

Tutorial of Frovedis Python Interface

1. Introduction

This document is a tutorial of Frovedis Python interface.

Frovedis is a MPI library that provides

- Matrix library using above API
- Machine learning algorithm library
- Dataframe for preprocessing

The Python interface wraps these functionalities can make it possible to call from Python script. Since the library is optimized for SX-Aurora TSUBASA, you can utilize vector architecture without being aware of it. You can use it also on x86 servers.

It is implemented by using a server program. An MPI program with Frovedis functionalities (frovedis_server) is invoked and the Python interpreter communicates with it.

2. Environment setting

In this tutorial, we assume that Frovedis is installed from rpm. Please follow /opt/nec/nosupport/frovedis/getting_started.md. As described in the file, if you want to use frovedis_server on x86, please do:

```
$ source /opt/nec/nosupport/frovedis/x86/bin/x86env.sh
```

If you want to use vector engine, please do:

```
$ source /opt/nec/nosupport/frovedis/ve/bin/veenv.sh
```

If you use vector engine, please make sure that MPI for vector engine is set up. Usually, you need to do:

```
$ source /opt/nec/ve/mpi/[YOUR_MPI_VERSION]/bin/necmpivars.sh
```

Main purpose of the script is to set PYTHONPATH and LD_LIBRARY_PATH. It also switches mpirun to call (x86 or ve).

Supported Python version is 2.7, which is installed in CentOS/RedHat by default. Since our wrapper is just a Python library and shared library, you can use tools like virtualenv, Jupyter, etc. together with the wrapper.

In this tutorial, we use python with virtualenv, because scikit-learn cannot be installed by yum, and using pip for system installed Python is a bit dangerous (virtualenv and pip will be installed together with Frovedis by yum).

Please create your environment by virtualenv and install scikit-learn:

```
$ virtualenv frovedis_tutorial
$ source frovedis_tutorial/bin/activate
(frovedis_tutorial) $ pip install scikit-learn
```

Installing scikit-learn is for tutorial purpose. If you want to use only Frovedis, you do not have to install scikit-learn.

If you want to run the tutorials on jupyter-notebook in the virtual environment, you might need to run following:

```
(frovedis_tutorial) $ pip install jupyter ipython ipykernel
(frovedis_tutorial) $ ipython kernel install --user --name=frovedis_tutorial
```

If you run jupyter notebook server on a server machine and run your browser on a client machine, following setting would need to be added in your `~/jupyter/jupyter_notebook_config.py`

```
c = get_config()
c.NotebookApp.ip = '0.0.0.0'
c.NotebookApp.open_browser = False
c.NotebookApp.notebook_dir = '/path/to/save/notebook'
```

Then, you can run

```
(frovedis_tutorial) $ jupyter-notebook
```

and access the server with the token printed by the command. Please change kernel to `frovedis_tutorial` at the kernel tab.

In addition, please copy the `src` directory to somewhere you have write permission, because it will create files.

3. Simple example

Please look at “`src/tut3/tut.py`”. It loads “breast cancer” data from scikit-learn, and run logistic regression on the data.

Lines with trailing `# frovedis` is specific for Frovedis. Lines with trailing `# sklearn` is for scikit-learn instead.

To use Frovedis, you need to import `FrovedisServer`:

```
from frovedis.exrpc.server import FrovedisServer
```

Then, import `LogisticRegression` in this case:

```
from frovedis.ml.lib.linear_model import LogisticRegression
```

In the case of scikit-learn, following module is imported instead:

```
from sklearn.linear_model import LogisticRegression
```

Since Frovedis only accepts `np.float64` type of `{-1, 1}` as label, loaded value of `y` that contains `{0, 1}` is converted to `np.float64` and modified as `y = 2 * y - 1`.

Before using the logistic regression routine, you need to invoke `frovedis_server`:

```
FrovedisServer.initialize("mpirun -np 4 {}".format(os.environ['FROVEDIS_SERVER']))
```

You need to specify the command to invoke the server as the argument of `initialize`. Since the server is an MPI program, `mpirun` is used here. The option `-np` is for specifying the number of MPI processes. Here, 4 processes will be used. You can use multiple cards (in the case of vector engine) and/or multiple servers by specifying command line option appropriately.

