The Journal of Open Source Software

OnlineNMF.jl: A Julia Package for Out-of-core and Sparse Non-negative Matrix Factorization

Koki Tsuyuzaki[1, 2]

1 Department of Artificial Intelligence Medicine, Graduate School of Medicine, Chiba University, Japan **2** Laboratory for Bioinformatics Research, RIKEN Center for Biosystems Dynamics Research, Japan

DOI: DOIunavailable

Software

- Review
- Repository
- Archive

Editor: Pending Editor

Reviewers:

- @Pending Reviewers

Submitted: N/A
Published: N/A

Summary

Non-negative Matrix Factorization (NMF) is a widely used dimensionality reduction technique for identifying a small number of non-negative components that minimize the reconstruction error when applied to high-dimensional data (Meng, 2016; Stein-O'Brien, 2018). NMF has been applied across various fields of data science, including face recognition (Lee, 1999), audio signal processing (Kameoka, 2015), recommender system (Sajad, 2025), natural language processing (also known as a "topic model") (Srivastava & Sahami, 2009), population genetics (also known as "admixture analysis") (Simanovsky, 2019), and omics studies Rodriques (2019).

Despite its broad applicability, NMF becomes computationally prohibitive for large data matrices, making it difficult to apply in practice. In particular, recent advances in single-cell omics have led to datasets with millions of cells, for which standard NMF implementations often fail to scale. To meet this requirement, I originally developed `OnlineNMF.jl`, which is a Julia package to perform some NMF algorithms (https://github.com/rikenbit/OnlineNMF.jl).

Statement of need

NMF is a workhorse algorithm for most data science tasks. However, as the size of the data matrix increases, it often becomes too large to fit into memory. In such cases, an out-of-core (OOC) implementation — where only subsets of data stored on disk are loaded into memory for computation — is desirable. Additionally, representing the data in a sparse matrix format, where only non-zero values and their coordinates are stored, is computationally advantageous. Therefore, a NMF implementation that supports both OOC computation and sparse data handling is highly desirable (Figure 1).

Similar discussions have been made in the context of Principal Component Analysis (PCA), and we have independently developed a Julia package, `OnlinePCA.jl` (Tsuyuzaki, 2020). `OnlineNMF.jl` is a spin-off version of `OnlinePCA.jl`, implementing NMF.

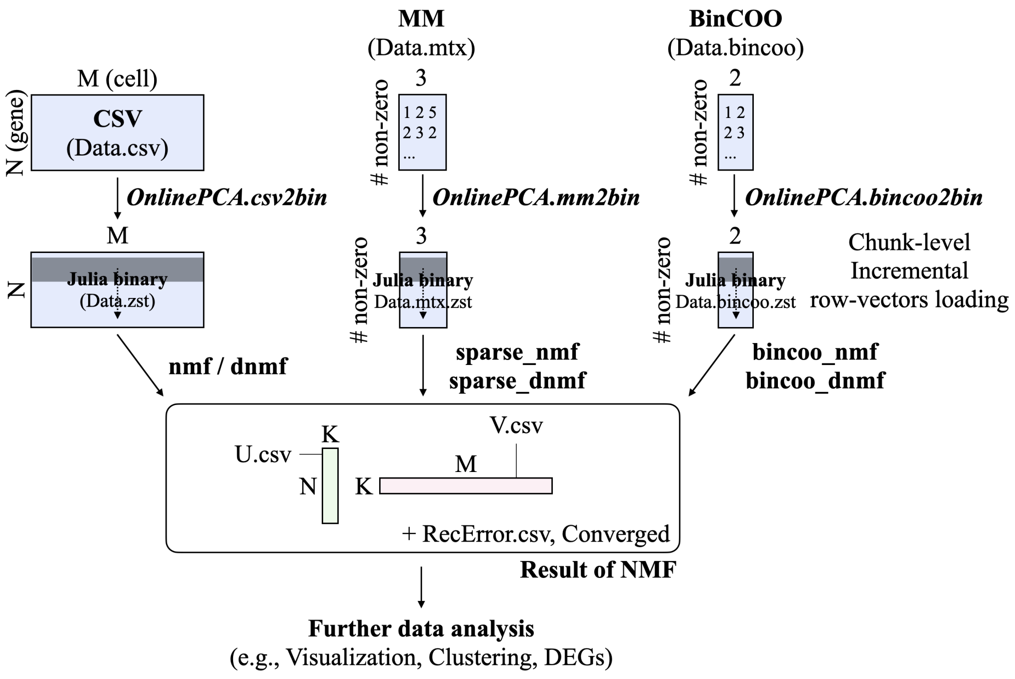

Figure 1: Overview of workflow in OnlineNMF.jl.

