value
fd_df.head().show()
```

Output

```
index   animal
0       alligator
1       bee
2       falcon
3       lion
4       monkey
```

Viewing the first 2 lines (n=2):

For example,

```
# head() demo with n = 2
fd_df.head(2).show()
```

Output

```
index   animal
0       alligator
1       bee
```

For negative values of n:

For example,

```
# head() demo with n = -3
fd_df.head(-3).show()
```

Output

```
index   animal
0       alligator
1       bee
2       falcon
3       lion
4       monkey
5       parrot
```

Return Value

- It n is positive integer, it returns a new DataFrame with the first n rows.
- If n is negative integer, it returns a new DataFrame with all rows except last n rows.

54.1.3.16 16. DataFrame.insert(loc, column, value, allow_duplicates = False)

Parameters
loc: It accepts an integer as parameter which represents the Insertion index. It must be in range **(0, n - 1)** where **n** is number of columns in dataframe.
column: It accepts a string object as parameter. It is the label of the inserted column.
value: It accepts an integer or a pandas Series instance or python ndarray as parameter. These are the values to be inserted in the specified 'column'.
allow_duplicates: It accepts a boolean value as parameter. Currently, frovedis does not support duplicate column names. (Default: False)

Purpose
It is used to insert column into DataFrame at specified location.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

pd_df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
fd_df = fdf.DataFrame(pd_df)
fd_df.show()
```

Output

```
index   col1    col2
0       1       3
1       2       4
```

Inserting [99, 99] at 1st index in 'newcol' column:

For example,

```
fd_df.insert(1, "newcol", [99, 99])

# display frovedis dataframe after insertion
fd_df.show()
```

Output

```
index   col1    newcol  col2
0       1       99      3
1       2       99      4
```

Inserting a Series object having values [5,6] at 0th index in "col0" column:

For example,

```
fd_df.insert(0, "col0", pd.Series([5, 6])).show()
```

Output

```
index   col0    col1    col2
0       5       1       3
1       6       2       4
```

Return Value
It returns a self reference.

54.1.3.17 17. DataFrame.isna()

Purpose
This method is used to detect missing values in the frovedis dataframe.

It returns a boolean same-sized object indicating if the values are NA. NA values, such as None or numpy.NaN, gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings '' or numpy.inf are not considered NA values.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# show which entries in a DataFrame are NA.
peopleDF = {
            'Ename' : ['Michael', None, 'Tanaka', 'Raul', ''],
            'Age' : [29, 30, 27, 19, 0],
            'Country' : ['USA', np.inf, 'Japan', np.nan, 'Japan'],
            'isMale': [False, False, False, False, True]
           }

# creating a pandas dataframe
pd_df = pd.DataFrame(peopleDF)

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# to display frovedis dataframe
fd_df.show()
```

Output

```
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       NULL    30      inf     0
2       Tanaka  27      Japan   0
3       Raul    19      NULL    0
4               0       Japan   1
```

Show which entries in a frovedis DataFrame are NA:

For example,

```
# isna() demo to display fields which are only NA
fd_df.isna().show()
```

Output

```
index   Ename   Age     Country isMale
0       0       0       0       0
1       1       0       0       0
2       0       0       0       0
3       0       0       1       0
4       0       0       0       0
```

Creating frovedis DataFrame from pandas Series:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# frovedis dataframe from a pandas Series object.
ser = pd.Series([5, 6, np.NaN])

# creating a frovedis dataframe
fd_df = fdf.DataFrame(ser)

# display frovedis dataframe
```

```
fd_df.show()
```

Output

```
index   0
0       5
1       6
2       NULL
```

Show which entries in corresonding frovedis DataFrame are NA:

For example,

```
# isna() demo to display na values mapped to corresponding dataframe
fd_df.isna().show()
```

Output

```
index   0
0       0
1       0
2       1
```

Return Value
It returns a new Frovedis DataFrame having all boolean values (0, 1) corresponding to each of the Frovedis DataFrame values depending on whether it is a valid NaN (True i.e. 1) value or not (False i.e. 0).

54.1.3.18 18. DataFrame.isnull()

Purpose
This method is used to detect missing values in the frovedis dataframe. It is an alias of isna().

It returns a boolean same-sized object indicating if the values are NA. NA values, such as None or numpy.NaN, gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings '' or numpy.inf are not considered NA values.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# show which entries in a DataFrame are NA.
peopleDF = {
    'Ename' : ['Michael', None, 'Tanaka', 'Raul', ''],
    'Age' : [29, 30, 27, 19, 0],
    'Country' : ['USA', 'England', 'Japan', np.nan, 'Japan'],
    'isMale': [False, False, False, False, True]
          }

# creating a pandas dataframe
pd_df = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       NULL    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      NULL    0
4               0       Japan   1
```

Show which entries in a frovedis DataFrame are NULL:

For example,

```
# isnull() demo
fd_df.isnull().show()
```

Output

```
index   Ename   Age     Country isMale
0       0       0       0       0
1       1       0       0       0
2       0       0       0       0
3       0       0       1       0
4       0       0       0       0
```

Creating frovedis DataFrame from pandas Series:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# frovedis dataframe from a Series object.
ser = pd.Series([5, 6, np.NaN])

# creating a frovedis dataframe
fd_df = fdf.DataFrame(ser)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   0
0       5
1       6
2       NULL
```

Show which entries in corresonding frovedis DataFrame are NULL:

For example,

```
# isnull() demo on a frovedis dataframe converted from Series object
fd_df.isnull().show()
```

Output

```
index   0
0       0
1       0
2       1
```

Return Value
It returns a Frovedis DataFrame having boolean values (0, 1) corresponding to each of the Frovedis DataFrame value depending of whether it is a valid NaN (True i.e. 1) value or not (False i.e. 0).

54.1.3.19 19. DataFrame.join(right, on, how = 'inner', lsuffix = '_left', rsuffix = '_right', sort = False, join_type = 'bcast')

Parameters
right: It accepts a Frovedis DataFrame instance or a pandas DataFrame instance or a list of DataFrame instances as parameter. Index should be similar to one of the columns in this one. If a pandas Series instance is passed, its name attribute must be set, and that will be used as the column name in the resulting joined dataframe.
on: It accepts a string object or a list of strings as parameter. It is the column or index name(s) in the caller to join on the index in other, otherwise joins index-on-index. **This parameter must be provided. It can not be None.**
how: It accepts a string object as parameter that specifies how to handle the operation of the two dataframes. (Default: 'inner')
1. **'left'**: form union of calling dataframe's index (or column if 'on' is specified) with other dataframe's index, and sort it lexicographically.
2. **'inner'**: form intersection of calling dataframe's index (or column if 'on' is specified) with other dataframe's index, preserving the order of the calling's one.

lsuffix: It accepts a string object as parameter. It adds the suffix to left DataFrame's overlapping columns. (Default: '_left')
rsuffix: It accepts a string object as parameter. It adds the suffix to right DataFrame's overlapping columns. (Default: '_right')
sort: It accepts a boolean type value. It orders resultant dataframe lexicographically by the join key. If False, the order of the join key depends on the join type ('how' keyword). (Default: False)
join_type: It accepts a string type object as parameter. It represents the type of join to be used internally. It can be specified as 'bcast' for broadcast join or 'hash' for hash join. (Default: 'bcast')

Purpose
It joins columns of another dataframe.

It joins columns with other dataframe either on index or on a key column. Efficiently join multiple DataFrame instances by index at once by passing a list.

Note:- Parameters 'on', 'lsuffix', and 'rsuffix' are not supported when passing a list of DataFrame instances.

Creating two frovedis dataframes:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df1 = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'],
                       'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})

# creating a frovedis dataframe
fd_df1 = fdf.DataFrame(pd_df1)

# display frovedis dataframe
fd_df1.show()
```

```
# creating another pandas dataframe
pd_df2 = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
                       'B': ['B0', 'B1', 'B2']})

# creating another frovedis dataframe
fd_df2 = fdf.DataFrame(pd_df2)

# display other frovedis dataframe
fd_df2.show()
```

Output

```
index   key     A
0       K0      A0
1       K1      A1
2       K2      A2
3       K3      A3
4       K4      A4
5       K5      A5

index   key     B
0       K0      B0
1       K1      B1
2       K2      B2
```

Join dataframes using their indexes:

For example,

```
# join() demo with lsuffix, rsuffix and index parameters
fd_df1.join(fd_df2, 'index', lsuffix = '_caller', rsuffix = '_other').show()
```

Output

```
index   key_caller      A       key_other       B
0       K0              A0      K0              B0
1       K1              A1      K1              B1
2       K2              A2      K2              B2
```

Using the on parameter:

For example,

```
# join() demo using the 'key' columns,
# 'key' to be available in both fd_df1 and fd_df2
fd_df1.join(fd_df2, on = 'key').show()
```

Output

```
index   key     A       B
0       K0      A0      B0
1       K1      A1      B1
2       K2      A2      B2
```

Using the how parameter:

For example,

```
# join() demo with parameter how = 'left'
fd_df1.join(fd_df2, 'key', how = 'left').show()
```

Output,

```
index   key     A       B
0       K0      A0      B0
1       K1      A1      B1
2       K2      A2      B2
3       K3      A3      NULL
4       K4      A4      NULL
5       K5      A5      NULL
```

Return Value
It returns a new Frovedis DataFrame containing columns from both the DataFrame instances.

54.1.3.20 20. DataFrame.last_element(col_name, skipna = None)

Parameters ***col_name***: It accepts a string parameter as column name. ***skipna***: It accepts boolean as parameter. When set to True, it will exclude the missing values while fetching the last element from dataframe. (Default: None) When it is None (not specified explicitly), it excludes missing values while fetching the last element.

Purpose It fetches the last element from the given column (numeric/non-numeric) in a dataframe.

By default, it will exclude missing values while fetching last element from the given column.

It is present only in frovedis.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd',
                             'B.Tech', 'Phd', 'B.Tech'],
            'Score': [np.nan, 34, 35, 45, np.nan, 50, 52, np.nan]
           }

pd_df = pd.DataFrame(peopleDF)
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   Name    Age     City       Qualification   Score
0       Jai     27      Nagpur     B.Tech          NULL
1       Anuj    24      Kanpur     Phd             34
2       Jai     22      Allahabad  B.Tech          35
3       Princi  32      Kannuaj    Phd             45
4       Gaurav  33      Allahabad  Phd             NULL
5       Anuj    36      Kanpur     B.Tech          50
6       Princi  27      Kanpur     Phd             52
7       Abhi    32      Kanpur     B.Tech          NULL
```

Fetching the last element from 'Score' column (numeric column) :

For example,

```
fdf1.last_element(col_name='Score').show()
```

Output

```
52
```

Also, the above call can be expressed in two ways as follows,

For example,

```
fdf1.Score.last_element().show()
```

Output

```
52
```

For example,

```
fdf1["Score"].last_element().show()
```

Output

```
52
```

Fetching the last element from 'Name' column (non-numeric column) :

For example,

```
fdf1.last_element(col_name='Ename').show()
```

Output

```
Yuta
```

Using skipna=False in order to fetch last element (including Nan) from 'Score' column:

For example,

```
fdf1.last_element(col_name='Score', skipna=False).show()
```

Output

```
nan
```

Return Value It returns a scalar value.

54.1.3.21 21. DataFrame.merge(right, on = None, how = 'inner', left_on = None, right_on = None, left_index = False, right_index = False, sort = False, suffixes = ('_x', '_y'), copy = True, indicator = False, join_type = 'bcast')

Parameters
right: It accepts a Frovedis DataFrame instance or a pandas DataFrame instance or a list of Frovedis DataFrame instances as parameter. Index should be similar to one of the columns in this one. If a panads Series instance is passed, its name attribute must be set, and that will be used as the column name in the resulting joined dataframe.
on: It accepts a string object or a list of strings as parameter. It is the column or index level names to join on. These must be present in both dataframes. (Default: None)
When it is None and not merging on indexes then this defaults to the intersection of the columns in both dataframes.
how: It accepts a string object as parameter. It informs the type of merge operation on the two objects. (Default: 'inner')

1. **'left'**: form union of calling dataframe index (or column if 'on' is specified) with other dataframe index and sort it lexicographically.
2. **'inner'**: form intersection of calling dataframe index (or column if 'on' is specified) with other dataframe index, preserving the order of the calling dataframe.

left_on: It accepts a string object or a list of strings as parameter. It represents column names to join on in the left dataframe. It can also be an array or list of arrays of the length of the left dataframe. These arrays are treated as if they are columns. (Default: None)
right_on: It accepts a string object or a list of strings as parameter. It represents column names to join on in the right dataframe. It can also be an array or list of arrays of the length of the right dataframe. These arrays are treated as if they are columns. (Default: None)
left_index: It accepts a boolean value as parameter. It is used to specify whether to use the index from the left dataframe as the join key. **Either parameter 'left_on' or 'left_index' can be used, but not combination of both.** (Default: False)
right_index: It accepts a boolean value as parameter. It is used to specify whether to use the index from the right dataframe as the join key. **Either parameter 'right_on' or 'right_index' can be used, but not combination of both.** (Default: False)
sort: It accepts a boolean value. When this is explicitly set to True, it sorts the join keys lexicographically in the resultant dataframe. When it is False, the order of the join keys depends on the join type ('how' parameter). (Default: False)
suffixes: It accepts a list like (list or tuple) object of strings of length two as parameter. It indicates the suffix to be added to the overlapping column names in left and right respectively. Need to explicitly pass a value of None instead of a string to indicate that the column name from left or right should be left as-it is, with no suffix. At least one of the values must not be None. (Default: ('_x', '_y'))

Note:- During merging two DataFrames, the overlapping column names should be different. For example: suffixes = (False, False), then the overlapping columns would have the same name so merging operation will fail. Also when there is no overlapping column, then this parameter is ignored automatically.

copy: It is an unused parameter. (Default: True)
indicator: It is an unused parameter. (Default: False)
join_type: It accepts a string type object as parameter. It represents the type of join to be used internally. It can be specified as 'bcast' for broadcast join or 'hash' for hash join. (Default: 'bcast')

Purpose
It is a utility to merge dataframe objects with a database-style join.

The join is done on columns or indexes. If joining columns on columns, the DataFrame indexes will be ignored. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
                       'value': [1, 2, 3, 5]})

# creating another pandas dataframe
pd_df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
                       'value': [5, 6, 7, 8]})

# creating a frovedis dataframe
fd_df1 = fdf.DataFrame(pd_df1)
```

```
# display frovedis dataframe
fd_df1.show()

# creating another frovedis dataframe
fd_df2 = fdf.DataFrame(pd_df2)

# display other frovedis dataframe
fd_df2.show()
```

Output

```
index   lkey    value
0       foo     1
1       bar     2
2       baz     3
3       foo     5

index   rkey    value
0       foo     5
1       bar     6
2       baz     7
3       foo     8
```

For example,

```
# merge() demo with parameters left_on and right_on
fd_df1.merge(fd_df2, left_on = 'lkey', right_on = 'rkey').show()
```

Output

```
index   lkey    value_x rkey    value_y
0       foo     1       foo     8
1       foo     1       foo     5
2       bar     2       bar     6
3       baz     3       baz     7
4       foo     5       foo     8
5       foo     5       foo     5
```

For example,

```
# merge() demo with parameters left_on , right_on and suffixes
fd_df1.merge(fd_df2, left_on = 'lkey', right_on = 'rkey', suffixes = ('_left', '_right')).show()
```

Output

```
index   lkey    value_left      rkey    value_right
0       foo     1               foo     8
1       foo     1               foo     5
2       bar     2               bar     6
3       baz     3               baz     7
4       foo     5               foo     8
5       foo     5               foo     5
```

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
```

```
                        'left_value': [1, 2, 3, 5]})

# creating another pandas dataframe
pd_df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
                        'right_value': [5, 6, 7, 8]})

# creating a frovedis dataframe
fd_df1 = fdf.DataFrame(pd_df1)

# display a frovedis dataframe
fd_df1.show()

# creating another frovedis dataframe
fd_df2 = fdf.DataFrame(pd_df2)

# display other frovedis dataframe
fd_df2.show()
```

Output

```
index   lkey    left_value
0       foo     1
1       bar     2
2       baz     3
3       foo     5

index   rkey    right_value
0       foo     5
1       bar     6
2       baz     7
3       foo     8
```

For example,

```
# merge() demo with same suffixes on 'lkey' and 'rkey'
fd_df1.merge(fd_df2, left_on = 'lkey', right_on = 'rkey', suffixes = ('_test', '_test')).show()
```

Output

```
index   lkey    left_value      rkey    right_value
0       foo     1               foo     8
1       foo     1               foo     5
2       bar     2               bar     6
3       baz     3               baz     7
4       foo     5               foo     8
5       foo     5               foo     5
```

Note:- In above example, suffix is ignored as merging column labels are different. To have suffix, the column name must be same in both dataframes but then the suffixes must be different.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating two pandas datfarames
pd_df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
                        'value': [1, 2, 3, 5]})
```

```
pd_df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
                       'value': [5, 6, 7, 8]})

# creating a frovedis dataframe
fd_df1 = fdf.DataFrame(pd_df1)

# update index values of fd_df1
fd_df1.update_index(['a','b', 'c', 'd'], key = 'index', inplace = True)

# display frovedis dataframe
fd_df1.show()

# creating another frovedis dataframe
fd_df2 = fdf.DataFrame(pd_df2)

# update index values of fd_df2
fd_df2.update_index(['a','b', 'c', 'd'], key = 'index', inplace = True)

# display frovedis dataframe
fd_df2.show()
```

Output

```
index   lkey    value
a       foo     1
b       bar     2
c       baz     3
d       foo     5

index   rkey    value
a       foo     5
b       bar     6
c       baz     7
d       foo     8
```

For example,

```
# merge() demo with left_index = True and right_index = True
fd_df1.merge(fd_df2, left_index = True, right_index = True).show()
```

Output

```
index   lkey    value_x rkey    value_y
a       foo     1       foo     5
b       bar     2       bar     6
c       baz     3       baz     7
d       foo     5       foo     8
```

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating two pandas datframes
pd_df1 = pd.DataFrame({'a': ['foo', 'bar'], 'b': [1, 2]})
pd_df2 = pd.DataFrame({'a': ['foo', 'baz'], 'c': [3, 4]})

# creating two frovedis datframes
```

```
fd_df1 = fdf.DataFrame(pd_df1)
fd_df2 = fdf.DataFrame(pd_df2)

# display frovedis dataframes
fd_df1.show()
fd_df2.show()
```

Output

```
index   a       b
0       foo     1
1       bar     2

index   a       c
0       foo     3
1       baz     4
```

For example,

```
# merge() demo with how = 'inner' and 'on' parameter
fd_df1.merge(fd_df2, how = 'inner', on = 'a').show()
```

Output

```
index   a       b       c
0       foo     1       3
```

For example,

```
# merge using 'how' = left and 'on' parameter
fd_df1.merge(fd_df2, how = 'left', on = 'a').show()
```

Output,

```
index   a       b       c
0       foo     1       3
1       bar     2       NULL
```

Return Value
It returns a new Frovedis DataFrame instance with the merged entries of the two DataFrame instances.