The last argument of `mpirun` is the binary to execute. Here, the path of the binary is obtained from the environment variable `FROVEDIS_SERVER`, which is set in `x86env.sh` or `veenv.sh`.

The `LogisticRegression` call is the same as `scikit-learn`. Within the call, the data in Python interpreter is sent to `frovedis_server` and the machine learning algorithm is executed there.

After executing the machine learning algorithm, please shutdown the server:

```
FrovedisServer.shut_down()
```

As you can see, what you need to do is changing the importing module and add `initialize` / `shutdown` the server.

You can run the sample by

```
(frovedis_tutorial) $ python tut.py  
score: 0.922671353251
```

Even if you change the import to use `scikit-learn`, it should produce similar result.

In this case, the speed of training of Frovedis is actually slower than `scikit-learn`. This is because the size of the data is very small (569, 30).

The `frovedis` server will be terminated when the interpreter exits. If it is not terminated because of abnormal termination, please kill the server manually by calling command like `pkill mpi`.

4. Machine learning algorithms

At this moment, we support following algorithms:

- `linear_model.LogisticRegression`
- `linear_model.LinearRegression`
- `linear_model.Lasso`
- `linear_model.Ridge`
- `svm.LinearSVC`
- `cluster.KMeans`
- `tree.DecisionTreeRegressor`
- `tree.DecisionTreeClassifier`
- `naive_bayes.MultinomialNB`
- `naive_bayes.BernoulliNB`
- `decomposition.TruncatedSVD`

Please add `frovedis.mllib.` to import these modules. (In the case of scikit-learn, `sklearn.` is added to import them.) The interface is almost the same as scikit-learn.

Other than scikit-learn algorithms, we support following algorithms.

- `fm.FactorizationMachineClassifier`
- `recommendation.ALS`

You can use both dense and sparse matrix as the input of machine learning just like scikit-learn. It is automatically sent to Frovedis server, and automatically distributed among MPI processes. (SX-Aurora TSUBASA shows much better performance with sparse matrix.)

For more information, please refer to the manual. You can also find other samples in `/opt/nec/nosupport/frovedis/x86/foreign_if_demo/python/`.

5. Distributed matrix

As we mentioned, you can use variable of Python side directly as the input of machine learning algorithms that works on Frovedis server. In addition, you can also use the distributed matrix and vector at Frovedis server explicitly, which can be used as input of the machine learning algorithms.

Since you can keep the data at Frovedis server side, you can reduce the communication cost of sending data from Python to the server if you reuse the data.

Please look at “src/tut5-1/tut.py”. It creates sparse matrix at the Frovedis server side from scipy csr matrix.

```
mat = csr_matrix((data, indices, indptr),
                 dtype=np.float64,
                 shape=(3, 3))
```

Here, `mat` is scipy's csr format of sparse matrix. (in Frovedis, it is called as *crs* format.) Then, `FrovedisServer.initialize` is called. This time, `-np` is 2. After that,

```
fmat = FrovedisCRSMatrix(mat)
```

creates crs matrix at Frovedis server. To check if it is really created, `debug_print()` is called. It should print like:

```
matrix:
num_row = 3, num_col = 3
node 0
local_num_row = 2, local_num_col = 3
val : 1 2 3
idx : 0 2 2
off : 0 2 3
node 1
local_num_row = 1, local_num_col = 3
val : 4 5 6
idx : 0 1 2
off : 0 3
```

It is printed at the server side. It shows that first 2 rows are in the node 0 and third row is in the node 1.

The data at Frovedis server is saved by `fmat.save("./result")`. The contents of this file should look like:

```
0:1 2:2
2:3
0:4 1:5 2:6
```

Each item is separated by space, and each row is separated as line. Each item is like “POS:VAL”; POS is 0-based column position. This is the sparse matrix text file format of Frovedis.

The memory of the server side is released when the variable `fmat` is garbage collected. But you can explicitly release it by calling `fmat.release()`.

