LIBRA

Release 0.0.28

Xabier

Nov 17, 2022

GETTING STARTED

LIBRA is a Python toolbox for Single-Cell data integration and prediction. This project aims to lower the barrier for new users to AI state-of-the-art techniques and propose an unbalanced-AE for shared latent representation extraction between two paired-Single-Cell omics. This package provides state-of-art performance while retains good execution timing in opposite to other AI methods that require many hours to finish similar training processes. This makes possible to train hundreds of models in parallel in search of parameters that further optimize the results in moderate times using aLIBRA.

LIBRA has been designed using TensorFlow, any system that supports it is compatible with LIBRA package. It has been tested on LINUX OS.

We encourage users and developers to report problems, request features, ask for help, or leave general comments on GitHub. Please refer to our usage guide if you wish to extend LIBRA's functionality and/or contribute to the project.

LIBRA is distributed under the open source GNU General Public License v3.0. Please cite this paper if you publish work using LIBRA:

CHAPTER

ONE

QUICKSTART

LIBRA is a modular toolbox and hence easy to use. All outputs from functions and directories tree are generated behind scenes and required user interaction is low.

Follow this steps to a proper installation:

1. Install Python **>=3.7.0**.
2. Install R **>=3.5.2**.
3. Install sc_libra Python package.
4. **OPTIONAL: Prepare the environment (Only for selecting a specific R version in case many are installed, otherwise avoid this step)**.
5. Importing sc_libra.

1.1 Installation

LIBRA is compatible with Python >=3.7, and depends on rpy2, NumPy, SciPy, Pandas and Keras. All required dependencies will be automatically installed when running:

```
$ pip install sc_libra
```

1.2 (Optional) Environment preparation

LIBRA makes use of rpy2 for running some specific R functions. In order to import properly this dependency is mandatory that Python knows where are the R libs for the specific R version used. This is a requirement of rpy2 and should be done, else sc_libra will raise an error when importing it (as it will import all dependencies required as it is imported) This can be done in two steps:

1. Run on console prior to run Python.

```
$ #Typical locations are:
$ # export LD_LIBRARY_PATH="/opt/R/3.5.2/lib64/R/lib:$LD_LIBRARY_PATH" (if local␣
↪installation of R was done)
$ # export LD_LIBRARY_PATH="/usr/lib64/R/:$LD_LIBRARY_PATH" (if global installaltion of␣
↪R was done)

$ export LD_LIBRARY_PATH="/YOUR_PATH_TO_R_LIBS_HERE:$LD_LIBRARY_PATH"
```

2. Open your Python environment and run:

```
$ import os
$ # Typical locations are:
$ # export os.environ['R_HOME'] = "/opt/R/3.5.2/lib64/R/" (if local installation of R was␣
↪done)
$ # export os.environ['R_HOME'] = "/usr/lib64/R/" (if global installation of R was done)

$ os.environ['R_HOME'] = "/YOUR_POATH_TO_R_HERE"
```

CHAPTER

TWO

TUTORIAL

LIBRA provides four main functions that cover all required aspects of the proposed pipeline.

- *1. Loading input data*
- *2. Training LIBRA model*
- *3. Prediction using LIBRA model*
- *4. Metrics computation*

2.1 Getting started

In order to have an ordered working directory is recommended to set the directory path desired where the tree of folders and files will be saved during the execution of the different functions. The tree of folders will be generated under working directory path automatically. LIBRA_outputs will be generated as the root folder and inside an additional three directories will be generated to store different outputs: *Integration*, *Models* and *supp_Models*.

```
import sc_libra
os.chdir("/desired_working_directory_path")
```

2.2 1. Loading input data

A universal loading data is provided for easy data loading into Python environment. Input file formats will be automatically detected among: **AnnData(".h5ad"), sparse matrix(".mtx",".txt"), comma separated matrix(.csv), 10XGenomics(.mtx folder) and 10XGenomics(.h5 file)**. If the format is different data should be loaded by the user with the corresponding command and harmonized as this function does.

In order dataset1 is the input modality to LIBRA model and dataset2 the output modality. **We strongly recommend to set RNA as the second modality**. By now specific actions are performed over RNA omic (feature space filtering to top 2000 most variable) so it is **important to set "RNA" name to RNA omic input dataset** (this requires that RNA should be normalized and log2 transformed to perform scanpy hvg function), in other cases the desired name can be used (no process will be applied on the data). If both input datasets are in working directory path, no paths are required by *load_data* function.

- **Input files are in working directory:**

```
par = sc_libra.load_data("ATAC","RNA", dataset1='example_dataset1_atac.h5ad', dataset2=
↪'example_dataset2_rna.h5ad')
```

- **Input files are in another directory than working directory):**

If input datasets are in a different location than working directory, *dataset1_path* and *dataset2_path* can be used (*dataset2_path* will be equal to dataset1_path if not setted).

```
par = sc_libra.load_data("ATAC","RNA", dataset1_path='/both_are_in_this_different_
↪location', dataset1='example_dataset1_atac', dataset2='example_dataset2_rna.h5ad')
```

- **10X folder as input:**

If 10X folder contains the input data desired, specify only the path where files are and they will be automatically loaded.

```
par = sc_libra.load_data("ATAC","RNA", dataset1_path='/location_to_10x_folder_for_input_
↪omic_ATAC', dataset2_path='/location_to_10x_folder_for_output_omic_RNA')
```

Output format for downstream analysis As a result output (*par* in these examples) will contain a dictionary such as:

- {**omic_1_name**: pandas.dataframe.omic1, **omic_2_name**: pandas.dataframe.omic2}.