54.1.3.22 22. DataFrame.rename(columns, inplace = False)

Parameters
columns: It accepts a dictionary object as parameter. It contains the key as the name of the labels to be renamed and values as the final names.
inplace.: It accepts a boolean object as parameter which specify whether to modify the original DataFrame instance or to return a copy. When it is set to True then the original DataFrame instance is modified. (Default: False)

Purpose
It is used to set the name of the columns.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
```

```
pd_df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   A       B
0       1       4
1       2       5
2       3       6
```

Rename columns using a mapping:

For example,

```
# rename columns 'A' and 'B' to 'a' and 'c' respectively
fd_df.rename(columns={"A": "a", "B": "c"}).show()
```

Output

```
index   a       c
0       1       4
1       2       5
2       3       6
```

Rename columns using a mapping and keep the dataframe in the same variable:

For example,

```
# rename() demo with inplace = True
fd_df.rename(columns={"A": "X", "B": "Y"}, inplace = True)
```

Output,

```
index   X       Y
0       1       4
1       2       5
2       3       6
```

Return Value

- It returns a new Frovedis DataFrame with the updated label name for the specified columns.
- It returns None when 'inplace' parameter is set to True.

54.1.3.23 23. DataFrame.reset_index(drop = False, inplace = False)

Parameters
drop: It accepts a boolean value as parameter. Do not try to insert index into dataframe columns. This resets the index to the default integer index. (Default: False)
inplace: It accepts a boolean values as parameter. When it is explicitly set to True, it modifies the original object directly instead of creating a copy of DataFrame object. (Default: False)

Purpose
It is used to reset the Index label of the DataFrame. A new Index label is inserted with default integer values from 0 to n-1 where n is the number of rows.

Note:- MultiIndex is not supported by Frovedis DataFrame.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
           }

# creating a pandas dataframe
pd_df = pd.DataFrame(peopleDF)

# updating the index values of pandas dataframe
pd_df.index = ['a', 'b', 'c', 'd', 'a']

# creating frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display the frovedid dataframe
fd_df.show()
```

Output

```
index   Ename   Age     Country isMale
a       Michael 29      USA     0
b       Andy    30      England 0
c       Tanaka  27      Japan   0
d       Raul    19      France  0
a       Yuta    31      Japan   1
```

Resetting the index and keeping the old index is as a column:

For example,

```
# reset_index() demo
fd_df.reset_index().show
```

Output

```
label_0 index_col       Ename   Age     Country isMale
0       a               Michael 29      USA     0
1       b               Andy    30      England 0
2       c               Tanaka  27      Japan   0
3       d               Raul    19      France  0
4       a               Yuta    31      Japan   1
```

When we reset the index, the old index is added as a column, and a new sequential index is used.

Using drop parameter:

For example,

```
# reset_index() demo with drop parameter
fd_df.reset_index(drop = True).show()
```

Output

```
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
4       Yuta    31      Japan   1
```

We can use the drop parameter to avoid the old index being added as a column.

Return Value

- It returns a new Frovedis DataFrame with the default sequence in index label.
- It returns None if 'inplace' parameter is set to True.

54.1.3.24 24. DataFrame.set_index(keys, drop = True, append = False, inplace = False, verify_integrity = False)

Parameters
keys: It accepts a string object as parameter. This parameter can be a single column key.
drop: It accepts a boolean value as parameter. When it is set to True, it will remove the column which is selected as new index. **Currently, Frovedis does not support drop = False.** (Default: True)
append: It accepts a boolean value as parameter. It will decide whether to append columns to existing index. **Currently, Frovedis does not support append = True.** (Default: False)
inplace: It accepts a boolean values as parameter which is when explicitly set to True, it modifies the original object directly instead of creating a copy of DataFrame instance. (Default: False)
verify_integrity: It accepts a boolean value as parameter. When it is set to True, it checks the new index for duplicates. Performance of this method will be better when it is set to False. (Default: False)

Purpose
It is used to set the frovedis dataframe index using existing columns. The index will replace the existing index.

Note:- Frovedis DataFrame does not support Multi Index.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df = pd.DataFrame({'month': [1, 4, 1, 10],
                      'year': [2012, 2014, 2013, 2014],
                      'sale': [55, 40, 84, 31]})

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   month   year    sale
0       1       2012    55
```

```
1       4       2014    40
2       1       2013    84
3       10      2014    31
```

Set the 'month' column to be the new index:

For example,

```
fd_df.set_index('month').show()
```

Output

```
month   year    sale
1       2012    55
4       2014    40
1       2013    84
10      2014    31
```

Set the 'month' column to be the new index. Also, it checks for duplicates before setting:

For example,

```
# set_index() demo with verify_integrity = True
fd_df.set_index('month', verify_integrity = True).show()
```

Output

```
month   year    sale
1       2012    55
4       2014    40
7       2013    84
10      2014    31
```

Note:- In above example, the column which is being selected as index must have unique values.

Return Value

- It returns a new Frovedis DataFrame where the Index column label is replace with specified column label.
- It returns None when 'inplace' parameter is set to True.

54.1.3.25 25. DataFrame.show()

Purpose
This method is used to display the Frovedis DataFrame on the console. It can display full dataframe or some selected columns of the DataFrame (single or multi-column).

It can be used either with any method which returns a Frovedis DataFrame instance type compatible with string type.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating the dataframe
peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
```

```
            'isMale': [False, False, False, False, True]
          }

pdf = pd.DataFrame(peopleDF)

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pdf)

# display frovedis dataframe
print("Displaying complete frovedis dataframe")
fd_df.show()
```

Output

```
Displaying complete frovedis dataframe
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
4       Yuta    31      Japan   1
```

Display dataframe by selecting single column:

For example,

```
print("Displaying frovedis dataframe with just Ename column")
fd_df["Ename"].show()         # single column
```

Output

```
Displaying frovedis dataframe with just Ename column
index   Ename
0       Michael
1       Andy
2       Tanaka
3       Raul
4       Yuta
```

Displaying multiple columns:

For example,

```
print("Displaying frovedis dataframe with Ename and Age columns")
fd_df[["Ename","Age"]].show() # multiple column
```

Output

```
Displaying frovedis dataframe with Ename and Age columns
index   Ename   Age
0       Michael 29
1       Andy    30
2       Tanaka  27
3       Raul    19
4       Yuta    31
```

Displaying frovedis dataframe using condition based indexing operator:

For example,

```
print("Displaying frovedis dataframe using condition based slicing operator")
```

```
fd_df[fd_df.Age > 19].show()
```

Output

```
Displaying frovedis dataframe using condition based slicing operator
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
4       Yuta    31      Japan   1
```

Displaying frovedis dataframe using chaining of methods:

For example,

```
print("Displaying frovedis dataframe using chaining of methods")
fd_df[fd_df.Country.str.contains("a")].show()
```

Output

```
Displaying frovedis dataframe using chaining of methods
index   Ename   Age     Country isMale
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
4       Yuta    31      Japan   1
```

Return Value
It return nothing.

54.1.3.26 26. DataFrame.tail(n = 5)

Parameters
n: It accepts an integer as parameter. It represents the number of rows which is to be selected. (Default: 5)

Purpose
This utility is used to return the last **n** rows from the DataFrame.

It is useful for quickly verifying data, for example, after sorting or appending rows.

For negative values of n, this function returns all rows except the first n rows, equivalent to **fdf[n:]**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',
                      'monkey', 'parrot', 'shark', 'whale', 'zebra']})

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

print("Viewing all entries of dataframe")
fd_df.show()
```

Output

```
Viewing all entries of dataframe
index   animal
0       alligator
1       bee
2       falcon
3       lion
4       monkey
5       parrot
6       shark
7       whale
8       zebra
```

Viewing the last 5 lines (by default n=5):

For example,

```
# tail() demo
print("Viewing the last 5 lines")
fd_df.tail().show()
```

Output

```
Viewing the last 5 lines
index   animal
4       monkey
5       parrot
6       shark
7       whale
8       zebra
```

Viewing the last 3 lines:

For example,

```
# tail() demo with n = 3
print("Viewing the last 3 lines")
fd_df.tail(3).show()
```

Output

```
Viewing the last 3 lines
index   animal
6       shark
7       whale
8       zebra
```

For negative values of n:

For example,

```
# tail() demo with n = -2
print("For negative values of n")
fd_df.tail(-2).show()
```

Output

```
For negative values of n
index   animal
2       falcon
3       lion
4       monkey
```

```
5       parrot
6       shark
7       whale
8       zebra
```

Return Value
It returns a new Frovedis DataFrame instance with last **n** rows.

54.2 SEE ALSO

- **DataFrame - Introduction**
- **DataFrame - Indexing Operations**
- **DataFrame - Conversion Functions**
- **DataFrame - Sorting Functions**
- **DataFrame - Math Functions**
- **DataFrame - Aggregate Functions**

Chapter 55

DataFrame Conversion Functions

55.1 NAME

DataFrame Conversion Functions - this manual contains all functions related to conversion of datatypes with respect to frovedis dataframe.

55.1.1 DESCRIPTION

In frovedis during data pre-processing and manipulation, there might be the need to change the data type of the variable to a particular type for better cleaning and understanding of the data.

For this inter-conversion within the variables, frovedis dataframe offer various conversion functions like asDF(), to_dict(), to_numpy(),etc.

55.1.2 Public Member Functions

```
1. asDF(df)
2. to_dict(orient = "dict", into = dict)
3. to_numpy(dtype = None, copy = False, na_value = None)
4. to_pandas()
5. to_frovedis_rowmajor_matrix(t_cols, dtype = np.float32)
6. to_frovedis_colmajor_matrix(t_cols, dtype = np.float32)
7. to_frovedis_crs_matrix(t_cols, cat_cols, dtype = np.float32, need_info = False)
8. to_frovedis_crs_matrix_using_info(info, dtype = np.float32)
```

55.1.3 Detailed Description

55.1.3.1 1. DataFrame.asDF(df)

Parameters
df: It takes either a Frovedis DataFrame or a Pandas DataFrame or a Series instance.

Purpose
It creates a new Frovedis DataFrame after suitable conversion.

Using asDF() to create frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
           }
# convert to pandas dataframe
pd_df = pd.DataFrame(peopleDF)

print("display the pandas dataframe type")
print(type(pd_df))

# convert pandas DataFrame to frovedis DataFrame object
print("display type after conversion to frovedis dataframe")
print(type(fdf.DataFrame.asDF(pd_df)))
```

Output

```
display the pandas dataframe type
<class 'pandas.core.frame.DataFrame'>

display type after conversion to frovedis dataframe
<class 'frovedis.dataframe.df.DataFrame'>
```

Using asDF() to create frovedis DataFrame from pandas Series:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# create a Series object
sdf1 = pd.Series([1, 2], dtype='int32')
print("display series type")
print(type(sdf1))

# convert a Series object to Frovedis DataFrame object
print("display type after conversion to frovedis dataframe")
print(type(fdf.DataFrame.asDF(sdf1)))
```

Output

```
display series type
<class 'pandas.core.series.Series'>

display type after conversion to frovedis dataframe
<class 'frovedis.dataframe.df.DataFrame'>
```

Return Value
It returns a Frovedis DataFrame instance after suitable conversion

55.1.3.2 2. DataFrame.to_dict(orient = "dict", into = dict)

Parameters
orient: It accepts a string object as parameter. It is used to determine the type of the values of the dictionary. (Default: 'dict')
1. *'dict'* : dict like {column -> {index -> value}}
2. *'list'* : dict like {column -> [values]}

into: This parameter is used for mapping in the return value. Currently it only supports OrderedDict as return type. (Default: dict)

Purpose
It is used to convert the Frovedis DataFrame to a dictionary.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a pandas dataframe
pd_df = pd.DataFrame({'col1': [1, 2],
                      'col2': [0.5, 0.75]},
                      index=['row1', 'row2'])

# create a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# to display frovedis dataframe
fd_df.show()
```

Output

```
index   col1    col2
row1    1       0.5
row2    2       0.75
```

Converting frovedis DataFrame to dictionary:

For example,

```
print(fd_df.to_dict())
```

Output

```
OrderedDict([('col1', {'row1': 1, 'row2': 2}), ('col2', {'row1': 0.5, 'row2': 0.75})])
```

Converting frovedis DataFrame to dictionary and using orient parameter:

For example,

```
# to_dict() demo with orient = 'list'
print(fd_df.to_dict(orient = 'list'))
```

Output

```
OrderedDict([('index', array(['row1', 'row2'], dtype='<U4')),
             ('col1', array([1, 2])), ('col2', array([0.5 , 0.75]))])
```

Return Value
It returns a dictionary representing the Frovedis DataFrame instance. The resulting transformation depends on the 'orient' parameter.

55.1.3.3 3. DataFrame.to_numpy(dtype = None, copy = False, na_value = None)

Parameters
dtype : It accepts the dtype parameter which decides the datatype of numpy ndarray. (Default: None)
When it is None (not specified explicitly), it will be set as double (float64).
copy : This is an unused parameter. (Default: False)
na_value : This is an unused parameter. (Default: None)

Purpose
This method is used to convert a frovedis dataframe into a numpy array.

The parameters: "copy" and "na_value" are simply kept in to make the interface uniform to the pandas DataFrame.to_numpy().
This is not used in the frovedis implementation.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a python dict
peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
           }

# a pandas dataframe object from dict
pd_df = pd.DataFrame(peopleDF)

# to create a frovedis dataframe object
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()

# below will display a Frovedis dataframe type
print(type(fd_df))
```

Output

```
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
4       Yuta    31      Japan   1

<class 'frovedis.dataframe.df.DataFrame'>
```

Convert frovedis DataFrame into numpy array:

For example,

```
# to_numpy() demo to convert a frovedis dataframe into numpy array
# below will display a numpy array type
```

```
print(fd_df.to_numpy())

# below will display a numpy array type
print(type(fd_df.to_numpy()))
```

Output

```
[['Michael' '29' 'USA' '0']
 ['Andy' '30' 'England' '0']
 ['Tanaka' '27' 'Japan' '0']
 ['Raul' '19' 'France' '0']
 ['Yuta' '31' 'Japan' '1']]

<<class 'numpy.ndarray'>
```

Return Value
It returns a numpy ndarray that represents a Frovedis DataFrame instance. It has shape **(nRows , nCols)**.

55.1.3.4 4. DataFrame.to_pandas()

Purpose
It converts a Frovedis DataFrame instance into a pandas DataFrame instance.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a python dictionary
peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
            }

# creating a pandas dataframe from python dict
pd_df = pd.DataFrame(peopleDF)

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# below will display a Frovedis dataframe
fd_df.show()

# below will display a Frovedis dataframe type
print(type(fd_df))
```

Output

```
index   Ename   Age     Country isMale
0       Michael 29      USA     0
1       Andy    30      England 0
2       Tanaka  27      Japan   0
3       Raul    19      France  0
```

```
4       Yuta    31      Japan   1

<class 'frovedis.dataframe.df.DataFrame'>
```

Convert frovedis DataFrame to pandas DataFrame:

For example,

```
# to_pandas() demo to convert frovedis dataframe to pandas dataframe
print(fd_df.to_pandas())

# below will display a pandas dataframe type after conversion
print(type(fd_df.to_pandas()))
```

Output

```
index  Ename  Age  Country  isMale
0      Michael   29      USA   False
1         Andy   30  England   False
2       Tanaka   27    Japan   False
3         Raul   19   France   False
4         Yuta   31    Japan    True

<class 'pandas.core.frame.DataFrame'>
```

Return Value
It returns a pandas DataFrame instance after suitable conversion.

55.1.3.5 5. DataFrame.to_frovedis_rowmajor_matrix(t_cols, dtype = np.float32)

Parameters
t_cols: It accepts a list of string type argument where each of the member of the list is the name of the column labels.
dtype: It accepts a dtype type argument which is the type of the resultant values. Currently, only float (float32) and double (float64) types are supported. (Default: np.float32)

Purpose
This method is used to convert a Frovedis DataFrame instance into FrovedisRowmajorMatrix instance.

Loading input file data into frovedis DataFrame:

For example,

```
import frovedis.dataframe as fdf

# read_csv demo to get values
df = fdf.read_csv("./input/numbers.csv", names = ['one', 'two', 'three', 'four'])

# display frovedis dataframe
df.show()
```

Output

```
index   one     two     three   four
0       10      10.23   F       0
1       12      12.2    F       0
2       13      34.8999 D       1
3       15      100.12  A       2
```

Convert frovedis DataFrame into FrovedisRowmajorMatrix:

For example,

```
# to convert frovedis dataframe into frovedis rowmajor matrix
row_mat = df.to_frovedis_rowmajor_matrix(['one', 'two'], dtype = np.float64)

# display row_mat (row major matrix)
row_mat.debug_print()
```

Output

```
matrix:
num_row = 4, num_col = 2
node 0
node = 0, local_num_row = 4, local_num_col = 2, val = 10 10.23 12 12.2 13 34.9 15 100.12
```

Return Value
It returns a FrovedisRowmajorMatrix instance after suitable conversion.

55.1.3.6 6. DataFrame.to_frovedis_colmajor_matrix(t_cols, dtype = np.float32)

Parameters
t_cols: It accepts a list of string type argument where each of the member of the list is the name of the column labels.
dtype: It accepts a dtype type argument which is the type of the resultant values. Currently only float (float32) and double (float64) types are supported. (Default: np.float32)

Purpose
This method converts a Frovedis DataFrame instance to FrovedisColmajorMatrix instance.

Loading input file data into frovedis DataFrame:

For example,

```
import frovedis.dataframe as fdf

# read_csv demo
df = fdf.read_csv("./input/numbers.csv", names=['one', 'two', 'three', 'four'])

# display frovedis dataframe
df.show()
```

Output

```
index   one     two     three   four
0       10      10.23   F       0
1       12      12.2    F       0
2       13      34.8999 D       1
3       15      100.12  A       2
```

Convert frovedis DataFrame into FrovedisColmajorMatrix:

For example,

```
# to convert frovedis dataframe into frovedis colmajor matrix
col_mat = df.to_frovedis_colmajor_matrix(['one', 'two']) # default dtype = np.float32

# display col_mat
col_mat.debug_print()
```

Output

```
matrix:
num_row = 4, num_col = 2
node 0
node = 0, local_num_row = 4, local_num_col = 2, val = 10 12 13 15 10.23 12.2 34.9 100.12
```

Return Value
It returns a FrovedisColmajorMatrix instance after converting the original Frovedis DataFrame instance.