You can create sparse matrix by loading from a file.

```
fmat2 = FrovedisCRSMatrix().load_text("./result")
```

creates a new matrix from the saved data. `fmat2.debug_print()` should produce the same output as the above.

In this case, we used text file format, but you can also use binary file format by using `save_binary` and `load_binary`. It should be much faster than text format on vector engine. Please refer to the C++ tutorial for binary format.

The file “src/tut5-2/tut.py” is dense matrix version. In this case, `FrovedisRowmajorMatrix` is created from `numpy.matrix`. You can try `FrovedisColmajorMatrix` version that is written as comment. Here, `debug_print()` shows internal data. If you want to see the data as row major way, use `get_rowmajor_view()` instead.

The text format of rowmajor matrix is like:

```
1 2 3 4
5 6 7 8
8 7 6 5
4 3 2 1
```

If you use data at Frovedis server side as the input of machine learning algorithms, you need to be aware of the type; for example, `LogisticRegression` takes `FrovedisColmajorMatrix`, but does not take `FrovedisRowmajorMatrix`. Please refer to the manuals for more details

Label of the machine learning algorithms is a vector, and you can also use the distributed vector at Frovedis server explicitly. The file “src/tut5.3/tut.py” shows how to create it.

```
dv = FrovedisDvector([1,2,3,4,5,6,7,8],dtype=np.float64)
dv.debug_print()
```

The `debug_print()` should print like this:

```
dvector(size: 8):
 1 2 3 4 5 6 7 8
```

So far, we explained sparse matrix (`FrovedisCRSMatrix`), dense matrix (`FrovedisRowmajorMatrix`, `FrovedisColmajorMatrix`), and distributed vector (`FrovedisDvector`). We also another kind of distributed dense matrix called `FrovedisBlockcyclicMatrix`.

`FrovedisBlockcyclicMatrix` supports distributed matrix operations that is backed by ScaLAPACK/PBLAS. It can be utilized for large scale matrix operations. Please see “src/tut5-4/tut.py”. It contains examples of various PBLAS functionalities.

First, input numpy matrices `x`, `y`, `m`, and `n` are created. Frovedis server side block cyclic matrix can be created like:

```
bcx = FrovedisBlockcyclicMatrix(x)
```

In ScaLAPACK/PBLAS, vectors are represented as one dimensional matrix.

First example swaps two vectors by `PBLAS.swap(bcx,bcy)`. To check if they are swapped, you can call `debug_print()` of these variables. However in this example, the blockcyclic matrix is copied back to Python interpreter and converted to numpy matrix by `to_numpy_matrix()` and printed.

Next example is multiplying by scalar: `PBLAS.scal(bcx,2)`. As you see, PBLAS interface overwrites the original matrix.

`PBLAS.axpy(bcx,bcy,2)` does $y = ax + y$, here a is 2. `PBLAS.copy(bcx,bcy)` copies the matrix ($y = x$).

`PBLAS.dot(bcx,bcy)` calculates dot product of x and y . Here, you can use numpy matrix x and y instead of bcx and bcy . In this case, blockcyclic matrix is created automatically. Other operations like `nrm2`, `gemv`, `ger`, `gemm`, and `geadd` also take numpy matrix as input.

`PBLAS.nrm2(bcx)` calculates L2 norm of the vector.

`PBLAS.gemv(bcm,bcx)` calculates matrix vector multiplication ($m * x$). The result is newly created blockcyclic matrix (vector).

`PBLAS.gemm(bcm,bcn)` does matrix-matrix multiplication ($m * n$). The result is also newly created blockcyclic matrix.

`PBLAS.geadd(bcm,bcn)` does matrix addition like $n = m + n$.

Lastly, you can explicitly release the blockcyclic matrix by calling `release()`, though they are automatically released when the variable is garbage collected.

Next, we will explain ScaLAPACK functionalities. Please see “src/tut5-5/tut.py”.