Example

NMF can be easily reproduced on any machine where Julia is pre-installed by using the following commands in the Julia REPL window:

Installation

First, install `OnlineNMF.jl` from the official Julia package registry or directly from GitHub:

```
# Install OnlineNMF.jl from Julia General
julia> Pkg.add("OnlineNMF")

# or GitHub for the latest version
julia> Pkg.add(url="https://github.com/rikenbit/OnlineNMF.jl.git")
```

Preprocess of CSV

Then, write a synthetic data as a CSV file, convert it to a compressed binary format using Zstandard, and prepare summary statistics for PCA. Matrix Market (MM) format is also supported for sparse matrices.

```
using OnlinePCA
using OnlinePCA: write_csv
using OnlineNMF
using Distributions
using DelimitedFiles
using SparseArrays
using MatrixMarket

# CSV
tmp = mktempdir()
```

```
data = rand(Binomial(10, 0.05), 300, 99)
data[1:50, 1:33] .= 100*data[1:50, 1:33]
data[51:100, 34:66] .= 100*data[51:100, 34:66]
data[101:150, 67:99] .= 100*data[101:150, 67:99]
write_csv(joinpath(tmp, "Data.csv"), data)

# Matrix Market (MM)
mmwrite(joinpath(tmp, "Data.mtx"), sparse(data))

# Binarization (Zstandard)
csv2bin(csvfile=joinpath(tmp, "Data.csv"), binfile=joinpath(tmp, "Data.zst"))

# Sparsification (Zstandard + MM format)
mm2bin(mmfile=joinpath(tmp, "Data.mtx"), binfile=joinpath(tmp, "Data.mtx.zst"))
```

Setting for plot

Define a helper function to visualize the results of NMF using the `PlotlyJS.jl` package. It generates two subplots: Component-1 vs Component-2 and Component-2 vs Component-3, with color-coded groups.

```
using DataFrames
using PlotlyJS

function subplots(out_nmf, group)
    # data frame
    data_left = DataFrame(nmf1=out_nmf[1][:,1], nmf2=out_nmf[1][:,2],
      group=group)
    data_right = DataFrame(nmf2=out_nmf[1][:,2], nmf3=out_nmf[1][:,3],
      group=group)
    # plot
    p_left = Plot(data_left, x=:nmf1, y=:nmf2, mode="markers",
      marker_size=10, group=:group)
    p_right = Plot(data_right, x=:nmf2, y=:nmf3, mode="markers",
      marker_size=10,
    group=:group, showlegend=false)
    p_left.data[1]["marker_color"] = "red"
    p_left.data[2]["marker_color"] = "blue"
    p_left.data[3]["marker_color"] = "green"
    p_right.data[1]["marker_color"] = "red"
    p_right.data[2]["marker_color"] = "blue"
    p_right.data[3]["marker_color"] = "green"
    p_left.data[1]["name"] = "group1"
    p_left.data[2]["name"] = "group2"
    p_left.data[3]["name"] = "group3"
    p_left.layout["title"] = "Component 1 vs Component 2"
    p_right.layout["title"] = "Component 2 vs Component 3"
    p_left.layout["xaxis_title"] = "nmf-1"
    p_left.layout["yaxis_title"] = "nmf-2"
    p_right.layout["xaxis_title"] = "nmf-2"
    p_right.layout["yaxis_title"] = "nmf-3"
    plot([p_left p_right])
end

group=vcat(repeat(["group1"],inner=100),
    repeat(["group2"],inner=100),
    repeat(["group3"],inner=100))
```