55.1.3.7 7. DataFrame.to_frovedis_crs_matrix(t_cols, cat_cols, dtype = np.float32, need_info = False)

Parameters
t_cols: It accepts a list of string type argument where each of the member of the list is the name of the column labels.
cat_cols: It accepts a list of strings as parameter where strings are the column names. It stands for categorical columns. It represents how the repetitive values are distributed in the specified column. It adds number of columns corresponding to the number of distinct values in the specified column to the matrix. And values corresponding to the value of the specified column is set to 1 and remaining values are set to 0.
dtype: It accepts a dataype as paramter which is the type of the resultant values. Currently only float (float32) and double (float64) types are supported. (Default: np.float32)
need_info: It accepts a boolean value as parameter. When this is explicitly set to True, it returns an additional value of type 'df_to_sparse_info'. (Default: False)

Purpose
It converts a Frovedis DataFrame instance to FrovedisCRSMatrix instance.

Loading input file data into frovedis DataFrame:

For example,

```
import frovedis.dataframe as fdf
import numpy as np

# read_csv demo
df = fdf.read_csv("./input/numbers.csv", names = ['one', 'two', 'three', 'four'])

# display frovedis dataframe
df.show()
```

Output

```
index   one     two     three   four
0       10      10.23   F       0
1       12      12.2    F       0
2       13      34.8999 D       1
3       15      100.12  A       2
```

Convert frovedis DataFrame to FrovedisCRSMatrix:

For example,

```
# to convert frovedis datafrme to frovedis crs matrix
crs_mat,info = df.to_frovedis_crs_matrix(['one', 'two', 'four'],
                                         ['three'],
                                         need_info = True) # default dtype = np.float32

# display crs_matrix
```

```
crs_mat.debug_print()

# converting crs_matrix into display it in better form
mat_t = crs_mat.to_scipy_matrix().todense()
print(mat_t)
```

Output

```
Active Elements:  12
matrix:
num_row = 4, num_col = 5
node 0
local_num_row = 4, local_num_col = 5
val : 10 10.23 1 12 12.2 1 13 34.9 1 15 100.12 1
idx : 0 1 2 0 1 2 0 1 3 0 1 4
off : 0 3 6 9 12

[[ 10.    10.23   0.     0.     1.  ]
 [ 12.    12.2    0.     0.     1.  ]
 [ 13.    34.9    0.     1.     0.  ]
 [ 15.   100.12   1.     0.     0.  ]]
```

Return Value
It returns a FrovedisCRSMatrix instance after converting the original Frovedis DataFrame instance.

55.1.3.8 8. DataFrame.to_frovedis_crs_matrix_using_info(info, dtype = np.float32)

Parameters
info: It accepts an instance of 'df_to_sparse_info' type.
dtype: It accepts a dtype type argument which is the type of the resultant values. Currently only float (float32) and double (float64) types are supported. (Default: np.float32)

Purpose
It converts a Frovedis DataFrame instance to FrovedisCRSMatrix instance and provided an info object of 'df_to_sparse_info' class.

Loading input file data into frovedis DataFrame:

For example,

```
import frovedis.dataframe as fdf

# read_csv demo
df = fdf.read_csv("./input/numbers.csv", names=['one', 'two', 'three', 'four'])

# display frovedis dataframe
df.show()
```

Output

```
index   one     two     three   four
0       10      10.23   F       0
1       12      12.2    F       0
2       13      34.8999 D       1
3       15      100.12  A       2
```

Convert frovedis DataFrame to FrovedisCRSMatrix:

For example,

```
# to use to_frovedis_crs_matrix_using_info
crs_mat,info = df.to_frovedis_crs_matrix(['one', 'two', 'four'],
                                         ['four'],
                                         need_info=True) # default dtype = np.float32
print(type(info))

crs_mat2 = df.to_frovedis_crs_matrix_using_info(info)
print(type(crs_mat2))

# display crs_matrix
crs_mat2.debug_print()
```

Output

```
<class 'frovedis.dataframe.info.df_to_sparse_info'>

<class 'frovedis.matrix.crs.FrovedisCRSMatrix'>

Active Elements:  12
matrix:
num_row = 4, num_col = 5
node 0
local_num_row = 4, local_num_col = 5
val : 10 10.23 1 12 12.2 1 13 34.9 1 15 100.12 1
idx : 0 1 2 0 1 2 0 1 3 0 1 4
off : 0 3 6 9 12
```

Return Value
It returns a FrovedisCRSMatrix instance after converting Frovedis DataFrame instance.

55.2 SEE ALSO

- **DataFrame - Introduction**
- **DataFrame - Indexing Operations**
- **DataFrame - Generic Functions**
- **DataFrame - Sorting Functions**
- **DataFrame - Math Functions**
- **DataFrame - Aggregate Functions**

Chapter 56

DataFrame Sorting Functions

56.1 NAME

DataFrame Sorting Functions - this manual contains list of all functions related to sorting operations performed on frovedis dataframe.

56.1.1 DESCRIPTION

Frovedis datafrme provides various sorting functions such as sort_index(), sort_values(), etc.

These can be used to sort data along rows or columns of frovedis dataframe, either in ascending or descending order.

Currently, sorting opertaions will be performed on a copy of frovedis dataframe. Inplace sorting is not supported yet.

56.1.2 Public Member Functions

```
1. nlargest(n, columns, keep = 'first')
2. nsmallest(n, columns, keep = 'first')
3. sort(columns = None, axis = 0, ascending = True, inplace = False, kind = 'radixsort',
        na_position = 'last', **kwargs)
4. sort_index(axis = 0, ascending = True, inplace = False, kind = 'quicksort', na_position = 'last')
5. sort_values(by, axis = 0, ascending = True, inplace = False, kind = 'radixsort', na_position = 'last
```

56.1.3 Detailed Description

56.1.3.1 1. DataFrame.nlargest(n, columns, keep = 'first')

Parameters
n: It accepts an integer parameter which represents the number of rows to return.
columns: It accepts a string type or list of strings type parameter which represents the column label(s) to order by.
keep: It accepts a string object parameter. This parameter is used when there are duplicate values. (Default: 'first')
- **'first'** : it prioritizes the first occurrence(s).

- **'last'** : it prioritizes the last occurrence(s).
- **'all'** : it is used to not drop any duplicates, even when it means selecting more than **n** items.

Purpose
It is used to return the **top n rows** ordered by the specified columns in descending order. The columns that are not specified are returned as well, but not used for ordering.

This method is equivalent to DataFrame.sort_values(columns, ascending = False).head(n), but it is more efficient.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

pd_df = pd.DataFrame({'population': [59000000, 65000000, 434000, 434000,
                                     434000, 337000, 11300, 11300, 11300],
                      'GDP': [1937894, 2583560 , 12011, 4520, 12128,
                              17036, 182, 38, 311],
                      'alpha-2': ["IT", "FR", "MT", "MV", "BN",
                                  "IS", "NR", "TV", "AI"]},
                      index=["Italy", "France", "Malta",
                             "Maldives", "Brunei", "Iceland",
                             "Nauru", "Tuvalu", "Anguilla"])
# create a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# to display frovedis dataframe
fd_df.show()
```

Output

```
index       population      GDP       alpha-2
Italy       59000000        1937894   IT
France      65000000        2583560   FR
Malta       434000          12011     MT
Maldives    434000          4520      MV
Brunei      434000          12128     BN
Iceland     337000          17036     IS
Nauru       11300           182       NR
Tuvalu      11300           38        TV
Anguilla    11300           311       AI
```

In order to use nlargest() to select the 3 rows having the largest values in 'population' column:

For example,

```
fd_df.nlargest(3, 'population').show()
```

Output

```
index      population      GDP       alpha-2
France     65000000        2583560   FR
Italy      59000000        1937894   IT
Malta      434000          12011     MT
```

When using keep='last', last ocurrances are prioritized. Ties are resolved in reverse order:

For example,

```
# nlargest() demo with keep = 'last'
fd_df.nlargest(3, 'population', keep='last').show()
```

Output

```
index       population     GDP       alpha-2
France      65000000       2583560   FR
Italy       59000000       1937894   IT
Brunei      434000         12128     BN
```

When using keep='all', all duplicate items are maintained (not dropped):

For example,

```
# nlargest() demo with keep = 'all'
fdf.nlargest(3, 'population', keep='all').show()
```

Output

```
index       population     GDP       alpha-2
France      65000000       2583560   FR
Italy       59000000       1937894   IT
Malta       434000         12011     MT
Maldives    434000         4520      MV
Brunei      434000         12128     BN
```

To order by the largest values in 'population' and then 'GDP' column, multiple columns may be specified:

For example,

```
# nlargest() demo with use of list of columns
fd_df.nlargest(3, ['population', 'GDP']).show()
```

Output

```
index       population     GDP       alpha-2
France      65000000       2583560   FR
Italy       59000000       1937894   IT
Brunei      434000         12128     BN
```

Return Value
It returns a Frovedis DataFrame instance with **n** rows ordered by the specified columns in descending order.

56.1.3.2 2. DataFrame.nsmallest(n, columns, keep = 'first')

Parameters
n: It accepts an integer type argument that represents the number of rows to return.
columns: It accepts a string type or list of strings type parameter which represents the column label(s) to order by.
keep: It accepts a string object parameter. This parameter is used when there are duplicate values. (Default: 'first')
1. **'first'** : it prioritizes the first occurrence(s).
2. **'last'** : it prioritizes the last last occurrence(s).
3. **'all'** : it is used to not drop any duplicates, even it means selecting more than **n** items.

Purpose
It is used to return the **top n rows** ordered by the specified columns in ascending order.

It returns the first n rows with the smallest values in columns, in ascending order. The columns that are not specified are returned as well, but not used for ordering.

This method is equivalent to DataFrame.sort_values(columns, ascending=True).head(n), but it is more efficient.

Creating frovedis DataFrame from pandas DataFrame:

For example

```
import pandas as pd
import frovedis.dataframe as fdf

pd_df = pd.DataFrame({'population': [59000000, 65000000, 434000,
                                     434000, 434000, 337000, 11300,
                                     11300, 11300],
                      'GDP': [1937894, 2583560 , 12011, 4520, 12128,
                              17036, 182, 38, 311],
                      'alpha-2': ["IT", "FR", "MT", "MV", "BN",
                                  "IS", "NR", "TV", "AI"]},
                      index=["Italy", "France", "Malta",
                             "Maldives", "Brunei", "Iceland",
                             "Nauru", "Tuvalu", "Anguilla"])

# create a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# to display frovedis dataframe
fd_df.show()
```

Output

```
index       population     GDP       alpha-2
Italy       59000000       1937894   IT
France      65000000       2583560   FR
Malta       434000         12011     MT
Maldives    434000         4520      MV
Brunei      434000         12128     BN
Iceland     337000         17036     IS
Nauru       11300          182       NR
Tuvalu      11300          38        TV
Anguilla    11300          311       AI
```

In order to use nlargest() to select the 3 rows having the largest values in 'population' column:

For example,

```
fd_df.nsmallest(3, 'population').show()
```

Output

```
index       population     GDP      alpha-2
Nauru       11300          182      NR
Tuvalu      11300          38       TV
Anguilla    11300          311      AI
```

When using keep='last', last ocurrances are prioritized. Ties are resolved in reverse order:

For example,

```
# nsmallest() demo with keep = 'last' parameter
fd_df.nsmallest(3, 'population', keep='last').show()
```

Output

```
index       population     GDP      alpha-2
Anguilla    11300          311      AI
Tuvalu      11300          38       TV
Nauru       11300          182      NR
```

When using keep='all', all duplicate items are maintained (not dropped):

For example

```
# example to use nsmallest with keep = 'all' parameter
fd_df.nsmallest(3, 'population', keep='all').show()
```

Output

```
index       population     GDP      alpha-2
Nauru       11300          182      NR
Tuvalu      11300          38       TV
Anguilla    11300          311      AI
```

To order by the largest values in 'population' and then 'GDP' column, multiple columns may be specified:

For example,

```
# nsmallest() demo with using a list of columns names
fd_df.nsmallest(3, ['population', 'GDP'])
```

Output

```
index       population     GDP      alpha-2
Tuvalu      11300          38       TV
Nauru       11300          182      NR
Anguilla    11300          311      AI
```

Return Value
It returns a Frovedis DataFrame object with **n** rows ordered by the specified columns in ascending order.

56.1.3.3 3. DataFrame.sort(columns = None, axis = 0, ascending = True, inplace = False, kind = 'radixsort', na_position = 'last', **kwargs)

Parameters
columns: It accepts the name or list of names on which sorting will be applied. (Default: None)
If axis is 0 or 'index' then this parameter should contain index levels and/or column labels. Currently axis = 0 or 'index' is only supported in Frovedis DataFrame.
When it is None (not specified explicitly), it will not perform sorting and it will raise an exception.
axis: It accepts an integer or a string object as parameter.To perform sorting along rows or columns, it is selected by this parameter. (Default: 0)
If axis is 0 or 'index', sorting is performed along the row. Currently axis = 0 or 'index' is only supported in Frovedis DataFrame.
ascending: It accepts a boolean value or a list of booleans as parameter. The order of sorting is decided by this parameter. Need to specify a list of booleans for multiple sort orders. If this is a list of booleans, then it must match the length of the 'columns' parameter. By default, the order of sorting will be ascending and to change the order to descending, explicitly pass it as False. (Default: True)
inplace: It accepts a boolean value as parameter. When it is explicitly set to True, it modifies the original object directly instead of creating a copy of DataFrame object. **Currently, 'inplace' = True is not supported by this Frovedis DataFrame method.** (Default: False)
kind: It accepts a string object as parameter to select the type of sorting algorithm. **Currently, Frovedis supports only 'radixsort' and other values for 'kind' parameter will be ignored internally with a warning.** (Default: 'radixsort')

na_position: It accepts a string object as parameter. It decides the position of NaNs after sorting. When it is set to 'last', it puts NaNs at the end. **Currently, Frovedis only supports 'na_position' = 'last'**. (Default: 'last')
****kwargs***: It accepts a dictionary object as parameter. It is used to pass all the other parameters at once in the form of a dictionary object. Currently this is not supported.

Purpose
It is used to sort the values in the specified column(s) along axis = 0 or axis = 'index' in the Frovedis DataFrame.

This method is present only in frovedis. It internally uses sort_values().

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

pd_df = pd.DataFrame({
                      'col1': ['A', 'A', 'B', np.nan, 'D', 'C'],
                      'col2': [2, 1, 9, 8, 7, 4],
                      'col3': [0, 1, 9, 4, 2, 3],
                      'col4': ['a', 'B', 'c', 'D', 'e', 'F'] })

# create a frovedis dataframe from pandas dataframe
fd_df = fdf.DataFrame(pd_df)

# display frovedis dataframe
fd_df.show()
```

Output

```
index   col1    col2    col3    col4
0       A       2       0       a
1       A       1       1       B
2       B       9       9       c
3       NULL    8       4       D
4       D       7       2       e
5       C       4       3       F
```

Sort by 'col1' on frovedis dataframe:

For example,

```
# Sort dataframe by 'col1'
fd_df.sort('col1').show()
```

Output

```
index   col1    col2    col3    col4
0       A       2       0       a
1       A       1       1       B
2       B       9       9       c
5       C       4       3       F
4       D       7       2       e
3       NULL    8       4       D
```

Sorting frovedis dataframe in ascending order (by default) by using multiple columns:

For example,

```
# Sort dataframe by multiple columns
fd_df.sort(['col1', 'col2']).show()
```

Output

```
index   col1    col2    col3    col4
1       A       1       1       B
0       A       2       0       a
2       B       9       9       c
5       C       4       3       F
4       D       7       2       e
3       NULL    8       4       D
```

NOTE: In the above example, in case of multiple columns sorting, **'col1'** will be sorted first and **'col2'** will only be considered for sorting in case of duplicate entries present in **'col1'**.

Sorting frovedis dataframe in descending orde by using ascending=False:

For example,

```
# Sort datafrme in descending order
fd_df.sort('col1', ascending = False).show()
```

Output

```
index   col1    col2    col3    col4
3       NULL    8       4       D
4       D       7       2       e
5       C       4       3       F
2       B       9       9       c
0       A       2       0       a
1       A       1       1       B
```

Return Value
It returns a new Frovedis DataFrame with sorted values.

56.1.3.4 4. DataFrame.sort_index(axis = 0, ascending = True, inplace = False, kind = 'quicksort', na_position = 'last')

Parameters
axis: It accepts an interger or a string object as parameter. It is the axis along which the sorting will be performed. (Default: 0)
When axis = 0 or axis = 'index', operation will be performed on rows. **Currently only axis = 0 or axis = 'index' is supported**.
ascending: It accepts a boolean value as parameter. This parameter decides the order to sort the data. (Default: True)
When this parameter is explicitly passed as False, it sorts the data into descending order.
inplace: It accepts a boolean value as paramter. To modify the original DataFrame object, argument is explicitly passed as True. Otherwise operation is performed on a copy of Frovedis DataFrame object. **Currently 'inplace' = True is not supported by this method.** (Default: False)
kind: It accepts a string object as parameter. This parameter is used to select the sorting algorithms. (Default: 'quicksort')
na_position: It accepts a string object as parameter. It is used to decide where to puts NaNs i.e at the beginning or at the end. (Default: 'last')
When na_position = 'last', it puts NaNs at the end. **Currently, na_position = 'last' is only supported for this method.**

Purpose
It is used to sort Frovedis DataFrame according to index values. It creates a new sorted DataFrame by the specified label.