This time, `FrovedisBlockcyclicMatrix` is created by loading from a file.

```
bcm = FrovedisBlockcyclicMatrix(dtype=np.float64)
bcm.load("./input")
```

`FrovedisBlockcyclicMatrix` can be saved by `save`, and binary format can also be used by `load_binary` and `save_binary`. To save the matrix, it is converted to Python numpy matrix.

```
m = bcm.to_numpy_matrix()
```

First example is `getrf`, which does LU factorization.

```
rf = SCALAPACK.getrf(bcm)
```

The argument matrix is overwritten to factorized matrix. The return value contains pivoting information (`ipiv`), which is needed to use the factorized matrix later.

Next, by using the factorized matrix, inverse of the matrix is calculated using `getri`.

```
SCALAPACK.getri(bcm,rf.ipiv())
```

As mentioned, `rf.ipiv()` is used as the input of `getri`. The result is overwritten to the argument matrix. The result is printed by `print (bcm.to_numpy_matrix())`. The result would be like:

```
[[ 2.53333333 -0.36666667 -0.03333333]
 [-1.46666667  0.63333333 -0.03333333]
 [-0.03333333 -0.13333333  0.03333333]]
```

You can also use the result of LU factorization for solving the system of linear equation by using `getrs`.

Next example solves the system of linear equation directly using `gesv`.

```
bcm = FrovedisBlockcyclicMatrix(m)
x = np.matrix([[1],[2],[3]], dtype=np.float64)
bcx = FrovedisBlockcyclicMatrix(x)
SCALAPACK.gesv(bcm,bcx)
```

The variable `bcm` is set again (since it was modified) and `bcx` is created from numpy matrix `x`; then `gesv(bcm,bcx)` is called. The result is overwritten to `bcx`; `print (bcx.to_numpy_matrix())` would produce:

```
[[ 1.7]
 [-0.3]
 [-0.2]]
```

Last example is singular value decomposition (SVD) by `gesvd`. Unlike `TruncatedSVD`, it computes full SVD (it takes more time than `TruncatedSVD` if you only need part of the SVD result).

```
bcm = FrovedisBlockcyclicMatrix(m)
svd = SCALAPACK.gesvd(bcm)
```

Calling `gesvd(bcm)` creates an object `svd` that contains result. The `to_numpy_results()` function extracts left singular vectors (`umat`), singular values (`svec`), and right singular vectors (`vmat`).

```
(umat,svec,vmat) = svd.to_numpy_results()
print (umat)
print (svec)
print (vmat)
```

It would produce like:

```
[[-0.03411749 -0.21215376 -0.97664056]
 [-0.13817611 -0.96682347  0.21484819]
 [-0.98981986  0.14227847  0.00367101]]
[69.30483143  2.5940231  0.33374433]
[[-0.19214106 -0.48689005 -0.85206801]
 [-0.31038551 -0.79352539  0.52342936]
 [-0.93099014  0.36504183  0.0013452  ]]
```

You can also save and load the SVD result.

6. DataFrame

In addition to machine learning algorithms, we support Pandas like `DataFrame`.

First, please install `pandas` to your virtual environment. Though `pandas` is installed to the system Python when `Frovedis` is installed, `virtualenv` does not copy system installed packages by default.

```
(frovedis_tutorial) $ pip install pandas
```

Then, please see “src/tut6-1/tut.py”.

First, pandas DataFrame pdf1 and pdf2 are created. Then, FrovedisDataframe is created from pandas DataFrame as fdf1 and fdf2.

```
peopleDF = {
    'Ename' : ['Michael', 'Andy', 'Tanaka', 'Raul', 'Yuta'],
    'Age' : [29, 30, 27, 19, 31],
    'Country' : ['USA', 'England', 'Japan', 'France', 'Japan']
}

countryDF = {
    'Ccode' : [1, 2, 3, 4],
    'Country' : ['USA', 'England', 'Japan', 'France']
}

pdf1 = pd.DataFrame(peopleDF)
pdf2 = pd.DataFrame(countryDF)
fdf1 = FrovedisDataframe(pdf1)
fdf2 = FrovedisDataframe(pdf2)
```

To show the contents of FrovedisDataframe, you can use show():

```
fdf1.show()
fdf2.show()
```

They should produce output like:

Age	Country	Ename
29	USA	Michael
30	England	Andy
27	Japan	Tanaka
19	France	Raul
31	Japan	Yuta

Ccode	Country
1	USA
2	England
3	Japan
4	France

To select columns, you can write like:

```
fdf1[["Ename", "Age"]].show()
```

It should produce output like:

Ename	Age
Michael	29
Andy	30
Tanaka	27
Raul	19
Yuta	31

To filter the rows, you can write like:

```
fdf1[df1.Age > 19 and df1.Country == 'Japan'].show()
```

It should produce output like:

Age	Country	Ename
27	Japan	Tanaka
31	Japan	Yuta

To sort the rows, you can write like:

```
fdf1.sort("Age",ascending=False).show()
```

Since `ascending=False`, it is sorted in descending order of Age. Output should be like:

Age	Country	Ename
31	Japan	Yuta
30	England	Andy
29	USA	Michael
27	Japan	Tanaka
19	France	Raul

You can specify multiple columns for sorting.

```
fdf1.sort(["Country", "Age"]).show()
```

This sorts the rows by Country, and then by Age in the same Country name. The output should be like:

Age	Country	Ename
30	England	Andy
19	France	Raul
27	Japan	Tanaka
31	Japan	Yuta
29	USA	Michael

Please note that the rows whose Country is Japan is sorted by Age.

To groupby the table, first call `groupby` and then call `agg` to aggregate the value like:

```
fdf1.groupby('Country').agg({'Age': ['max','min','mean'],  
                             'Ename': ['count']}).show()
```

It should produce output like:

Country	count(Ename)	max(Age)	min(Age)	mean(Age)
USA	1	29	29	29
England	1	30	30	30
Japan	2	31	27	29
France	1	19	19	19

To join (or merge in Pandas term) tables, it is required that the column names are unique in the current implementation. So first we rename the column name.

```
fdf3 = fdf2.rename({'Country' : 'Cname'})
```

Then, join like this:

```
fdf1.merge(fdf3, left_on="Country", right_on="Cname").show()
```

It produces output like:

Age	Country	Ename	Ccode	Cname
29	USA	Michael	1	USA
30	England	Andy	2	England
27	Japan	Tanaka	3	Japan
19	France	Raul	4	France
31	Japan	Yuta	3	Japan

You can chain operations. Here, join, sort, and select are chained.

```
fdf1.merge(fdf3, left_on="Country", right_on="Cname") \
    .sort("Age")["Age", "Ename", "Country"].show()
```

It produces output like:

Age	Ename	Country
19	Raul	France
27	Tanaka	Japan
29	Michael	USA
30	Andy	England
31	Yuta	Japan

You can get the statistics of the columns like min, max, sum, avg, std, and count by calling like `min("Age")`. Like Pandas DataFrame, you can also call `describe()` to see all these information.

```
print ("min(Age): {}".format(fdf1.min("Age")))
print ("max(Age): {}".format(fdf1.max("Age")))
print ("sum(Age): {}".format(fdf1.sum("Age")))
print ("avg(Age): {}".format(fdf1.avg("Age")))
print ("std(Age): {}".format(fdf1.std("Age")))
print ("count(Age): {}".format(fdf1.count("Age")))
print ("describe: ")
```

This prints like:

```
min(Age): [19.0]
max(Age): [31.0]
sum(Age): [136.0]
avg(Age): [27.2]
std(Age): [4.816638]
count(Age): [5.0]
```

```
describe:
      Age
count    5.000000
mean     27.200000
std       4.816638
sum      136.000000
min       19.000000
max       31.000000
```

So far, we only used Frovedis side DataFrame. It is also possible to convert to Pandas DataFrame or use Pandas DataFrame together.

```
pdf2.rename(columns={'Country' : 'Cname'},inplace=True)
joined = fdf1.merge(pdf2, left_on="Country", right_on="Cname")
```

Here, Frovedis DataFrame is joined with Pandas DataFrame. The output should be the same as previous join.

You can convert Frovedis DataFrame using `to_panda_dataframe()`.