NMF based on Alpha-Divergence

This example demonstrates NMF using the α-divergence as the loss function (Figure 2). By setting alpha=2, the objective corresponds to the Pearson divergence. The input data is assumed to be a dense matrix compressed with Zstandard (.zst format).

```
out_nmf_alpha = nmf(input=joinpath(tmp, "Data.zst"),
    dim=3, alpha=2, numepoch=30, algorithm="alpha")

subplots(out_nmf_alpha, group)
```

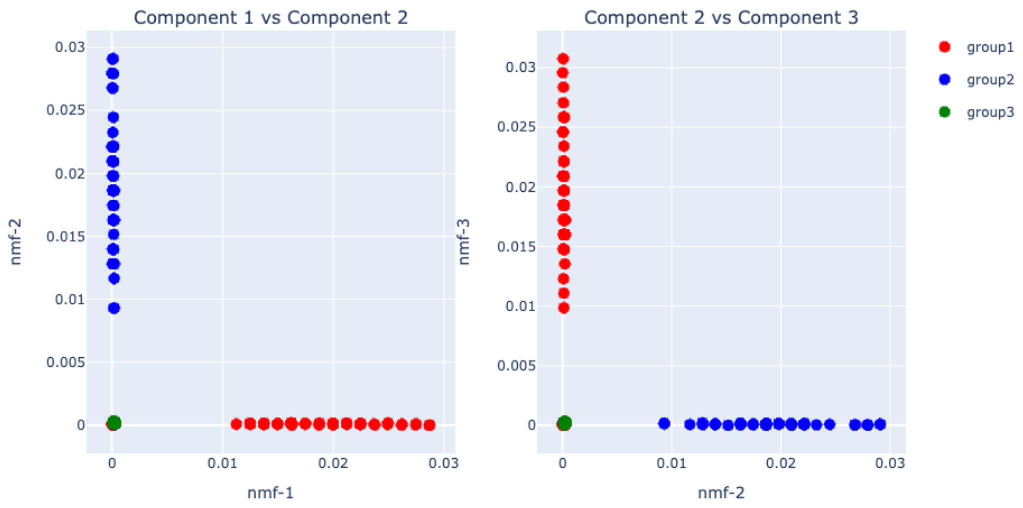

Figure 2: Output of nmf against binarized CSV format.

Sparse-NMF based on Beta-Divergence

This example performs NMF on a sparse matrix using the β-divergence (Figure 3). The input is a MM formatted sparse matrix file (.mtx.zst). When beta=1, the loss corresponds to the Kullback-Leibler divergence, and sparse-specific optimization is used internally.

```
out_sparse_nmf_beta = sparse_nmf(input=joinpath(tmp, "Data.mtx.zst"),
    dim=3, beta=1, numepoch=30, algorithm="beta")

subplots(out_sparse_nmf_beta, group)
```

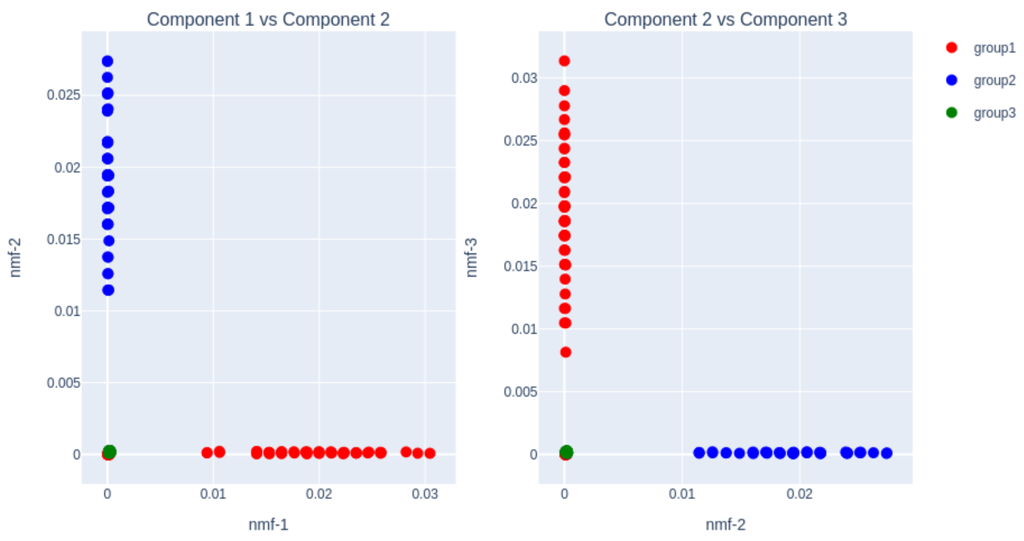

Figure 3: Output of sparse_nmf against binarized MM format.