Currently it only supports 'radixsort' and other values for 'kind' parameter are ignored internally along with a warning. Also, this method does not support MultiIndex yet.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df = pd.DataFrame([1, 2, 3, 4, 5],
                     index=[100, 29, 234, 1, 150],
                     columns=['A'])

# creating frovedis dataframe from pandas dataframe
fd_df = fdf.DataFrame(pd_df)

# to display frovedis dataframe
fd_df.show()
```

Output

```
index   A
100     1
29      2
234     3
1       4
150     5
```

Sort values in dataframe, by default in ascending order (ascending=True):

For example,

```
# to display sorted dataframe by index
fd_df.sort_index().show()
```

Output

```
index   A
1       4
29      2
100     1
150     5
234     3
```

In order to sort in descending order, use ascending=False:

For example,

```
fd_df.sort_index(ascending=False)
```

Output

```
index   A
234     3
150     5
100     1
```

```
29      2
1       4
```

Return Value
It returns a new Frovedis DataFrame instance sorted by the labels.

56.1.3.5 5. DataFrame.sort_values(by, axis = 0, ascending = True, inplace = False, kind = 'radixsort', na_position = 'last')

Parameters
by: It accepts the name or list of names on which sorting will be applied.
If axis is 0 or 'index', then 'by' should contain index levels and/or column labels. **Currently axis = 1 or axis = 'columns' is not supported.**
axis: It accepts an integer or a string object as parameter. To perform sorting along rows or columns, it is selected by this parameter. (Default: 0)
If axis is 0 or 'index', sorting is performed along the row. **Currently axis = 1 or axis = 'columns' is not supported.**
ascending: It accepts a boolean value or a list of boolean values as parameter. The order of sorting is decided by this parameter. Need to specify a list of booleans for multiple sorting orders. If this is a list of booleans, the length of the list must match the length of the 'by' parameter list. By default, the order of sorting will be ascending and to change the order to descending, explicitly pass it as False. (Default: True)
inplace: It accepts a boolean value as parameter. To modify the original DataFrame object, argument is explicitly passed as True. Otherwise operation is performed on a copy of Frovedis DataFrame object. **Currently 'inplace' = True is not supported by this method.** (Default: False)
kind: It accepts a string object as parameter. The type of sorting algorithm is decided from this parameter. **Currently it only supports 'radixsort' and other values for 'kind' parameter are ignored internally along with a warning.** (Default: 'radixsort')
na_position: It accepts a string object as parameter. It is used to decide where to puts NaNs i.e at the beginning or at the end. (Default: 'last')
When na_position = 'last', it puts NaNs at the end. **Currently, it only supports na_position = 'last'.**

Purpose
To sort the DataFrame by the values along axis = 0 or 'index'.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# creating a pandas dataframe
pd_df = pd.DataFrame({
                      'col1': ['A', 'A', 'B', np.nan, 'D', 'C'],
                      'col2': [2, 1, 9, 8, 7, 4],
                      'col3': [0, 1, 9, 4, 2, 3],
                      'col4': ['a', 'B', 'c', 'D', 'e', 'F']
                     })

# creating a frovedis dataframe
fd_df = fdf.DataFrame(pd_df)

# displaying the frovedis dataframe
fd_df.show()
```

Output

```
index   col1    col2    col3    col4
0       A       2       0       a
1       A       1       1       B
2       B       9       9       c
3       NULL    8       4       D
4       D       7       2       e
5       C       4       3       F
```

Sort by 'col1' on frovedis dataframe:

For example,

```
fd_df.sort_values(by=['col1']).show() #Sort by col1
```

Output

```
index   col1    col2    col3    col4
0       A       2       0       a
1       A       1       1       B
2       B       9       9       c
5       C       4       3       F
4       D       7       2       e
3       NULL    8       4       D
```

Sorting frovedis dataframe in ascending order (by default) by using multiple columns:

For example,

```
fd_df.sort_values(by=['col1', 'col2'], ascending = [True, True]).show()
```

Output

```
index   col1    col2    col3    col4
1       A       1       1       B
0       A       2       0       a
2       B       9       9       c
5       C       4       3       F
4       D       7       2       e
3       NULL    8       4       D
```

Sorting frovedis dataframe in descending orde by using ascending=False:

For example,

```
# Sort in descending order
fd_df.sort_values(by='col1', ascending=False).show()
```

Output

```
index   col1    col2    col3    col4
3       NULL    8       4       D
4       D       7       2       e
5       C       4       3       F
2       B       9       9       c
0       A       2       0       a
1       A       1       1       B
```

Return Value
It returns a new Frovedis DataFrame with sorted values.

56.2 SEE ALSO

- **DataFrame - Introduction**
- **DataFrame - Indexing Operations**
- **DataFrame - Generic Functions**
- **DataFrame - Conversion Functions**
- **DataFrame - Math Functions**
- **DataFrame - Aggregate Functions**

Chapter 57

DataFrame Aggregate Functions

57.1 NAME

DataFrame Aggregate Functions - list of all functions related to aggregate operations on frovedis dataframe are illustrated here.

57.1.1 DESCRIPTION

An essential piece of analysis of large data is efficient summarization: computing aggregations like sum(), mean(), median(), min(), and max(), which gives insight into the nature of a potentially large dataset. In this section, we will explore list of all such aggregation operations done on frovedis dataframe.

Aggregation can be performed in **two ways** on frovedis dataframe:

- Either using agg().
- Or using aggregation functions such as min(), max() median(), mode(), etc. on frovedis dataframe.

57.1.2 Public Member Functions

```
1. agg(func=None, axis=0, *args, **kwargs)
2. cov((min_periods = None, ddof = 1.0, low_memory = True, other = None)
3. mad(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)
4. max(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)
5. mean(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)
6. median(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)
7. min(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)
8. mode(axis = 0, numeric_only = False, dropna = True)
9. sem(axis = None, skipna = None, level = None, ddof = 1, numeric_only = None, **kwargs)
10. std(axis = None, skipna = None, level = None, ddof = 1, numeric_only = None, **kwargs)
11. sum(axis = None, skipna = None, level = None, numeric_only = None,
        min_count = 0, **kwargs)
12. var(axis = None, skipna = None, level = None, ddof = 1, numeric_only = None, **kwargs)
```

57.1.3 Detailed Description

57.1.3.1 1. DataFrame.agg(func=None, axis=0, *args, **kwargs)

Parameters
func: Names of functions to use for aggregating the data. The input to be used with the function must be a frovedis DataFrame instance having atleast one numeric column.
Accepted combinations for this parameter are:

- A string function name such as 'max', 'min', etc.
- list of functions and/or function names, For example, ['max', 'mean'].
- dictionary with keys as column labels and values as function name or list of such functions.
- In case func is None, then it will confirm if any keyword arguments are provided. If **kwargs are not provided then it will raise an Exception.

For Example, {'Age': ['max','min','mean'], 'Ename': ['count']}

axis: It accepts an integer or string object as parameter. It is used to decide whether to perform aggregation along the columns or rows. (Default: 0)

- **0 or 'index'**: perform aggregation along the indices.

***args**: This is an unused parameter.
****kwargs**: Additional keyword arguments to be passed to the function.

Purpose
It computes an aggregate operation based on the condition specified in 'func'.

Currently, this method will perform aggregation operation to each column.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Aggregate these functions over the rows, where func is a function string name:

For example,

```
print(fdf1.agg('max'))
```

Output

```
Name             nan
Age               36
City             nan
Qualification    nan
Score             52
Name: max, dtype: object
```

It displays a pandas dataframe containing numeric column(s) with newly computed aggregates of each groups.

Performing aggregation along the rows where func is a dictionary:

For example,

```
print(fdf1.agg({"Age": ["std", "mean"]}))
```

Output

```
            Age
mean  29.125000
std    4.853202
```

However, when aggregation is performed where func = 'cov', then it will perfom covariance only on numeric columns.

For example,

```
print(fdf1.agg('cov'))
```

Output

```
index   Age     Score
Age     23.5535 31.8
Score   31.8    123.766
```

Aggregation along the rows where func is a list of functions:

For example,

```
print(fdf1.agg(['max','min','mean']))
```

Output

```
      Name     Age  City  Qualification      Score
max    NaN  36.000   NaN            NaN  52.000000
min    NaN  22.000   NaN            NaN  23.000000
```

```
mean   NaN  29.125   NaN            NaN  39.833333
```

However, when list of functions contain combination of cov + other func, currently it will not compute covariance rather compute only aggregation on other function.

For example,

```
print(fdf1.agg(['cov','min']))
```

Output

```
    Name  Age City Qualification  Score
min  nan   22  nan           nan   23.0
```

Using keyword arguments in order to perform aggregation operation:

For example,

```
fdf1.agg(sum_age = ("Age", "sum"), min_score = ("Score", "min"))
```

Output

```
             Age  Score
sum_age    233.0    NaN
min_score    NaN   23.0
```

Return Value

1. **If one 'func' provided and 'func' is a string:**
   - It returns a pandas Series instance with numeric column(s) only, after aggregation function is completed.

2. **If one or more 'func' provided and 'func' is list/dict of string:**
   - It returns a pandas DataFrame instance with numeric column(s) only, after aggregation function is completed.

57.1.3.2 2. DataFrame.cov(min_periods = None, ddof = 1.0, low_memory = True, other = None)

Parameters
min_periods: It accepts an integer as parameter. It specifies the minimum number of observations required per pair of columns to have a valid result. (Default: None)
When it is None (not specified explicitly), then **min_periods = 1**.
ddof: It accepts a float parameter that specifies the delta degrees of freedom. (Default: 1.0)
low_memory: It accepts boolean parameter that specifies whethet to enable memory optimised computation or time optimsed computation. (Default: True)
other: It accepts frovedis dataframe as parameter, where it must be expressed in **"df[col_name]"** form. Also, it can be expressed in **"df.col_name"** form as well. (Default: None)

- **When it is not None (specified explicitly)**, it performs covariance operation between both the given frovedis dataframes. Although, the input dataframe must be used as expressions mentioned above.

- **When it is None (not specified explicitly)**, it will perform covarince on input dataframe to give covraince matrix represented as a dataframe.

Purpose
It computes pairwise covariance of columns, excluding missing values.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'Score': [23, 34, 35, 45, 23, 50, 52, 34]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Age     Score
0       27      23
1       24      34
2       22      35
3       32      45
4       33      23
5       36      50
6       27      52
7       32      34
```

Compute covariance on frovedis dataframe:

For example,

```
# cov() demo
fdf1.cov().show()
```

Output

```
index   Age     Score
Age     23.5535 11
Score   11      124.571
```

It displays a covariance matrix as the frovedis DataFrame instance.

Using min_periods parameter to calculate covariance:

For example,

```
# cov() demo using min_periods = 8
fdf1.cov(min_periods = 8).show()
```

Output

```
index   Age     Score
Age     NULL    NULL
Score   NULL    NULL
```

Using ddof parameter to calculate covariance:

For example,

```
# cov() demo using ddof = 2
fdf1.cov(ddof = 2).show()
```

Output

```
index   Age     Score
Age     27.4791 12.8333
Score   12.8333 145.333
```

In the below example, while using 'other' parameter, both inputs must be frovedis series. Also, the output returned by this method will be a float value.

For example,

```
# create another dataframe
pdf2 = pd.DataFrame({'Score': [51, 34, 33, 45, 12, 82, 67, 91]})
fdf2 = fdf.DataFrame(pdf2)

# cov() demo using 'other' parameter
print(fdf1['Score'].cov(other = fdf2['Score']))
```

Output

```
158.28571428571428
```

Here, it could also be expressed as **"fdf1['Score'].cov(other = fdf2['Score'])"**.

Note:- While using input dataframe in the form **'fdf1['Score']' or 'fdf1.Score'**, **'other'** parameter must be provided.

Return Value

- **If other = None:**
  It returns a covariance matrix represented as frovedis DataFrame instance.

- **If other != None:**
  It returns covariance as scalar value.

57.1.3.3 3. DataFrame.mad(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform mean absolute deviation along the columns or rows. (Default: None)

- **0 or 'index'**: perform mean absolute deviation along the indices.

- **1 or 'columns'**: perform mean absolute deviation along the columns.

When it is None (not specified explicitly), it performs mean absolute deviation along the rows.

skipna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during mean absolute deviation computation.
level: This is an unused parameter. (Default: None)

numeric_only: This is an unsed parameter. (Default: None)
****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes the mean absolute deviation of the values over the requested axis.

Currently, mean absolute deviation will be calculated for dataframe having atleast one numeric columns.

The parameters: "level", "numeric_only", "**kwargs" are simply kept in to make the interface uniform to the pandas DataFrame.mad(). These are not used internally in frovedis.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Mean absolute deviation along the rows (by default):

For example,

```
# mad() demo, axis = 0 by default
fdf1.mad().show()
```

Output

```
index   mad
Age     4.125
Score   9.16666
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed mean absolute deviation for each column.

Also, it excludes the missing value in **'Score'** column while computing the mean absolute deviation.

Mean absolute deviation along the rows and using skipna parameter:

For example,

```
# mad() demo using skipna = False
fdf1.mad(skipna = False).show()
```

Output

```
index   mad
Age     4.125
Score   NULL
```

Here, it includes the missing value in **'Score'** column while computing the mean absolute deviation.

Mean absolute deviation along the columns:

For example,

```
# mad() demo using axis = 1
fdf1.mad(axis = 1).show()
```

Output

```
index   mad
0       2
1       5
2       6.5
3       6.5
4       0
5       7
6       12.5
7       0
```

Mean absolute deviation along the columns and using skipna parameter:

For example,

```
# mad() demo using axis = 1 and skipna = False
fdf1.mad(axis = 1, skipna = False).show()
```

Output

```
index   mad
0       2
1       5
2       6.5
3       6.5
4       NULL
5       7
6       12.5
7       NULL
```

Return Value
It returns a frovedis DataFrame instance with the result of the specified aggregate operation.

57.1.3.4 4. DataFrame.max(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform maximum operation along the columns or rows. (Default: None)

- **0 or 'index'**: perform maximum operation along the indices to get the maximum value.
- **1 or 'columns'**: perform maximum operation along the columns to get the maximum value.

When it is None (not specified explicitly), it performs maximum operation along the rows.

skipna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during maximum value computation.
level: This is an unused parameter. (Default: None)
numeric_only: It accepts a boolean parameter. It determines whether only numeric columns are used or non-numeric also. (Default: None)

- **True**: Use only float, int, boolean columns.
- **False/None**: Attempt to use all columns(see note below).

*****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes the maximum of the values over the requested axis.

The parameters: "level", "**kwargs" are simply kept in to make the interface uniform to the pandas DataFrame.max(). These are not used internally in frovedis.

Note:-

- Parameter "numeric_only" is None(interpreted as False) by default, in this state all numeric columns and non-numeric columns including Datetime and Timedelta type are valid input. In case value is set as True then only numeric(float, int, boolean) columns are valid input.
- As of now Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of non-numeric columns of different types is not supported.

Following input dataframes would result in exception as explained above:

For example,

```
# numeric mixed with datetime
index   data0   data1
0       91      2022-11-30
1       21      2022-11-30
2       21      2022-11-29
3       43      2022-11-28

# Input dataframe with datetime mixed with timedelta
index   data1           data2
0       2022-11-30      1669766400000000091
1       2022-11-27      1669766400000000021
```

```
2       2022-11-29      1669766400000000041
3       2022-11-28      1669766400000000045

# Input dataframes like above would result in following exception:
TypeError: Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of
non-numeric columns of different types is not supported.
```

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Max operation along the rows (by default):

For example,

```
# max() demo
fdf1.max().show()
```

Output

```
index   max
Age     36
Score   52
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed maximum value for each column.

Also, it excludes the missing value in **'Score'** column while computing the maximum value.

Max operation along the rows and using skipna parameter:

For example,

```
# max() demo using skipna = False
fdf1.max(skipna = False).show()
```

Output

```
index   max
Age     36
Score   NULL
```

Here, it includes the missing value in **'Score'** column while computing the maximum value.

Max operation along the columns:

For example,

```
# max() demo using axis = 1
fdf1.max(axis = 1).show()
```

Output

```
index   max
0       27
1       34
2       35
3       45
4       33
5       50
6       52
7       32
```

Max operation along the columns and using skipna parameter:

For example,

```
# max() demo using axis = 1 and skipna = False
fdf1.max(axis = 1, skipna = False).show()
```

Output

```
index   max
0       27
1       34
2       35
3       45
4       NULL
5       50
6       52
7       NULL
```

Return Value
It returns a frovedis DataFrame instance with the result of the specified aggregate operation.

57.1.3.5 5. DataFrame.mean(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform mean along the columns or rows. (Default: None)
- **0 or 'index'**: perform mean along the indices.
- **1 or 'columns'**: perform mean along the columns.

When it is None (not specified explicitly), it performs mean operation along the rows.

skipna: It is a boolean parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during mean computation.
level: This is an unused parameter. (Default: None)
numeric_only: It accepts a boolean parameter. It determines whether only numeric columns are used or non-numeric also. (Default: None)

- **True**: Use only float, int, boolean columns.
- **False/None**: Attempt to use all columns(see note below).

*****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes mean of the values over the requested axis.

The parameters: "level", "**kwargs" are simply kept in to make the interface uniform to the pandas DataFrame.mean(). These are not used internally in frovedis.

Note:-

- Parameter "numeric_only" is None(interpreted as False) by default, in this state all numeric columns and non-numeric columns including Datetime and Timedelta type are valid input. In case value is set as True then only numeric(float, int, boolean) columns are valid input.
- As of now Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of non-numeric columns of different types is not supported.

Following input dataframes would result in exception as explained above:

For example,

```
# numeric mixed with datetime
index   data0   data1
0       91      2022-11-30
1       21      2022-11-30
2       21      2022-11-29
3       43      2022-11-28

# Input dataframe with datetime mixed with timedelta
index   data1           data2
0       2022-11-30      1669766400000000091
1       2022-11-27      1669766400000000021
2       2022-11-29      1669766400000000041
3       2022-11-28      1669766400000000045

# Input dataframes like above would result in following exception:
TypeError: Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of
```

non-numeric columns of different types is not supported.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Mean computation along the rows (by default):

For example,

```
# mean() demo
fdf1.mean(),show()
```

Output

```
index   mean
Age     29.125
Score   39.8333
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed mean for each column.

Also, it excludes the missing value in **'Score'** column while computing the mean.

Mean computation along the rows and using skipna parameter:

For example,

```
# mean() demo using skipna = False
fdf1.mean(skipna = False).show()
```

Output

```
index   mean
Age     29.125
Score   NULL
```

Here, it includes the missing value in **'Score'** column while computing the mean.

Max operation along the columns:

For example,

```
# mean() demo using axis = 1
fdf1.mean(axis = 1).show()
```

Output

```
index   mean
0       25
1       29
2       28.5
3       38.5
4       33
5       43
6       39.5
7       32
```

Max operation along the columns and using skipna parameter:

For example,

```
# mean() demo using axis = 1 and skipna = False
fdf1.mean(axis = 1, skipna = False).show()
```

Output

```
index   mean
0       25
1       29
2       28.5
3       38.5
4       NULL
5       43
6       39.5
7       NULL
```

Return Value
It returns a frovedis DataFrame instance.

57.1.3.6 6. DataFrame.median(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform median operation along the columns or rows. (Default: None)

- **0 or 'index'**: perform median operation along the indices.
- **1 or 'columns'**: perform median operation along the columns.

When it is None (not specified explicitly), it performs median operation along the rows.

skipna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during median computation.
level: This is an unused parameter. (Default: None)
numeric_only: It accepts a boolean parameter. It determines whether only numeric columns are used or non-numeric also. (Default: None)

- **True**: Use only float, int, boolean columns.
- **False/None**: Attempt to use all columns(see note below).

****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes median of the values over the requested axis.

The parameters: "level", "**kwargs" are simply kept in to make the interface uniform to the pandas DataFrame.mean(). These are not used internally in frovedis.