2.3 2. Training LIBRA model

LIBRA can run in many different ways using the *libra* function. This step uses the previously generated dictionary as input (in this example, *par*), if you want to run *libra* as part of an existing pipeline a dictionary with the above structure can be created by the user for the compatibility with the following functions.

- **Default use:**

The most basic way is to follow the example presented. This will train the LIBRA model with default parameters finding a good balance between prediction/integration performance. Will generate integration output file containing latent space for each cell and store it in the automatically generated tree of directories. The model will also be stored in .hdf5 format.

```
output_data = sc_libra.libra(par)
```

- **Boosting one task over the other:**

LIBRA can also be used for training a bunch of models for boosting performance on one of the main tasks over the other (prediction/integration). To this aim, a grid of parameters will be used generating hundreds of models and storing the outputs following the same default schema. A custom grid can also be used if desired by user.

```
#For prediction best model finding
output_data = sc_libra.libra(par, training_mode = 'fine_tune_prediction')
#For prediction best model finding
output_data = sc_libra.libra(par, training_mode = 'fine_tune_integration')
#For custom grid user
output_data = sc_libra.libra(par, training_mode = 'custom')
```

- **Using another amount of genes than 2000 HVG:**

Extra parameters can be added to the function for example *n_top_genes*. In the case of containing an omic named as "RNA" *libra* function will filter gen space to contain only the most 2000 highly variable genes, this is performed because in our experiments RNA has proved to provide better performance over LIBRA model when only using HVG. If a different amount of genes is wanted it can be setted as in the following example:

```
#For use 3000 number of HVG
output_data = sc_libra.libra(par, n_top_genes = 3000)
```

- **Parallel training for grid based version:**

For bosting speed (if user hardware is sufficient) and extra parameter can be added, *n_jobs*. This parameter setted as default to 1, can be changed to any amount of cores present in users CPU to perform multiple models trainings in paralel. This is designed specifically for other that the default *libra* option where many models will be trained depending on grid selected. This reduces the time required but also requires more RAM memory.

```
output_data = sc_libra.libra(par, n_jobs=20) #For training 20 models in parallel (your␣
↪CPU should have at least 20 cores, and enought RAM to handle them in memmory).
```

All these parameters can be combined for desired task.

2.4 3. Prediction using LIBRA model

If user wants to use LIBRA model generated for a prediction task over same or new input dataset, it can be done through this function, *libra_predict* as the following example. Either latent of output spaces can be predicted.

```
model = load_model('/.../LIBRA_outputs/Models/model_n_layers2_n_nodes512_alpha0.3_
↪dropout0.2_batch_size7000_mid_layer10.hdf5')
input_data = output_data[0].todense() #For predict over input dataset. A novel one can␣
↪be used here.
to_predict = 'integrated_space' #For latent space prediction or 'modality_B' for output␣
↪prediction.

predicted_data = sc_libra.libra_predict(model, input_data, to_predict)
```

2.5 4. Metrics computation

LIBRA provides a function *libra_metrics* to compute three different measurements explained on the paper.

Setting *libra_metrics* metric parameter as *nn_consistency* will compute euclidean distance between latent space computed in LIBRA model to output obtained of a secondary neural network with same hyperparameters to encode to the obtained latent space. Through this metric the consistency of the neural network can be measured for each independent paired cell. Biomodal distances for each modal peak will be given and plotted as output apart from the global euclidean distance computed for each cell and encoding models in .hdf5 format. If multiple output models are present in folder due to a grid used during model training, metric will be computed for all available models and all outputs will be stored with the corresponding hyperparameter as names. If user desires only to compute metric over one specific model it can be selected through the *libra_output* parameter. In order to train these secondary networks in parallel *n_jobs* parameter let user select the number of models to be trained at same.

- **nn_consistency**:

```
output_metris=sc_libra.libra_metrics(output_data, metric='nn_consistency', n_jobs=20,␣
↪path_to_libra_outputs='/...LIBRA_outputs/Integration/') #For compute over all models␣
↪trained with a parallel value of 20.
```

- **nn_mse:**

Setting the metric parameter as *nn_mse* will predict overall present models stored and compute the mean squared error against the output omic. As previously *libra_output* can be used to specify the name of a model to compute it only for the desired model. Outputs will be summarized and stored in the corresponding path automatically.

```
output_metris=sc_libra.libra_metrics(output_data, metric='nn_mse', path_to_libra_outputs=
↪'/...LIBRA_outputs/Models/')
```

- **ppji:**

Finally PPJI metric can be computed against the reference obtained clustering of either omics to measure how preserved is the biological information in clusters in the integrated latent space obtained in LIBRA model. To include this reference clustering information *cluster_origin* parameter is used. To feed this parameter information "cluster_origin=adata.obs['leiden']" serves as example of expected input format (pandas.core.series.Series), one column dataframe extracted from Seurat meta.data is also valid and will be automatically transformed into required format (from pandas.core.frame.DataFrame to pandas.core.series.Series). **We strongly recommend to compute reference clusterings using *leiden* algorithm as it has proved to provide good results and to exclude divergences in clusters due to different algorithms used and not because of the model performance (LIBRA use *leiden* as the method for latent clustering computation).** Desired level of granularity can be modified by changing *n_neighbors* (default is 10) parameter and *resolution* (default is 1). As before *libra_output* can be used to specify the name of a model to compute it only for the desired model. Outputs will be saved after function ends.

```
output_metris=sc_libra.libra_metrics(output_data, cluster_origin=your_reference_cluster,␣
↪metric='ppji', path_to_libra_outputs='/...LIBRA_outputs/Integration/')
```