Related work

There are various implementations of NMF (Boureima, 2024; Pedregosa, 2011; Tsuyuzaki, 2023) and some of them support OOC computation or sparse data formats (Lab, 2023; Pedregosa, 2011). While `RcppPlanc/PLANC` supports both OOC and R's internal sparse format (dgCMatrix), `OnlineNMF.jl` is designed to handle language-agnostic sparse formats such as MM and Binary COO (BinCOO), enabling seamless integration with external data pipelines.

Function Name	Language	OOC	Sparse Format
`nnTensor::NMF`	R	No	-
`sklearn.decomposition.NMF`	Python	No	-
`pyDNMFk`	Python	No	-
`NMF.MultUpdate`	Julia	No	-
`sklearn.decomposition.MiniBatchNMF`	Python	Yes	-
`RcppPlanc/PLANC`	R/C++	Yes	dgCMatrix
`OnlineNMF.jl`	Julia	Yes	MM/BinCOO

References

Boureima, I. et al. (2024). Distributed out-of-memory NMF on CPU/GPU architectures. *J Supercomput*, *80*, 3970–3999. https://doi.org/10.1007/s11227-023-05587-4

Kameoka, H. (2015). Non-negative matrix factorization and its variants with applications to audio signal processing. *Journal of the Japan Statistical Society, Japanese Issue*, *44(2)*, 383–407. https://doi.org/10.11329/jjssj.44.383

Lab, W. (2023). *RcppPlanc: R wrapper for the PLANC nonnegative matrix factorization library*. https://github.com/welch-lab/RcppPlanc. https://doi.org/10.32614/CRAN.package.RcppPlanc

Lee, D. D. et al. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401(6755)*, 788–791. https://doi.org/10.1038/44565

Meng, C. et al. (2016). Dimension reduction techniques for the integrative analysis of multi-omics data. *Briefings in Bioinformatics*, *17(4)*, 628–641. https://doi.org/10.1093/bib/bbv108

Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12(85)*, 2825–2830.

Rodriques, S. G. et al. (2019). Slide-seq: A scalable technology for measuring genome-wide expression at high spatial resolution. *Science*, *363*, 1463–1467. https://doi.org/10.1126/science.aaw1219

Sajad, A. et al. (2025). Recommender systems based on non-negative matrix factorization: A survey. *IEEE Transactions on Artificial Intelligence*, 1–21. https://doi.org/10.1109/TAI.2025.3559053

Simanovsky, A. L. et al. (2019). Single haplotype admixture models using large scale HLA genotype frequencies to reproduce human admixture. *Immunogenetics*, *71*, 589–604. https://doi.org/10.1007/s00251-019-01144-7

Srivastava, A. N., & Sahami, M. (Eds.). (2009). *Text mining: Classification, clustering, and applications*. Chapman; Hall/CRC. https://doi.org/10.1201/9781420059458

Stein-O'Brien, G. L. et al. (2018). Enter the matrix: Factorization uncovers knowledge from omics. *Trends in Genetics*, *34(10)*, 790–805. https://doi.org/10.1016/j.tig.2018.07.003

Tsuyuzaki, K. et al. (2020). Benchmarking principal component analysis for large-scale single-cell RNA-sequencing. *Genome Biology*, *21(1)*. https://doi.org/10.1186/s13059-019-1900-3

Tsuyuzaki, K. et al. (2023). nnTensor: An r package for non-negative matrix/tensor decomposition. *Journal of Open Source Software*, *8(84)*, 5015. https://doi.org/10.21105/joss.05015