Note:-

- Parameter "numeric_only" is None(interpreted as False) by default, in this state all numeric columns and non-numeric columns including Datetime and Timedelta type are valid input. In case value is set as True then only numeric(float, int, boolean) columns are valid input.
- As of now Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of non-numeric columns of different types is not supported.

Following input dataframes would result in exception as explained above:

For example,

```
# numeric mixed with datetime
index   data0   data1
0       91      2022-11-30
1       21      2022-11-30
2       21      2022-11-29
3       43      2022-11-28

# Input dataframe with datetime mixed with timedelta
index   data1           data2
0       2022-11-30      1669766400000000091
1       2022-11-27      1669766400000000021
2       2022-11-29      1669766400000000041
3       2022-11-28      1669766400000000045

# Input dataframes like above would result in following exception:
TypeError: Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of
non-numeric columns of different types is not supported.
```

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
```

```
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Median computation along the rows (by default):

For example,

```
# median() demo
fdf1.median().show()
```

Output

```
index   median
Age     29.5
Score   40
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed median for each column.

Also, it excludes the missing value in **'Score'** column while computing the median.

Median computation along the rows and using skipna parameter:

For example,

```
# median() demo using skipna = False
fdf1.median(skipna = False).show()
```

Output

```
index   median
Age     29.5
Score   NULL
```

Here, it includes the missing value in **'Score'** column while computing the median.

Median computation along the columns:

For example,

```
# median() demo using axis = 1
fdf1.median(axis = 1).show()
```

Output

```
index   median
0       25
1       29
2       28.5
3       38.5
4       33
5       43
6       39.5
7       32
```

Median computation along the columns and using skipna parameter:

For example,

```
# median() demo using axis = 1 and skipna = False
fdf1.median(axis = 1, skipna = False).show()
```

Output

```
index   median
0       25
1       29
2       28.5
3       38.5
4       NULL
5       43
6       39.5
7       NULL
```

Return Value
It returns a frovedis DataFrame instance.

57.1.3.7 7. DataFrame.min(axis = None, skipna = None, level = None, numeric_only = None, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform minimum operation along the columns or rows. (Default: None)

- **0 or 'index'**: perform minimum operation along the indices to get the minimum value.
- **1 or 'columns'**: perform minimum operation along the columns to get the minimum value.

When it is None (not specified explicitly), it performs minimum operation along the rows.

skipna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during minimum value computation.
level: This is an unused parameter. (Default: None)
numeric_only: It accepts a boolean parameter. It determines whether only numeric columns are used or non-numeric also. (Default: None)

- **True**: Use only float, int, boolean columns.
- **False/None**: Attempt to use all columns(see note below).

****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes the minimum of the values over the requested axis.

The parameters: "level", "**kwargs" are simply kept in to make the interface uniform to the pandas DataFrame.min(). These are not used internally in frovedis.

Note:-

- Parameter "numeric_only" is None(interpreted as False) by default, in this state all numeric columns and non-numeric columns including Datetime and Timedelta type are valid input. In case value is set as True then only numeric(float, int, boolean) columns are valid input.
- As of now Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of non-numeric columns of different types is not supported.

Following input dataframes would result in exception as explained above:

For example,

```
# numeric mixed with datetime
index   data0   data1
0       91      2022-11-30
1       21      2022-11-30
2       21      2022-11-29
3       43      2022-11-28

# Input dataframe with datetime mixed with timedelta
index   data1           data2
0       2022-11-30      1669766400000000091
1       2022-11-27      1669766400000000021
2       2022-11-29      1669766400000000041
3       2022-11-28      1669766400000000045

# Input dataframes like above would result in following exception:
TypeError: Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of
non-numeric columns of different types is not supported.
```

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
```

```
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Min operation along the rows (by default):

For example,

```
# min() demo
fdf1.min().show()
```

Output

```
index   min
Age     22
Score   23
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed minimum value for each column.

Also, it excludes the missing value in **'Score'** column while computing the minimum value.

Min operation along the rows and using skipna parameter:

For example,

```
# min() demo using skipna = False
fdf1.min(skipna = False).show()
```

Output

```
index   min
Age     22
Score   NULL
```

Here, it includes the missing value in **'Score'** column while computing the median.

Min operation along the columns:

For example,

```
# min() demo using axis = 1
fdf1.min(axis = 1).show()
```

Output

```
index   min
0       23
1       24
2       22
3       32
4       33
5       36
6       27
7       32
```

Min operation along the columns and using skipna parameter:

For example,

```
# min() demo using axis = 1 and skipna = False
fdf1.min(axis = 1, skipna = False).show()
```

Output

```
index   min
0       23
1       24
2       22
3       32
4       NULL
5       36
6       27
7       NULL
```

Return Value
It returns a frovedis DataFrame instance with the result of the specified aggregate operation.

57.1.3.8 8. DataFrame.mode(axis = 0, numeric_only = False, dropna = True)

axis: It accepts an integer or string object as parameter. It is used to decide whether to perform standard error of the mean along the columns or rows. (Default: 0)

- **0 or 'index'**: perform mode along the indices.
- **1 or 'columns'**: perform mode along the columns.

When it is None (not specified explicitly), it performs standard error of the mean along the rows.

numeric_only: It accepts string object as parameter. If True, mode operation will result in a dataframe having only numeric columns. Otherwise, it will result in a dataframe having both numeric and non-numeric columns. (Default: False)
dropna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result of mode operation. (Default: True)
When it is None (not specified explicitly), it excludes missing values during mode computation.

Purpose
This method gets the mode(s) of each element along the selected axis.

The mode of a set of values is the value that appears most often. It can be multiple values.

When input with non-numeric columns is used with mode(), then it ignores the non-numeric columns for mode computation.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Mode computation along the rows (by default):

For example,

```
# mode() demo
fdf1.mode().show()
```

Output

```
index   Score   Name    Age     City    Qualification
0       23      Anuj    27      Kanpur  B.Tech
1       34      Jai     32      NULL    Phd
2       35      Princi  NULL    NULL    NULL
```

```
3       45      NULL    NULL    NULL    NULL
4       50      NULL    NULL    NULL    NULL
5       52      NULL    NULL    NULL    NULL
```

Mode will be calculated for dataframe having string and numeric columns when axis = 0 or 'index'.
Also, resultant dataframe has both numeric and non-numeric columns.

Mode computation along the rows and using numeric_only parameter:

For example,

```
# mode() demo using numeric_ony = True
fdf1.mode(numeric_ony = True).show()
```

Output

```
index   Score   Age
0       23      27
1       34      32
2       35      NULL
3       45      NULL
4       50      NULL
5       52      NULL
```

Here, resultant dataframe has only numeric columns.

Mode computation along the rows and using dropna parameter:

For example,

```
# mode() demo using dropna = False
fdf1.mode(dropna = False).show()
```

Output

```
index   Name    Age     City    Qualification  Score
0       Anuj    27      Kanpur  B.Tech         NULL
1       Jai     32      NULL    Phd            NULL
2       Princi  NULL    NULL    NULL           NULL
```

For axis = 1 or 'columns', mode will be calculated for dataframe having only numeric columns.

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
bmiDF = {
        'height':[157, 124, 162, np.nan, 133, 176, np.nan, 152],
        'weight': [53, 64, np.nan, 65, 63, 80, np.nan, 84]
        }

# create pandas dataframe with only numeric columns
pdf1 = pd.DataFrame(bmiDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)
```

```
# display the frovedis dataframe
fdf1.show()
```

Output

```
index   height  weight
0       157     53
1       124     64
2       162     NULL
3       NULL    65
4       133     63
5       176     80
6       NULL    NULL
7       152     84
```

Mode computation along the columns:

For example,

```
# mode() demo using axis = 1
fdf1.mode(axis = 1).show()
```

Output

```
index   0
0       157
1       124
2       162
3       65
4       133
5       176
6       0
7       152
```

Mode computation along the columns and using dropna parameter:

For example,

```
# mode() demo using axis = 1 and dropna = False
fdf1.mode(axis = 1, dropna = False).show()
```

Output

```
index   0
0       157
1       124
2       162
3       NULL
4       133
5       176
6       NULL
7       152
```

Return Value

- **If numeric_only = False**:
  It returns a frovedis DataFrame instance having both numeric and non-numeric columns (if any).

- **If numeric_only = True**:
  It returns a frovedis DataFrame instance having only numeric columns.

57.1.3.9 9. DataFrame.sem(axis = None, skipna = None, level = None, ddof = 1, numeric_only = None, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform standard error of the mean along the columns or rows. (Default: None)

- **0 or 'index'**: perform standard error of the mean along the indices.
- **1 or 'columns'**: perform standard error of the mean along the columns.

When it is None (not specified explicitly), it performs standard error of the mean along the rows.

skipna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during standard error of the mean computation.
level: This is an unused parameter. (Default: None)
ddof: It accepts an integer parameter that specifies the delta degrees of freedom. (Default: 1)
numeric_only: This is an unsed parameter. (Default: None)
****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes standard error of the mean over requested axis.

Currently, standard error of the mean will be calculated for dataframe having atleast one numeric columns.

The parameters: "level", "numeric_only", "**kwargs" are simply kept in to make the interface uniform to the pandas DataFrame.sem(). These are not used internally in frovedis.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Standard error of the mean operation along the rows (by default):

For example,

```
# sem() demo
fdf1.sem().show()
```

Output

```
index   sem
Age     1.71586
Score   4.54178
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed standard error of the mean for each column.

Standard error of the mean operation along the rows and using ddof parameter:

For example,

```
# sem() demo using ddof = 2
fdf1.sem(ddof = 2).show()
```

Output

```
index   sem
Age     1.85334
Score   5.07786
```

Standard error of the mean operation along the rows and using skipna parameter:

For example,

```
# sem() demo using skipna = False
fdf1.sem(skipna = False).show()
```

Output

```
index   sem
Age     1.71586
Score   NULL
```

Standard error of the mean operation along the columns:

For example,

```
# sem() demo using axis = 1
fdf1.sem(axis = 1).show()
```

Output

```
index   sem
0       2
1       5
```

```
2       6.49999
3       6.49999
4       NULL
5       6.99999
6       12.5
7       NULL
```

Standard error of the mean operation along the columns and using skipna parameter:

For example,

```
# sem() demo using axis = 1 and skipna = False
fdf1.sem(axis = 1, skipna = False).show()
```

Output

```
index   sem
0       2
1       5
2       6.49999
3       6.49999
4       NULL
5       6.99999
6       12.5
7       NULL
```

Return Value
It returns a frovedis DataFrame instance with the result of the specified aggregate operation.

57.1.3.10 10. DataFrame.std(axis = None, skipna = None, level = None, ddof = 1, numeric_only = None, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform standard deviation along the columns or rows. (Default: None)

- **0 or 'index'**: perform standard deviation along the indices.
- **1 or 'columns'**: perform standard deviation along the columns.

When it is None (not specified explicitly), it performs standard deviation along the rows.
skipna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during standard deviation computation.
level: This is an unused parameter. (Default: None)
ddof: It accepts an integer parameter that specifies the delta degrees of freedom. (Default: 1)
numeric_only: It accepts a boolean parameter. It determines whether only numeric columns are used or non-numeric also. (Default: None)

- **True**: Use only float, int, boolean columns.
- **False/None**: Attempt to use all columns(see note below).

*****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes standard deviation over requested axis.

The parameters: “level”, “**kwargs” are simply kept in to make the interface uniform to the pandas DataFrame.min(). These are not used internally in frovedis.

Note:-

- Parameter “numeric_only” is None(interpreted as False) by default, in this state all numeric columns and non-numeric columns including Datetime and Timedelta type are valid input. In case value is set as True then only numeric(float, int, boolean) columns are valid input.

- As of now Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of non-numeric columns of different types is not supported.

Following input dataframes would result in exception as explained above:

For example,

```
# numeric mixed with datetime
index   data0   data1
0       91      2022-11-30
1       21      2022-11-30
2       21      2022-11-29
3       43      2022-11-28

# Input dataframe with datetime mixed with timedelta
index   data1           data2
0       2022-11-30      1669766400000000091
1       2022-11-27      1669766400000000021
2       2022-11-29      1669766400000000041
3       2022-11-28      1669766400000000045

# Input dataframes like above would result in following exception:
TypeError: Frovedis does not support mixing of numeric and non-numeric columns. Also mixing of
non-numeric columns of different types is not supported.
```

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)
```

```
# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Standard deviation computation along the rows (by default):

For example,

```
# std() demo
fdf1.std().show()
```

Output

```
index   std
Age     4.8532
Score   11.125
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed standard deviation for each column.

Standard deviation computation along the rows and using skipna parameter:

For example,

```
# std() demo using skipna = False
fdf1.std(skipna = False).show()
```

Output

```
index   std
Age     4.8532
Score   NULL
```

Standard deviation computation along the rows and using ddof parameter:

For example,

```
# std() demo using ddof = 2
fdf1.std(ddof = 2).show()
```

Output

```
index   std
Age     5.24205
Score   12.4381
```

Standard deviation computation along the columns:

For example,

```
# std() demo using axis = 1
fdf1.std(axis = 1).show()
```

Output

```
index   std
0       2.82842
1       7.07106
2       9.19238
3       9.19238
4       NULL
5       9.89949
6       17.6776
7       NULL
```

Standard deviation computation along the columns and using skipna parameter:

For example,

```
# std() demo using axis = 1 and skipna = False
fdf1.std(axis = 1, skipna = False).show()
```

Output

```
index   std
0       2.82842
1       7.07106
2       9.19238
3       9.19238
4       NULL
5       9.89949
6       17.6776
7       NULL
```

Return Value
It returns a frovedis DataFrame instance with the result of the specified aggregate functions.

57.1.3.11 11. DataFrame.sum(axis = None, skipna = None, level = None, numeric_only = None, min_count = 0, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform summation operation along the columns or rows. (Default: None)

- **0 or 'index'**: perform summation operation along the indices.
- **1 or 'columns'**: perform summation operation along the columns.

When it is None (not specified explicitly), it performs summation operation along the rows.
skipna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during summation computation.
level: This is an unused parameter. (Default: None)
numeric_only: This is an unsed parameter. (Default: None)
min_count: It is an integer, float or double (float64) parameter that specifies the minimum number of values that needs to be present to perform the action. (Default: 0)
*****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes the sum of the values over the requested axis.

Currently, summation will be calculated for dataframe having atleast one numeric columns.

The parameters: “level”, “numeric_only”, “**kwargs” are simply kept in to make the interface uniform to the pandas DataFrame.sum(). These are not used internally in frovedis.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Sum operation along the rows (by default):

For example,

```
# sum() demo
fdf1.sum().show()
```

Output

```
index   sum
Age     233
Score   239
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed summation for each column.

Also, it excludes the missing value in **‘Score’** column while computing summation.

Sum operation along the rows and using skipna parameter:

For example,

```
# sum() demo using skipna = False
fdf1.sum(skipna = False).show()
```

Output

```
index   sum
Age     233
Score   NULL
```

Here, it includes the missing value in **'Score'** column while computing the sum.

Sum operation along the columns:

For example,

```
# sum() demo using axis = 1
fdf1.sum(axis = 1).show()
```

Output

```
index   sum
0       50
1       58
2       57
3       77
4       33
5       86
6       79
7       32
```

Sum operation along the columns and using skipna parameter:

For example,

```
# sum() demo using axis = 1 and skipna = False
fdf1.sum(axis = 1, skipna = False).show()
```

Output

```
index   sum
0       50
1       58
2       57
3       77
4       NULL
5       86
6       79
7       NULL
```

Return Value
It returns a frovedis DataFrame instance with the result of the specified aggregate functions.

57.1.3.12 12. DataFrame.var(axis = None, skipna = None, level = None, ddof = 1, numeric_only = None, **kwargs)

Parameters
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform variance

along the columns or rows. (Default: None)

- **0 or 'index'**: perform variance along the indices.
- **1 or 'columns'**: perform variance along the columns.

When it is None (not specified explicitly), it performs variance along the rows.
skipna: It accepts boolean as parameter. When set to True, it will exclude missing values while computing the result. (Default: None)
When it is None (not specified explicitly), it excludes missing values during variance computation.
level: This is an unused parameter. (Default: None)
ddof: It accepts an integer parameter that specifies the delta degrees of freedom. (Default: 1)
numeric_only: This is an unsed parameter. (Default: None)
****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes variance over requested axis.

Currently, variance will be calculated for dataframe having atleast one numeric columns.

The parameters: "level", "numeric_only", "**kwargs" are simply kept in to make the interface uniform to the pandas DataFrame.var(). These are not used internally in frovedis.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
```

```
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Variance computation along the rows (by default):

For example,

```
# var() demo
fdf1.var().show()
```

Output

```
index   var
Age     23.5535
Score   123.766
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed variance for each column.

Also, it excludes the missing value in **'Score'** column while computing variance.

Variance computation along the rows and using skipna parameter:

For example,

```
# var() demo using skipna = False
fdf1.var(skipna = False).show()
```

Output

```
index   var
Age     23.5535
Score   NULL
```

Variance computation along the rows and using ddof parameter:

For example,

```
# var() demo using ddof = 2
fdf1.var(ddof = 2).show()
```

Output

```
index   var
Age     27.4791
Score   154.708
```

Variance computation along the columns:

For example,

```
# var() demo using axis = 1
fdf1.var(axis = 1).show()
```

Output

```
index   var
0       8
1       50
2       84.5
3       84.5
4       NULL
5       98
6       312.5
7       NULL
```

Variance computation along the columns and using skipna parameter:

For example,

```
# var() demo using axis = 1 and skipna = False
fdf1.var(axis = 1, skipna = False).show()
```

Output

```
index   var
0       8
1       50
2       84.5
3       84.5
4       NULL
5       98
6       312.5
7       NULL
```

Return Value
It returns a frovedis DataFrame instance with the result of the specified aggregate operation.

57.2 SEE ALSO

- **DataFrame - Introduction**
- **DataFrame - Indexing Operations**
- **DataFrame - Generic Fucntions**
- **DataFrame - Conversion Functions**
- **DataFrame - Sorting Functions**
- **DataFrame - Math Functions**

Chapter 58

DataFrame Math Functions

58.1 NAME

DataFrame Math Functions - this manual contains all the methods for carrying out mathematical operations.

58.1.1 DESCRIPTION

Frovedis dataframe has several math functions defined for performing operations like add(), sub(), mul(), etc. between two dataframes or between scalar value and dataframe. These functions return a new frovedis DataFrame instance as a result.

Also, it contains reverse operations such as radd(), rsub(), rmul(), etc.

However, there are some special cases while using frovedis dataframe with mathematical operations as mentioned below:

Binary operation on two frovedis dataframes having same columns but different datatypes:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

pdf1 = pd.DataFrame(data1)
fdf1 = fdf.DataFrame(pdf1)

data2 = {
         "points": [2., 3., 8.],
         "total": [7, 9, 11]
        }

pdf2 = pd.DataFrame(data2)
fdf2 = fdf.DataFrame(pdf2)

print(fdf1 + fdf2)
```

Output

```
index   points  total
0       7       17
1       9       20
2       12      23
```

Here, **'points'** column in first dataframe is **int type** and in other dataframe is **float type**.