Frovedis DataFrame can be converted to matrix. Please see “src/tut6-2/tut.py”.

First, Pandas DataFrame is created and converted to Frovedis DataFrame.

```
data = {'A': [10, 12, 13, 15],
        'B': [10.23, 12.20, 34.90, 100.12],
        'C': ['male', 'female', 'female', 'male'],
        }
pdf = pd.DataFrame(data)
df = FrovedisDataframe(pdf)
print (pdf)
```

The DataFrame is:

	A	B	C
0	10	10.23	male
1	12	12.20	female
2	13	34.90	female
3	15	100.12	male

You can create `FrovedisRowmajorMatrix` by specifying the columns. The columns should be integer or floating point values. In this case,

```
row_mat = df.to_frovedis_rowmajor_matrix(['A', 'B'], dtype=np.float64)
print (row_mat.to_numpy_matrix())
```

In this case, columns A and B are selected and converted to matrix. This produces

```
[[ 10.    10.23]
 [ 12.    12.2 ]
 [ 13.    34.9 ]
 [ 15.   100.12]]
```

You can also create `FrovedisColmajorMatrix` by `to_frovedis_colmajor_matrix`.

Then, you can specify columns as category variable. In this case, it can be any data type; it is converted using on-hot encoding. In this case, the result becomes `FrovedisCRSMatrix`.

```
crs_mat,info = df.to_frovedis_crs_matrix(['A', 'B', 'C'],
                                         ['C'], need_info=True)
crs_mat.debug_print()
```

Here, columns 'A', 'B', and 'C' is selected to create the matrix. The second argument is to specify which column is used as categorical variable. In this case column 'C' is specified. If `need_info=True`, `info` data structure is also returned. It is used to create a matrix from `FrovedisDataFrame` next time (explained later).

The result of debug print is as follows:

```
num_row = 4, num_col = 4
node 0
local_num_row = 2, local_num_col = 4
val : 10 10.23 1 12 12.2 1
idx : 0 1 3 0 1 2
off : 0 3 6
node 1
local_num_row = 2, local_num_col = 4
val : 13 34.9 1 15 100.12 1
idx : 0 1 2 0 1 3
off : 0 3 6
```

If it is shown as dense matrix, it should look like:

```
10 10.23 0 1
12 12.2  1 0
13 34.9  1 0
15 100.12 0 1
```

Here, 'female' is assigned to 2nd column (start from 0), and 'male' is assigned to 3rd column.

If you use this data for machine learning, you would want to convert other matrix using the same way for inference, for example. The `info` structure is used for this purpose.

For example,

	A	B	C
0	12	34.56	male
1	13	78.90	male

This `DataFrame` is converted to `FrovedisCRSMatrix` using the `info` created above:

```
crs_mat2 = df2.to_frovedis_crs_matrix_using_info(info)
crs_mat2.debug_print()
```

This should produce output like:

```

matrix:
num_row = 4, num_col = 4
node 0
local_num_row = 1, local_num_col = 4
val : 12 34.56 1
idx : 0 1 3
off : 0 3
node 1
local_num_row = 1, local_num_col = 4
val : 13 78.9 1
idx : 0 1 3
off : 0 3

```

If it is shown as dense matrix, it should look like:

```

12 34.56 0 1
13 78.9 0 1

```

As you can see, ‘male’ is assigned to 3rd column, not 2nd column. The data structure `info` can be saved and loaded to a file.

7. Manuals

Manuals are in `../manual` directory. In addition to PDF file, you can also use `man` command (MANPATH is set in `x86env.sh` or `veenv.sh`). For python interface, the section is `3p` (same name of the manual may exist in section `3` or `3s`.), so you can run like `man -s 3p logistic_regression`. Currently, there are following manual entries:

- `logistic_regression`
- `linear_regression`
- `lasso_regression`
- `ridge_regression`
- `linear_svm`
- `kmeans`
- `als`
- `crs_matrix`
- `dvector`
- `blockcyclic_matrix`
- `scalapack_wrapper`
- `pblas_wrapper`
- `arpack_wrapper`
- `getrf_result`
- `gesvd_result`