Type conversion occurs for 'points' column in resultant dataframe.

```
fdf1.points(int) + fdf2.points(float) -> res.points(float)
```

Also, binary operation can be performed between columns in dataframe as well.

For example,

```
print(fdf1["points"] + fdf2["points"])
```

Output

```
index   (points+points)
0       7
1       9
2       12
```

The above expression can also be written as follows:

```
print(fdf1.points + fdf2.points)
```

Output

```
index   (points+points)
0       7
1       9
2       12
```

Binary operation on two frovedis dataframes having atleast one common column:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

pdf1 = pd.DataFrame(data1)
fdf1 = fdf.DataFrame(pdf1)

data2 = {
         "score": [2, 3, 8],
         "total": [7, 9, 11]
        }

pdf2 = pd.DataFrame(data2)
fdf2 = fdf.DataFrame(pdf2)

print(fdf1 + fdf2)
```

Output

```
index   points  score   total
0       NULL    NULL    17
1       NULL    NULL    20
2       NULL    NULL    23
```

Here, the resultant dataframe will contains all columns from input frovedis dataframes lexicographically. Also, binary operation (i.e. addition) is performed on the common column only.

Binary operation on two frovedis dataframes having same columns. Also, same indices but in different order:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

pdf1 = pd.DataFrame(data1, index = [1,2,3])
fdf1 = fdf.DataFrame(pdf1)

data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

pdf2 = pd.DataFrame(data2, index = [2,3,1])
fdf2 = fdf.DataFrame(pdf2)

print(fdf1 + fdf2)
```

Output

```
index   points  total
1       13      21
2       8       18
3       7       21
```

Binary operation (i.e. addition) will be performed between same indices for each column irrespective of the index order.

Currently, binary operation on two frovedis dataframes having same columns but different indices is not supported.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

pdf1 = pd.DataFrame(data1, index = [0,1,2])
fdf1 = fdf.DataFrame(pdf1)

data2 = {
```

```
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

pdf2 = pd.DataFrame(data2, index = [0,1,3])
fdf2 = fdf.DataFrame(pdf2)

print(fdf1 + fdf2)
```

In this case, dataframes have same column but indices are different. This will raise an exception in frovedis.

Binary operation between columns of frovedis dataframes. Also, having same indices:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

pdf1 = pd.DataFrame(data1, index = [1,2,3])
fdf1 = fdf.DataFrame(pdf1)

data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

pdf2 = pd.DataFrame(data2, index = [1,2,3])
fdf2 = fdf.DataFrame(pdf2)

print(fdf1['points'] + fdf2['points'])
```

Output

```
index   (points+points)
1       7
2       9
3       12
```

Binary operation between two frovedis dataframes having single column (common) only. Also, having same indices but different order:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
pdf1 = pd.Series([27, 24, 22, 32, 33, 36, 27, 32],index = [0,1,2,3,4,5,6,7])
pdf2 = pd.Series([23, 34, 35, 45, 23, 50, 52, 34],index = [2,3,1,0,7,4,5,6])

fdf1 = fdf.DataFrame(pdf1)
fdf2 = fdf.DataFrame(pdf2)

print(fdf1 + fdf2)
```

Output

```
index   (0+0)
0       72
1       59
2       45
3       66
4       83
5       88
6       61
7       55
```

Binary operation (i.e. addition) will be performed between same indices irrespective of the index order in the frovedis dataframe.

Currently, binary operation on two frovedis dataframes single column only but different indices is not supported.

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
pdf1 = pd.Series([27, 24, 22, 32, 33, 36, 27, 32],index = [0,1,2,3,4,5,6,7])
pdf2 = pd.Series([23, 34, 35, 45, 23, 50, 52, 34],index = [2,3,1,0,8,4,5,9])

fdf1 = fdf.DataFrame(pdf1)
fdf2 = fdf.DataFrame(pdf2)

print(fdf1 + fdf2)
```

In this case, resultant dataframe has single column but indices are different. This will raise an exception in frovedis.

Binary operation between frovedis dataframe and scalar value (float type):

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

pdf1 = pd.DataFrame(data1)
fdf1 = fdf.DataFrame(pdf1)

print(fdf1 + 12.)
```

Output

```
index   points  total
0       17      22
1       18      23
2       16      24
```

Here, binary operation (i.e. addition) will be performed between the scalar value of float type on each column of the resultant frovedis dataframe.

Binary operation between frovedis dataframe and row vector:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

pdf1 = pd.DataFrame(data1)
fdf1 = fdf.DataFrame(pdf1)

print(fdf1 + [51, 34])
```

Output

```
index   points  total
0       56      44
1       57      45
2       55      46
```

Here, binary operation (i.e addition) will be performed between each element of row vector with each column on the given index of the frovedis dataframe.

Binary operation between frovedis dataframe and column vector:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

pdf1 = pd.DataFrame(data1)
fdf1 = fdf.DataFrame(pdf1)

print(fdf1['points'] + [21, 34, 45])
print(fdf1['total'] + [21, 34, 45])
```

Output

```
index   points
0       26
1       40
2       49

index   total
0       31
1       45
2       57
```

Here, binary operation (i.e addition) will be performed between the array and the given column of frovedis dataframe.

58.1.2 Public Member Functions

```
1. abs()
2. add(other, axis = 'columns', level = None, fill_value = None)
```

```
3. div(other, axis = 'columns', level = None, fill_value = None)
4. floordiv(other, axis = 'columns', level = None, fill_value = None)
5. mod(other, axis = 'columns', level = None, fill_value = None)
6. mul(other, axis = 'columns', level = None, fill_value = None)
7. pow(other, axis = 'columns', level = None, fill_value = None)
8. sub(other, axis = 'columns', level = None, fill_value = None)
9. truediv(other, axis = 'columns', level = None, fill_value = None)
10. radd(other, axis = 'columns', level = None, fill_value = None)
11. rdiv(other, axis = 'columns', level = None, fill_value = None)
12. rfloordiv(other, axis = 'columns', level = None, fill_value = None)
13. rmod(other, axis = 'columns', level = None, fill_value = None)
14. rmul(other, axis = 'columns', level = None, fill_value = None)
15. rpow(other, axis = 'columns', level = None, fill_value = None)
16. rsub(other, axis = 'columns', level = None, fill_value = None)
17. rtruediv(other, axis = 'columns', level = None, fill_value = None)
```

58.1.3 Detailed Description

58.1.3.1 1. DataFrame.abs()

Purpose
It computes absolute numeric value of each element.

This function only applies to elements that are all numeric.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
tempDF = {
          'City': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                   'Kanpur', 'Kanpur', 'Kanpur'],
          'Temperature': [-2, 10, 18, 34, -8, -4, 36, 45]
         }

# create pandas dataframe
pdf1 = pd.DataFrame(tempDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   City       Temperature
0       Nagpur     -2
1       Kanpur     10
2       Allahabad  18
3       Kannuaj    34
```

```
4       Allahabad  -8
5       Kanpur     -4
6       Kanpur     36
7       Kanpur     45
```

Absolute numeric values in a frovedis dataframe:

For example,

```
# abs() demo
print(fdf1['Temperature'].abs())
```

Output

```
index   Temperature
0       2
1       10
2       18
3       34
4       8
5       4
6       36
7       45
```

Return Value
It returns a frovedis DataFrame instance.

58.1.3.2 2. DataFrame.add(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be added with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform addition operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform addition operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs addition between two operands. It is equivalent to **'dataframe + other'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Add a scalar value using operator version:

For example,

```
print(fdf1 + 10)
```

Output

```
index   points  total
0       15      20
1       16      21
2       14      22
```

Add a scalar value using method version:

For example,

```
fdf1.add(10).show()
```

Output

```
index   points  total
0       15      20
1       16      21
2       14      22
```

In both versions, all column elements (axis = 1 by default) are added with a scalar value.

Creating two frovedis dataframes to perform addition:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Add two dataframes using operator version:

For example,

```
print(fdf1 + fdf2)
```

Output

```
index   points  total
```

```
0       7       17
1       9       20
2       12      23
```

Add two dataframes using method version:

For example,

```
fdf1.add(other = fdf2).show()
```

Output

```
index   points  total
0       7       17
1       9       20
2       12      23
```

In both versions, only common columns in both dataframes are added. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes in order to use fill_value parameter during addition:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)
```

```
# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Add two dataframes and using fill_value parameter:

For example,

```
# add() demo on two dataframes using method version and fill_value = 10
fdf1.add(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       7       17
1       9       21
2       12      20
```

Here, only common columns in both dataframes are added, excluding the misiing values. Other column elements are added with the fill_value = 10 (exluding missing values) in resultant dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.3 3. DataFrame.div(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be divided over the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform division operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform division operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element

needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs floating division operation between two operands. It is equivalent to **'dataframe / other'**.

It is an alias of truediv().

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Divide a scalar value using operator version:

For example,

```
print(fdf1 / 10)
```

Output

```
index   points  total
0       0.5     1
1       0.6     1.1
2       0.4     1.19999
```

Divide a scalar value using method version:

For example,

```
fdf1.div(10).show()
```

Output

```
index   points  total
0       0.5     1
1       0.6     1.1
```

```
2       0.4     1.19999
```

In both versions, all column elements (axis = 1 by default) are divied by a scalar value.

Creating two frovedis dataframes to perform division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Divide two dataframes using operator version:

For example,

```
print(fdf1 / fdf2)
```

Output

```
index   points  total
0       2.5     1.42857
1       2       1.22222
2       0.5     1.0909
```

Divide two dataframes using method version:

For example,

```
fdf1.div(other = fdf2).show()
```

Output

```
index   points  total
0       2.5     1.42857
1       2       1.22222
2       0.5     1.0909
```

In both versions, only common columns in both dataframes are divided. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes in order to use fill_value parameter during division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }
```

```
# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Divide two dataframes and using fill_value parameter:

For example,

```
# div() demo on two dataframes using method version and fill_value = 10
fdf1.div(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       2.5     1.42857
1       2       1.1
2       0.5     1
```

Here, only common columns in both dataframes are divided, excluding the missing values. Other column elements are divided with the fill_value = 10 (excluding the missing values) in resultant dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.4 4. DataFrame.floordiv(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be divided over the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform division operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform division operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs integer division operation between operands. It is equivalent to **'dataframe // other'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Floor Division on a scalar value using operator version:

For example,

```
print(fdf1 // 10)
```

Output

```
index   points  total
0       0       1
1       0       1
2       0       1
```

Floor Division on a scalar value using method version:

For example,

```
fdf1.floordiv(10).show()
```

Output

```
index   points  total
0       0       1
1       0       1
2       0       1
```

In both versions, all column elements (axis = 1 by default) are divied by a scalar value.
Also, resultant dataframe column elements will contain floor integer value.

Creating two frovedis dataframes to perform floor division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
```

```
2       8       11
```

Floor Division on two dataframes using operator version:

For example,

```
print(fdf1 // fdf2)
```

Output

```
index   points  total
0       2       1
1       2       1
2       0       1
```

Floor Division on two dataframes using method version:

For example,

```
fdf1.floordiv(other = fdf2).show()
```

Output

```
index   points  total
0       2       1
1       2       1
2       0       1
```

In both versions, only common columns in both dataframes are divided. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes in order to use fill_value parameter during floor division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Floor Division on two dataframes and using fill_value parameter:

For example,

```
# floordiv() demo on two dataframes using method version and fill_value = 10
fdf1.floordiv(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       2       1
1       2       1
2       0       1
```

Here, only common columns in both dataframes are divided, excluding the missing values. Other column elements are divided with the fill_value = 10 (excluding the missing values) in resultant dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.5 5. DataFrame.mod(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series

- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to perform modulo operation with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform modulo operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform modulo operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs modulo operation between two operands. It is equivalent to **'dataframe % other'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame()

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Modulo on a scalar value using operator version:

For example,

```
print(fdf1 % 10)
```

Output

```
index   points  total
0       5       0
1       6       1
2       4       2
```

Modulo on a scalar value using method version:

For example,

```
fdf1.mod(10).show()
```

Output

```
index   points  total
0       5       0
1       6       1
2       4       2
```

In both versions, modulo operation is performed on all column elements (axis = 1 by default) by a scalar value.

Creating two frovedis dataframes to perform modulo:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Modulo on two dataframes using operator version:

For example,

```
print(fdf1 % fdf2)
```

Output

```
index   points  total
0       1       3
1       0       2
2       4       1
```

Creating two frovedis dataframes to perform modulo, use fill_value parameter too:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)
```

```
# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Modulo on two dataframes using method version:

For example,

```
fdf1.mod(other = fdf2).show()
```

Output

```
index   points  total
0       1       3
1       0       2
2       4       1
```

In both versions, modulo is performed on only common columns in both dataframes. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Modulo on two dataframes and using fill_value parameter:

For example,

```
# mod() demo on two dataframes using method version and fill_value = 10
fdf1.mod(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       1       3
1       0       1
2       4       0
```

Here, modulo is performed on only common columns in both dataframes, excluding the missing values. Modulo is performed on other column elements using the value 10 (excluding the missing values) in resultant dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.6 6. DataFrame.mul(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.

- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be multiplied with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform multiplication operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform multiplication operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs multiplication operation between two operands. It is equivalent to **'dataframe * other'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Multiply a scalar value using operator version:

For example,

```
print(fdf1 * 10)
```

Output

```
index   points  total
0       50      100
1       60      110
2       40      120
```

Multiply a scalar value using method version:

For example,

```
fdf1.mul(10).show()
```

Output

```
index   points  total
0       50      100
1       60      110
2       40      120
```

In both versions, all column elements (axis = 1 by default) are multiplied with a scalar value.

Creating two frovedis dataframes to perform multiplication:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)
```

```
# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Multiply two dataframes using operator version:

For example,

```
print(fdf1 * fdf2)
```

Output

```
index   points  total
0       10      70
1       18      99
2       32      132
```

Multiply two dataframes using method version:

For example,

```
# mul() demo on two dataframes using method version
fdf1.mul(other = fdf2).show()
```

Output

```
index   points  total
0       10      70
1       18      99
2       32      132
```

In both versions, only common columns in both dataframes are multiplied. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes in order to use fill_value parameter during multiplication:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)
```

```
# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Multiply two dataframes and using fill_value parameter:

For example,

```
# mul() demo on two dataframes using method version and fill_value = 10
fdf1.mul(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       10      70
1       18      110
2       32      100
```

Here, only common columns in both dataframes are multiplied, excluding the missing values. Other column elements are multiplied with the value 10 (excluding the missing values) in resultant dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.7 7. DataFrame.pow(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to perform exponential power operation with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform exponential power operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform exponential power operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs exponential power operation between two operands. It is equivalent to **'dataframe ** other'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame()

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Exponential power operation on a scalar value using operator version:

For example,

```
print(fdf1 ** 2)
```

Output

```
index   points  total
0       10      20
1       12      22
2       8       24
```

Exponential power operation on a scalar value using method version:

For example,

```
fdf1.pow(2).show()
```

Output

```
index   points  total
0       10      20
1       12      22
2       8       24
```

In both versions, exponential power operation is performed on all column elements (axis = 1 by default) by a scalar value.

Creating two frovedis dataframes to perform exponential power operation:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
```

```
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Exponential power operation on two dataframes using operator version:

For example,

```
print(fdf1 ** fdf2)
```

Output

```
index   points  total
0       10      70
1       18      99
2       32      132
```

Exponential power operation on two dataframes using method version:

For example,

```
fdf1.pow(other = fdf2).show()
```

Output

```
index   points  total
0       10      70
1       18      99
2       32      132
```

In both versions, exponential power operation on only common columns in both dataframes. Exponential power operation on other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during exponential power operation:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
```

```
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Exponential power operation on two dataframes and using fill_value parameter:

For example,

```
# pow() on two dataframes using method version and fill_value = 10
fdf1.pow(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       25      1.00000e+07
1       216     2.59374e+10
2       65536   1.00000e+10
```

Here, exponential power operation is performed on only common columns in both dataframes, excluding the missing values. Exponential power operation on other column elements is performed with the value 10 (excluding the missing values) in resultant dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.8 8. DataFrame.sub(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be subtracted with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform subtraction operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform subtraction operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs subtraction operation between two operands. It is equivalent to **'dataframe - other'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
```

```
0       5       10
1       6       11
2       4       12
```

Subtract a scalar value using operator version:

For example,

```
print(fdf1 - 10)
```

Output

```
index   points  total
0       -5      0
1       -4      1
2       -6      2
```

Subtract a scalar value using method version:

For example,

```
fdf1.sub(10).show()
```

Output

```
index   points  total
0       -5      0
1       -4      1
2       -6      2
```

In both versions, all column elements (axis = 1 by default) are subtracted by a scalar value.

Creating two frovedis dataframes to perform subtraction:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Subtract two dataframes using operator version:

For example,

```
print(fdf1 - fdf2)
```

Output

```
index   points  total
0       3       3
1       3       2
2       -4      1
```

Subtract two dataframes using method version:

For example,

```
fdf1.sub(other = fdf2).show()
```

Output

```
index   points  total
0       3       3
1       3       2
2       -4      1
```

In both versions, only common columns in both dataframes are subtracted. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during subtraction:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
```

```
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Subtract two dataframes and using fill_value parameter:

For example,

```
# sub() demo on two dataframes using method version and fill_value = 10
fdf1.sub(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       3       3
1       3       1
2       -4      0
```

Here, only common columns in both dataframes are subtracted, excluding the missing values. Other column elements are subtracted with the value 10 (excluding the missing values) in resultant dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.9 9. DataFrame.truediv(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be divided with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform division operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform division operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs floating division operation between two operands. It is equivalent to **'dataframe / other'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [8, 5, 9],
         "total": [3, 2, 1]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
```

```
fdf1.show()
```

Output

```
index   points  total
0       8       3
1       5       2
2       9       1
```

Floating Division on a scalar value using operator version:

For example,

```
print(fdf1 / 10)
```

Output

```
index   points  total
0       0.8     0.3
1       0.5     0.2
2       0.9     0.1
```

Floating Division on a scalar value using method version:

For example,

```
fdf1.truediv(10).show()
```

Output

```
index   points  total
0       0.8     0.3
1       0.5     0.2
2       0.9     0.1
```

In both versions, all column elements (axis = 1 by default) are divied by a scalar value.

Creating two frovedis dataframes to perform floating division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [8, 5, 9],
         "total": [3, 2, 1]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       8       3
```

```
1       5       2
2       9       1
```

For example,

```
# a dictionary
data2 = {
         "points": [4, 7, 2],
         "total": [9, 1, 9]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       4       9
1       7       1
2       2       9
```

Floating Division on two dataframes using operator version:

For example,

```
print(fdf1 / fdf2)
```

Output

```
index   points    total
0       2         0.333333
1       0.714285  2
2       4.5       0.111111
```

Floating Division on two dataframes using method version:

For example,

```
fdf1.truediv(other = fdf2).show()
```

Output

```
index   points    total
0       2         0.333333
1       0.714285  2
2       4.5       0.111111
```

In both versions, only common columns in both dataframes are divided. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during floating division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
```

```
# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Floating Division on two dataframes and using fill_value parameter:

For example,

```
# truediv() demo on two dataframes using method version and fill_value = 10
fdf1.truediv(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       2.5     1.42857
1       2       1.1
2       0.5     1
```

Here, only common columns in both dataframes are divided, excluding the missing values. Other column elements are divided with the value 10 (excluding the missing values) in resultant dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.10 10. DataFrame.radd(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be added with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform reverse addition operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform reverse addition operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs reverse addition operation between two operands. It is equivalent to **'other + dataframe'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)
```

```
# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Reverse addition on a scalar value using operator version:

For example,

```
print(10 + fdf1)
```

Output

```
index   points  total
0       15      20
1       16      21
2       14      22
```

Reverse addition on a scalar value using method version:

For example,

```
# radd() demo with scalar value using method version
fdf1.radd(10).show()
```

Output

```
index   points  total
0       15      20
1       16      21
2       14      22
```

Here, it adds the scalar to all columns in dataframe (axis = 1 by default).

Creating two frovedis dataframes to perform reverse addition:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For examples,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Reverse addition on two dataframes using operator version:

For example,

```
print(fdf2 + fdf1)
```

Output

```
index   points  total
0       7       17
1       9       20
2       12      23
```

Reverse addition on two dataframes using method version:

For example,

```
# radd() demo on two dataframes using method version
fdf1.radd(other = fdf2).show()
```

Output

```
index   points  total
0       7       17
1       9       20
2       12      23
```

Here, only common columns in both dataframes are added. Column values in other datframe are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during reverse addition:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Reverse addition on two dataframes and using fill_value parameter:

For example,

```
# radd() demo on two dataframes using method version and fill_value = 10
fdf1.radd(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       7       17
1       9       21
2       12      20
```

Here, only common columns in both dataframes are added excluding the missing values. The fill_value = 10 is added to both column values in the dataframe (excluding the missing values) and stored in new dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.11 11. DataFrame.rdiv(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be divided with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform reverse division operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform reverse division operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs reverse floating division operation between two operands. It is equivalent to **'other / dataframe'**.

It is an alias of rtruediv().

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }
```

```
# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Reverse division on a scalar value using operator version:

For example,

```
print(10 / fdf1)
```

Output

```
index   points  total
0       2       1
1       1.66666 0.90909
2       2.5     0.833333
```

Reverse division on a scalar value using method version:

For example,

```
# rdiv() demo with scalar value using method version
fdf1.rdiv(10).show()
```

Output

```
index   points  total
0       2       1
1       1.66666 0.90909
2       2.5     0.833333
```

Here, it uses the scalar to perform division on all column elements in dataframe (axis = 1 by default).

Creating two frovedis dataframes to perform reverse division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)
```

```
# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Reverse division on two dataframes using operator version:

For example,

```
print(fdf2 / fdf1)
```

Output

```
index   points  total
0       0.4     0.7
1       0.5     0.818181
2       2       0.916666
```

Reverse division on two dataframes using method version:

For example,

```
# rdiv() demo on two dataframes using method version
fdf1.rdiv(other = fdf2).show()
```

Output

```
index   points  total
0       0.4     0.7
1       0.5     0.818181
```

```
2       2       0.916666
```

Here, only common columns in both dataframes are divided. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during reverse division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Reverse division on two dataframes and using fill_value parameter:

For example,

```
# rdiv() demo on two dataframes using method version and fill_value = 10
fdf1.rdiv(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       0.4     0.7
1       0.5     0.90909
2       2       1
```

Here, only common columns in both dataframes are divided, excluding the missing values. The fill_value = 10 is used to divide over column values in other dataframe (excluding the missing values) and stored in new dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.12 12. DataFrame.rfloordiv(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be divided with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform reverse division operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform reverse division operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs reverse floating division operation between two operands. It is equivalent to **'other // dataframe'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
```

```
# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Reverse floor division on a scalar value using operator version:

For example,

```
print(10 // fdf1)
```

Output

```
index   points  total
0       2       1
1       1       0
2       2       0
```

Reverse floor division on a scalar value using method version:

For example,

```
# rfloordiv() demo with scalar value using method version
fdf1.rfloordiv(10).show()
```

Output

```
index   points  total
0       2       1
1       1       0
2       2       0
```

Here, it uses the scalar to perform division on all column elements (axis = 1 by default).
Also, resultant dataframe column elements will contain floor integer value.

Creating two frovedis dataframes to perform reverse floor division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
```

```
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Reverse floor division on two dataframes using operator version:

For example,

```
print(fdf2 // fdf1)
```

Output

```
index   points  total
0       0       0
1       0       0
2       2       0
```

Reverse floor division on two dataframes using method version:

For example,

```
# rfloordiv() demo on two dataframes using method version
fdf1.rfloordiv(other = fdf2).show()
```

Output

```
index   points  total
0       0       0
1       0       0
2       2       0
```

Here, only common columns in both dataframes are divided. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during reverse floor division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Reverse floor division on two dataframes and using fill_value parameter:

For example,

```
# rfloordiv()  demo on two dataframes using method version and fill_value = 10
fdf1.rfloordiv(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       0       0
1       0       0
2       2       1
```

Here, only common columns in both dataframes are divided, excluding the missing values. The fill_value = 10 is used to divide over column values in other dataframe (excluding the missing values) and stored in new dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.13 13. DataFrame.rmod(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to perform modulo operation with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform reverse modulo operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform reverse modulo operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs reverse modulo operation between two operands. It is equivalent to **'other % dataframe'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [50, 40, 20]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Reverse modulo on a scalar value using operator version:

For example,

```
print(10 % fdf1)
```

Output

```
index   points  total
0       0       0
1       4       10
2       2       10
```

Reverse modulo on a scalar value using method version:

For example,

```
# rmod() demo with scalar value using method version
fdf1.rmod(10).show()
```

Output

```
index   points  total
0       0       0
1       4       10
2       2       10
```

Here, it uses the scalar to perform modulo operation on all column elements (axis = 1 by default).

Creating two frovedis dataframes to perform reverse modulo:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Reverse modulo on two dataframes using operator version:

For example,

```
print(fdf2 % fdf1)
```

Output

```
index   points  total
```

```
0       2       7
1       3       9
2       0       11
```

Reverse modulo on two dataframes using method version:

For example,

```
# rmod() demo on two dataframes using method version
fdf1.rmod(other = fdf2).show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       0       11
```

Here, modulo is performed on only common columns in both dataframes. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during reverse modulo:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)
```

```
# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Reverse modulo on two dataframes and using fill_value parameter:

For example,

```
# rmod() demo on two dataframes using method version and fill_value = 10
fdf1.rmod(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       2       7
1       3       10
2       0       0
```

Here, modulo is performed on only common columns in both dataframes, excluding the missing values. The fill_value = 10 is used to perform modulo over column values in other dataframe (excluding the missing values) and stored in new dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.14 14. DataFrame.rmul(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be multiplied with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform reverse multiplication operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform reverse multiplication operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs reverse multiplication operation between two operands. It is equivalent to **'other * dataframe'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Reverse multiplication on a scalar value using operator version:

For example,

```
print(10 * fdf1)
```

Output

```
index   points  total
0       50      100
1       60      110
2       40      120
```

Reverse multiplication on a scalar value using method version:

For example,

```
# rmul() demo with scalar value using method version
fdf1.rmul(10).show()
```

Output

```
index   points  total
0       50      100
```

```
1       60      110
2       40      120
```

Here, it uses the scalar to perform multiplication on all column elements in dataframe (axis = 1 by default).

Creating two frovedis dataframes to perform reverse multiplication:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Reverse multiplication on two dataframes using operator version:

For example,

```
print(fdf2 * fdf1)
```

Output

```
index   points  total
0       10      70
1       18      99
2       32      132
```

Reverse multiplication on two dataframes using method version:

For example,

```
# rmul() demo on two dataframes using method version
fdf1.rmul(other = fdf2).show()
```

Output

```
index   points  total
0       10      70
1       18      99
2       32      132
```

Here, only common columns in both dataframes are multiplied. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during reverse multiplication:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
```

```
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Reverse multiplication on two dataframes and using fill_value parameter:

For example,

```
# rmul() demo on two dataframes using method version and fill_value = 10
fdf1.rmul(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       10      70
1       18      110
2       32      100
```

Here, only common columns in both dataframes are multiplied, excluding the missing values. The fill_value = 10 is multiplied with column values in other dataframe (excluding the missing values) and stored in new dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.15 15. DataFrame.rpow(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to perfomr exponential power operation with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform reverse exponential power operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform reverse exponential power operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs reverse exponential power operation between two operands. It is equivalent to **'other ** dataframe'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Reverse exponential power operation on a scalar value using operator version:

For example,

```
print(10 ** fdf1)
```

Output

```
index   points  total
0       32      1024
1       64      2048
2       16      4096
```

Reverse exponential power operation on a scalar value using method version:

For example,

```
# rpow() demo with scalar value using method version
fdf1.rpow(2).show()
```

Output

```
index   points  total
0       32      1024
1       64      2048
2       16      4096
```

Here, it uses the scalar to perform exponential power operation on all column elements in dataframe (axis = 1 by default).

Creating two frovedis dataframes to perform reverse exponential power operation:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
```

```
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Reverse exponential power operation on two dataframes using operator version:

For example,

```
print(fdf2 ** fdf1)
```

Output

```
index   points  total
0       10      70
1       18      99
2       32      132
```

Reverse exponential power operation on two dataframes using method version:

For example,

```
# rpow() demo on two dataframes using method version
fdf1.rpow(other = fdf2).show()
```

Output

```
index   points  total
0       10      70
1       18      99
2       32      132
```

Here, exponential power operation is performed on only common columns in both dataframes. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during reverse exponential power operation:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Reverse exponential power operation on two dataframes and using fill_value parameter:

For example,

```
# rpow() demo on two dataframes using method version and fill_value = 10
fdf1.rpow(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       32      2.82475e+08
1       729     9.99999e+10
2       4096    1.00000e+10
```

Here, exponential power operation is performed on only common columns in both dataframes, excluding the missing values. The fill_value = 10 is used to perform exponential power operation on column values in other dataframe (excluding the missing values) and stored in new dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.16 16. DataFrame.rsub(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number

- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be subtracted with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform reverse subtraction operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform reverse subtraction operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs reverse subtraction operation between two operands. It is equivalent to **'other - dataframe'**.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

Reverse subtraction on a scalar value using operator version:

For example,

```
print(10 - fdf1)
```

Output

```
index   points  total
0       5       0
1       4       -1
2       6       -2
```

Reverse subtraction on a scalar value using method version:

For example,

```
# rsub() with scalar value using method version
fdf1.rsub(10).show()
```

Output

```
index   points  total
0       5       0
1       4       -1
2       6       -2
```

Here, it subtracts the scalar to all columns in dataframe (axis = 1 by default).

Creating two frovedis dataframes to perform reverse subtraction:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, 12]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       12
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, 9, 11]
        }
```

```
# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       9
2       8       11
```

Reverse subtraction on two dataframes using operator version:

For example,

```
print(fdf2 - fdf1)
```

Output

```
index   points  total
0       -3      -3
1       -3      -2
2       4       -1
```

Reverse subtraction on two dataframes using method version:

For example,

```
# rsub() on two dataframes using method version
fdf1.rsub(other = fdf2).show()
```

Output

```
index   points  total
0       -3      -3
1       -3      -2
2       4       -1
```

Here, only common columns in both dataframes are subtracted. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during reverse subtraction:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)
```

```
# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Reverse subtraction on two dataframes and using fill_value parameter:

For example,

```
# rsub() on two dataframes using method version and fill_value = 10
fdf1.rsub(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       -3      -3
1       -3      -1
2       4       0
```

Here, only common columns in both dataframes are subtracted, excluding the missing values. The fill_value = 10 is subtracted on column values in other dataframe (excluding the missing values) and stored in new dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.1.3.17 17. DataFrame.rtruediv(other, axis = 'columns', level = None, fill_value = None)

Parameters
other: It can accept single or multiple element data structure like the following:

- Number
- List having 1 dimension. Currently, this method supports operation on only list of numeric values.
- A numpy ndarray having 1 dimension. Currently, this method supports operation on only an array of numeric values.
- pandas DataFrame. It must not be an empty dataframe.
- pandas Series
- frovedis DataFrame. It must not be an empty dataframe.

Any of these is considered as the value to be divided with the current dataframe.
axis: It accepts an integer or string object as parameter. It is used to decide whether to perform reverse division operation along the indices or by column labels. (Default: 'columns')

- **1 or 'columns'**: perform reverse division operation on the columns. Currently, axis = 1 is supported in this method.

level: This is an unused parameter. (Default: None)
fill_value: It accepts scalar values or None. It fills existing missing (NaN) values, and any new element needed for successful dataframe alignment, with this value before computation. (Default: None)
Irrespective of the specified value, if data in both corresponding dataframe locations is missing, then the result will be missing (contains NaNs).

Purpose
It performs reverse floating division operation between two operands. It is equivalent to **'other / dataframe'**.

Currently, it does not perform reverse division of scalar using operator version. Only method version is supported.

Creating frovedis DataFrame from pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [8, 5, 9],
         "total": [3, 2, 1]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       8       3
1       5       2
2       9       1
```

Reverse floating division on a scalar value using operator version:

For example,

```
print(10 / fdf1)
```

Output

```
index   points  total
0       1.25    3.33333
1       2       5
2       1.11111 10
```

Reverse floating division on a scalar value using method version:

For example,

```
# rtruediv() demo with scalar value using method version
fdf1.rtruediv(10).show()
```

Output

```
index   points  total
0       1.25    3.33333
1       2       5
2       1.11111 10
```

Here, it uses the scalar to perform division on all column elements in dataframe (axis = 1 by default).

Creating two frovedis dataframes to perform reverse floating division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
data1 = {
         "points": [8, 5, 9],
         "total": [3, 2, 1]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       8       3
```

```
1       5       2
2       9       1
```

For example,

```
# a dictionary
data2 = {
         "points": [4, 7, 2],
         "total": [9, 1, 9]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       4       9
1       7       1
2       2       9
```

Reverse floating division on two dataframes using operator version:

For example,

```
print(fdf2 / fdf1)
```

Output

```
index   points    total
0       0.5       3
1       1.39999   0.5
2       0.222222  9
```

Reverse floating division on two dataframes using method version:

For example,

```
# rtruediv() demo on two dataframes using method version
fdf1.rtruediv(other = fdf2).show()
```

Output

```
index   points    total
0       0.5       3
1       1.39999   0.5
2       0.222222  9
```

Here, only common columns in both dataframes are divided. Other are replaced with NaN values in resultant dataframe (fill_value = None by default).

Creating two frovedis dataframes and use fill_value parameter during reverse floating division:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf
```

```
# a dictionary
data1 = {
         "points": [5, 6, 4],
         "total": [10, 11, np.nan]
        }

# create pandas dataframe
pdf1 = pd.DataFrame(data1)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   points  total
0       5       10
1       6       11
2       4       NULL
```

For example,

```
# a dictionary
data2 = {
         "points": [2, 3, 8],
         "total": [7, np.nan, np.nan]
        }

# create pandas dataframe
pdf2 = pd.DataFrame(data2)

# create frovedis dataframe
fdf2 = fdf.DataFrame(pdf2)

# display the frovedis dataframe
fdf2.show()
```

Output

```
index   points  total
0       2       7
1       3       NULL
2       8       NULL
```

Reverse floating division on two dataframes and using fill_value parameter:

For example,

```
# rtruediv() demo on two dataframes using method version and fill_value = 10
fdf1.rtruediv(other = fdf2, fill_value = 10).show()
```

Output

```
index   points  total
0       0.4     0.7
1       0.5     0.90909
```

```
2       2       1
```

Here, only common columns in both dataframes are divided, excluding the missing values. The fill_value = 10 is divided with column values in other dataframe (excluding the missing values) and stored in new dataframe.

Return Value
It returns a frovedis DataFrame which contains the result of arithmetic operation.

58.2 SEE ALSO

- **DataFrame - Introduction**
- **DataFrame - Indexing Operations**
- **DataFrame - Generic Functions**
- **DataFrame - Conversion Functions**
- **DataFrame - Sorting Functions**
- **DataFrame - Aggregate Functions**

Chapter 59

FrovedisGroupedDataframe

59.1 NAME

FrovedisGroupedDataframe - A python class for handling grouped dataframes. These are returned by groupby calls.

59.2 SYNOPSIS

```
frovedis.dataframe.grouped_df.FrovedisGroupedDataframe(df = None)
```

59.3 DESCRIPTION

FrovedisGroupedDataframe instance contains information about the grouped dataframe. After columns are grouped, various aggregations can be performed like groupwise average, groupwise variance, etc.
In FrovedisGroupedDataframe, currently aggregation operations is performed along the rows only.

This module provides a client-server implementation, where the client application is a normal python program. The FrovedisGroupedDataframe interface is almost same as pandas DataFrameGroupBy interface, but it doesn't have any dependency on pandas. It can be used simply even if the system doesn't have pandas installed. The FrovedisGroupedDataframe instance is created when groupby interface is called for frovedis Dataframe instance. Thus, in this implementation, a python client can interact with a frovedis server sending the required python dataframe for performing query at frovedis side. Python dataframe is converted into frovedis compatible dataframe internally and the python client side call is linked with the respective frovedis side call to get the job done at frovedis server.

59.3.1 Detailed description

59.3.1.1 1. FrovedisGroupedDataframe(df = None)

Parameters
df: It accepts only frovedis dataframe as parameter. (Default: None)
When it is None (not specified explicitly), an empty FrovedisGroupedDataframe instance is created.

Purpose
It is used for a specific purpose. It is instance is created in order to hold result of groupby method calls. This instance can then further be used with aggregate functions such as mean(), sem(), etc.

Creating FrovedisGroupedDataframe instance using groupby operation:

For example,

```
# create frovedis dataframe
import pandas as pd
import frovedis.dataframe as fdf
peopleDF = {
            'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
            'Age' : [29, 30, 27, 19, 31],
            'Country' : ['USA', 'England', 'Japan', 'France', 'Japan'],
            'isMale': [False, False, False, False, True]
           }
pdf1 = pd.DataFrame(peopleDF)
fdf1 = fdf.DataFrame(pdf1)

# create FrovedisGroupedDataframe object
g_df = fdf1.groupby('Country')
```

The groupby call returns a FrovedisGroupedDataframe instance.

Multiple columns can be used in groupby operation to create this instance:

For example,

```
# using multiple columns to create FrovedisGroupedDataframe object
g_df = fdf1.groupby(['Age','Country'])
```

In order to select one of the grouped column:

For example,

```
print(g_df['Country'])
```

Output

```
Country
England
France
Japan
USA
```

In order to select multiple columns from the grouped dataframe:

For example,

```
print(g_df[['Age','Country']])
```

Output

```
index   Age     Country
0       19      France
1       27      Japan
2       29      USA
3       30      England
4       31      Japan
```

Return Value
It simply returns “self” reference.

59.3.1.2 2. release()

Purpose
This method acts like a destructor. It is used to release dataframe pointer from server heap and it resets all its attributes to None.

For example,

```
g_df.release()
```

Return Value
It returns nothing.

59.3.2 Public Member Functions

FrovedisGroupedDataFrame provides a lot of utilities to perform various operations.

59.3.2.1 List of Aggregate Functions

1. **agg()** - it agggregates using one or more operations over the specified axis. It is an alias for aggregate().
2. **aggregate()** - it agggregates using one or more operations over the specified axis. The alias agg() can be used instead.
3. **count()** - it computes count of group, excluding missing values.
4. **max()** - it computes maximum of group values.
5. **mean()** - it computes mean of groups, excluding missing values.
6. **min()** - it computes minimum of group values.
7. **sem()** - it computes standard error of the mean of groups, excluding missing values.
8. **size()** - it computes group sizes.
9. **sum()** - it computes sum of group values.
10. **var()** - it computes variance of groups, excluding missing values.

59.4 SEE ALSO

- **Using aggregate functions on FrovedisGroupedDataFrame**
- **Introduction to frovedis DataFrame**

Chapter 60

FrovedisGroupedDataFrame Aggregate Functions

60.1 NAME

FrovedisGroupedDataFrame Aggregate Functions - aggregate operations performed on grouped dataframe are being illustrated here.

60.1.1 DESCRIPTION

Once FrovedisGroupedDataframe instance is created, several aggregation operations can be performed on it such as max(), min(), sem(), var(), etc.

The aggregation operation will basically compute summary for each group. Some examples:

- Compute group sums, means, var or sem.
- Compute group sizes / counts.
- Compute max or min in the group.

Also, aggregation functions can be chained along with groupby() calls in frovedis.

60.1.2 Public Member Functions

```
1. agg(func, *args, **kwargs)
2. aggregate(func, \*args, \*\*kwargs)
3. count(numeric_only = True)
4. groupby()
5. max(numeric_only = True, min_count = -1)
6. mean(numeric_only = True)
7. min(numeric_only = True, min_count = -1)
8. sem(ddof = 1.0)
9. size(numeric_only = True)
10. sum(numeric_only = True, min_count = 0)
11. var(ddof = 1.0)
```

60.1.3 Detailed Description

60.1.3.1 1. FrovedisGroupedDataFrame.agg(func, *args, **kwargs)

Parameters
func: Names of functions to use for aggregating the data. The input to be used with the function must be a frovedis DataFrame instance having atleast one numeric column.
Accepted combinations for this parameter are: - A string function name such as 'max', 'min', etc.
- list of functions and/or function names, For example, ['max', 'mean'].
- dictionary with keys as column labels and values as function name or list of such functions.
For Example, {'Age': ['max','min','mean'], 'Ename': ['count']}

****args***: This is an unused parameter.
*****kwargs***: Additional keyword arguments to be passed to the function.

Purpose
It computes an aggregate operation based on the condition specified in 'func'. It is an alias for aggregate().

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Aggregate the function on the grouped dataframe where func is a function string name:

For example,

```
# agg() demo with func as a function string name
fdf1.groupby('Qualification').agg('min').show()
```

Output

```
Qualification   min_Age min_Score
B.Tech          22      23
Phd             24      34
```

It displays a frovedis dataframe containing numeric column(s) with newly computed minimum of each groups.

Aggregate the function on the grouped dataframe where func is a dictionary:

For example,

```
# agg() demo with func as a dictionary
fdf1.groupby('Qualification').agg({'Age': ['max','min','mean'], 'Score': ['sum']}).show()
```

Output

```
Qualification   max_Age min_Age mean_Age  sum_Score
B.Tech          36      22      29.25     108
Phd             33      24      29        13
```

Aggregate the function on the grouped dataframe where func is a list of functions:

For example,

```
# agg() demo where func is a list of functions
fdf1.groupby('Qualification').agg(['min', 'max']).show()
```

Output

```
Qualification   min_Age max_Age min_Score  max_Score
B.Tech          22      36      23         50
Phd             24      33      34         52
```

Using keyword arguments in order to perform aggregation operation:

For example,

```
# agg() demo where **kwargs are provided
fdf1.groupby("Qualification").agg(sum_a = ("Age", "sum"), min_b = ("Score", "min")).show()
```

Output

```
Qualification   sum_a   min_b
B.Tech          117     23
Phd             116     34
```

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.2 2. FrovedisGroupedDataFrame.aggregate(func, *args, **kwargs)

Parameters
func: Names of functions to use for aggregating the data. The input to be used with the function must be a frovedis DataFrame instance having atleast one numeric column.
Accepted combinations for this parameter are: - A string function name such as 'max', 'min', etc.
- list of functions and/or function names, For example, ['max', 'mean'].

- dictionary with keys as column labels and values as function name or list of such functions.
For Example, {'Age': ['max','min','mean'], 'Ename': ['count']}

***args**: This is an unused parameter.
****kwargs**: This is an unused parameter.

Purpose
It computes an aggregate operation based on the condition specified in 'func'.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }


# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Aggregate the function on the grouped dataframe where func is a function string name:

For example,

```
# aggregate() demo using FrovedisGroupedDataframe instance and 'func' as a function string name
fdf1.groupby('Qualification').agg('max').show()
```

Output

```
Qualification   max_Age max_Score
B.Tech          36      50
Phd             33      52
```

It displays a frovedis dataframe containing numeric column(s) with newly computed minimum of each groups.

Aggregate the function on the grouped dataframe where func is a dictionary:

For example,

```
# aggregate() demo using FrovedisGroupedDataframe instance and 'func' as a dictionary
fdf1.groupby('Qualification').agg({'Age': ['count'], 'Score': ['max','min']}).show()
```

Output

```
Qualification   count_Age  max_Score  min_Score
B.Tech          4          50         23
Phd             4          52         34
```

Aggregate the function on the grouped dataframe where func is a list of functions:

For example,

```
# aggregate() demo using FrovedisGroupedDataframe instance and 'func' is a list of functions
fdf1.groupby('Qualification').agg(['mean','sum']).show()
```

Output

```
Qualification   mean_Age  sum_Age mean_Score  sum_Score
B.Tech          29.25     117     36          108
Phd             29        116     43.6666     131
```

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.3 3. FrovedisGroupedDataFrame.count(numeric_only = True)

Parameters
numeric_only: It accepts a boolean parameter that specifies whether or not to use only numeric column data as input. (Default: True)

Purpose
It computes count of groups, excluding the missing values.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
```

```
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Count groups in each column of the grouped dataframe:

For example,

```
# count() demo using FrovedisGroupedDataframe instance
fdf1.groupby('Qualification').count().show()
```

Output

```
Qualification   count_Age   count_Score  count_Name  count_City
B.Tech          4           3            4           4
Phd             4           3            4           4
```

It displays a frovedis dataframe containing the newly computed count of each groups.

Also, it excludes the missing value(s) in **'Score'** column while computing count of groups **'B.Tech'** and **'Phd'**.

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.4 4. DataFrame.groupby(by = None, axis = 0, level = None, as_index = True, sort = True, group_keys = True, squeeze = False, observed = False, dropna = True)

Parameters
by: It accepts a string object or an iterable to determine the groups on which groupby operation will be applied. Currently, gropuby operation will be applied along the index levels. It must be provided, otherwise it will raise an exception. (Default: None)
axis: It accepts an integer as parameter. It is used to decide whether to perform groupby operation along the indices or by column labels. (Default: 0)
Currently, axis = 0 is supported by this method.
level: This is an unused parameter. (Default: None)
as_index: This is an unused parameter. (Default: True)
sort: This is an unused parameter. (Default: True)
group_keys: This is an unused parameter. (Default: True)
squeeze: This is an unused parameter. (Default: False)
observed: This is an unused parameter. (Default: False)
dropna: It accepts a boolean parameter. It is used to remove missing values (NaNs) from the frovedis DataFrame during groupby operation. Currently, it removes missing values along the index levels. (Default: True)

Purpose
This method can be used to group large amounts of data and compute operations on these groups.

The parameters: “level”, “as_index”, “sort”, “group_keys”, “squeeze” and “observed” is simply kept in to make the interface uniform to the pandas DataFrame.groupby(). This is not used anywhere within the frovedis implementation.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import numpy as np
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name': ['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age': [27, 24, 22, 32, 33, 36, 27, 32],
            'City': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                     'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification': ['B.Tech', 'Phd', 'B.Tech', np.nan, 'Phd', 'B.Tech', 'Phd', np.nan],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
           }

# create pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# create frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# display the frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age   City       Qualification  Score
0       Jai     27    Nagpur     B.Tech         23
1       Anuj    24    Kanpur     Phd            34
2       Jai     22    Allahabad  B.Tech         35
3       Princi  32    Kannuaj    NULL           45
4       Gaurav  33    Allahabad  Phd            NULL
5       Anuj    36    Kanpur     B.Tech         50
6       Princi  27    Kanpur     Phd            52
7       Abhi    32    Kanpur     NULL           NULL
```

Performing groupby operation on the frovedis dataframe:

For example,

```
fdf1.groupby('Qualification')
```

This will perform groupby operation on the dataframe over **‘Qualification’** column data.

Using groupby() to perform aggregation on resultant grouped dataframe. Also dropna = True:

For example,

```
fdf1.groupby('Qualification', dropna = True).agg({'Score': 'count'})
```

Output

```
Qualification   count_Score
B.Tech  3
Phd     2
```

Here, it excludes **NULL** group since missing values were dropped during groupby().

Using groupby() to perform aggregation on resultant grouped dataframe. Also dropna = False:

For example,

```
fdf1.groupby('Qualification', dropna = False).agg({'Score': 'count'})
```

Output

```
Qualification   count_Score
B.Tech  3
Phd     2
NULL    1
```

Here, it includes **NULL** as new group since missing values were not dropped during groupby().

Return Value
It returns a FrovedisGroupedDataFrame instance. This instance is then further used to perform aggregate operations.

60.1.3.5 5. DataFrame.max(numeric_only = True, min_count = -1)

Parameters
numeric_only: It accepts a boolean parameter that specifies whether or not to use only numeric columns for aggregation. (Default: True)
min_count: This is an unused parameter. (Default: -1)

Purpose
It computes the maximum of group values.

The parameter: "min_count" is simply kept in to to make the interface uniform to the pandas GroupBy.max(). This is not used anywhere within the frovedis implementation.

Currently, this method only displays result for dataframe having atleast one numeric column.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }


# convert to pandas dataframe
```

```
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

To calculate maximum value in each group of grouped dataframe:

For example,

```
# max() demo using FrovedisGroupedDataframe instance
fdf1.groupby('Qualification').max().show()
```

Output

```
Qualification   max_Age max_Score
B.Tech          36      50
Phd             33      52
```

It displays a frovedis dataframe containing numeric column(s) with newly computed maximum of each groups.

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.6 6. FrovedisGroupedDataFrame.mean(numeric_only = True)

Parameters
numeric_only: It accepts a boolean parameter that specifies whether or not to use only numeric column data as input. (Default: True)

Purpose
It computes mean of groups, excluding the missing values.

Currently, this method only displays result for dataframe having atleast one numeric column.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
```

```
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Compute mean of each group:

For example,

```
# mean() demo using FrovedisGroupedDataframe instance
fdf1.groupby('Qualification').mean().show()
```

Output

```
Qualification   mean_Age  mean_Score
B.Tech          29.25     36
Phd             29        43.6666
```

It displays a frovedis dataframe containing numeric column(s) with newly computed mean of each groups.

Also, it excludes the missing value(s) in **'Score'** column while computing mean of groups **'B.Tech'** and **'Phd'**.

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.7 7. FrovedisGroupedDataFrame.min(numeric_only = True, min_count = -1)

Parameters
numeric_only: It accepts a boolean parameter that specifies whether or not to use only numeric columns for aggregation. (Default: True)
min_count: This is an unused parameter. (Default: -1)

Purpose
It computes the minimum of group values.

The parameter: "min_count" is simply kept in to to make the interface uniform to the pandas GroupBy.min(). This is not used anywhere within the frovedis implementation.

Currently, this method only displays result for dataframe having atleast one numeric column.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

To calculate minimum value in each group of grouped dataframe:

For example,

```
# min() demo using FrovedisGroupedDataframe instance
fdf1.groupby('Qualification').min().show()
```

Output

```
Qualification   min_Age min_Score
B.Tech          22      23
Phd             24      34
```

It displays a frovedis dataframe containing numeric column(s) with newly computed minimum of each groups.

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.8 8. FrovedisGroupedDataFrame.sem(ddof = 1.0)

Parameters
ddof: It accepts an integer parameter that specifies the delta degrees of freedom. (Default: 1.0)

Purpose
It computes standard error of the mean of groups, excluding missing values.

Currently, this method only displays result for dataframe having atleast one numeric column.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

Compute standard error of the mean of each group:

For example,

```
# sem() demo using FrovedisGroupedDataframe instance and default ddof value
fdf1.groupby('Qualification').sem().show()
```

Output

```
Qualification   sem_Age  sem_Score
B.Tech          3.03795  7.81024
Phd             2.12132  5.23874
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed standard error of mean for each groups.

Also, it excludes the missing value(s) in **'Score'** column while computing standard error of mean for the groups **'B.Tech'** and **'Phd'**.

Using ddof parameter to compute standard error of the mean of each group:

For example,

```
# sem() demo using ddof = 2
fdf1.groupby('Qualification').sem(ddof = 2).show()
```

Output

```
Qualification   sem_Age  sem_Score
B.Tech          3.72071  11.0453
Phd             2.59807  7.4087
```

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.9 9. FrovedisGroupedDataFrame.size(numeric_only = True)

Parameters
numeric_only: It accepts a boolean parameter that specifies whether or not to use only numeric column data as input. (Default: True)

Purpose
It computes group sizes, including the missing values.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

To compute size of groups for the grouped dataframe:

For example,

```
# size() demo using FrovedisGroupedDataframe instance
fdf1.groupby('Qualification').size().show()
```

Output

```
Qualification   size_Qualification
B.Tech          4
Phd             4
```

It displays a frovedis dataframe containing a column with the newly computed size of each group.

Also, it does not exclude the missings values while computing group size.

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.10 10. FrovedisGroupedDataFrame.sum(numeric_only = True, min_count = 0)

Parameters
numeric_only: It accepts a boolean parameter that specifies whether or not to use only numeric column data as input. (Default: True)
min_count: This is an unused parameter. (Default: 0)

Purpose
It computes the sum of group values.

The parameter: "min_count" is simply kept in to to make the interface uniform to the pandas GroupBy.sum(). This is not used anywhere within the frovedis implementation.

Currently, this method only displays result for dataframe having atleast one numeric column.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
```

```
            }

# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

To compute sum of group values for the grouped dataframe:

For example,

```
# sum() demo using FrovedisGroupedDataframe instance
fdf1.groupby('Qualification').sum().show()
```

Output

```
Qualification   sum_Age sum_Score
B.Tech          117     108
Phd             116     131
```

It displays a frovedis dataframe containing numeric column(s) with newly computed sum of each groups.

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.1.3.11 11. FrovedisGroupedDataFrame.var(ddof = 1.0)

Parameters
ddof: It accepts an integer parameter that specifies the delta degrees of freedom. (Default: 1.0)

Purpose
It computes the variance of groups, excluding missing values.

Currently, this method only displays result for dataframe having atleast one numeric column.

Constructing frovedis DataFrame from a pandas DataFrame:

For example,

```
import pandas as pd
import frovedis.dataframe as fdf

# a dictionary
peopleDF = {
```

```
            'Name':['Jai', 'Anuj', 'Jai', 'Princi', 'Gaurav', 'Anuj', 'Princi', 'Abhi'],
            'Age':[27, 24, 22, 32, 33, 36, 27, 32],
            'City':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Allahabad',
                    'Kanpur', 'Kanpur', 'Kanpur'],
            'Qualification':['B.Tech', 'Phd', 'B.Tech', 'Phd', 'Phd', 'B.Tech', 'Phd', 'B.Tech'],
            'Score': [23, 34, 35, 45, np.nan, 50, 52, np.nan]
            }

# convert to pandas dataframe
pdf1 = pd.DataFrame(peopleDF)

# convert to frovedis dataframe
fdf1 = fdf.DataFrame(pdf1)

# displaying created frovedis dataframe
fdf1.show()
```

Output

```
index   Name    Age     City       Qualification  Score
0       Jai     27      Nagpur     B.Tech         23
1       Anuj    24      Kanpur     Phd            34
2       Jai     22      Allahabad  B.Tech         35
3       Princi  32      Kannuaj    Phd            45
4       Gaurav  33      Allahabad  Phd            NULL
5       Anuj    36      Kanpur     B.Tech         50
6       Princi  27      Kanpur     Phd            52
7       Abhi    32      Kanpur     B.Tech         NULL
```

To compute variance of groups in the grouped dataframe:

For example,

```
# var() demo using FrovedisGroupedDataframe instance and default ddof value
fdf1.groupby('Qualification').var().show()
```

Output

```
Qualification   var_Age  var_Score
B.Tech          36.9166  183
Phd             18       82.3333
```

It displays a frovedis dataframe with numeric column(s) containing the newly computed variance for each groups.

Also, it excludes the missing value in **'Score'** column while computing variance of groups **'B.Tech'** and **'Phd'**.

Using ddof parameter to compute variance of each group:

For example,

```
# var() demo using ddof = 2
fdf1.groupby('Qualification').var(ddof = 2).show()
```

Output

```
Qualification   var_Age var_Score
B.Tech          55.375  366
Phd             27      164.666
```

Return Value
It returns a new frovedis DataFrame instance with the result of the specified aggregate functions.

60.2 SEE ALSO

- **Introduction to FrovedisGroupedDataFrame**
- **Introduction to frovedis DataFrame**