NIPALS_PCA

A Julia package for calculating PCA and PLS using the NIPALS implementation. Both models handles missing values

The package contains data structures for models and datasets

Installation

In Julia add https://gitlab.moffitt.usf.edu:8000/Bios2Projects/NIPALS_PCA as unregistered package

```
using Pkg
Pkg.add("https://gitlab.moffitt.usf.edu:8000/Bios2Projects/NIPALS_PCA")
```

Running Julia REPL

Julia can be started using

1. The base installation
2. From Singularity container (in progress, details pending)
3. Utilizing the downloaded folder as a local environment.

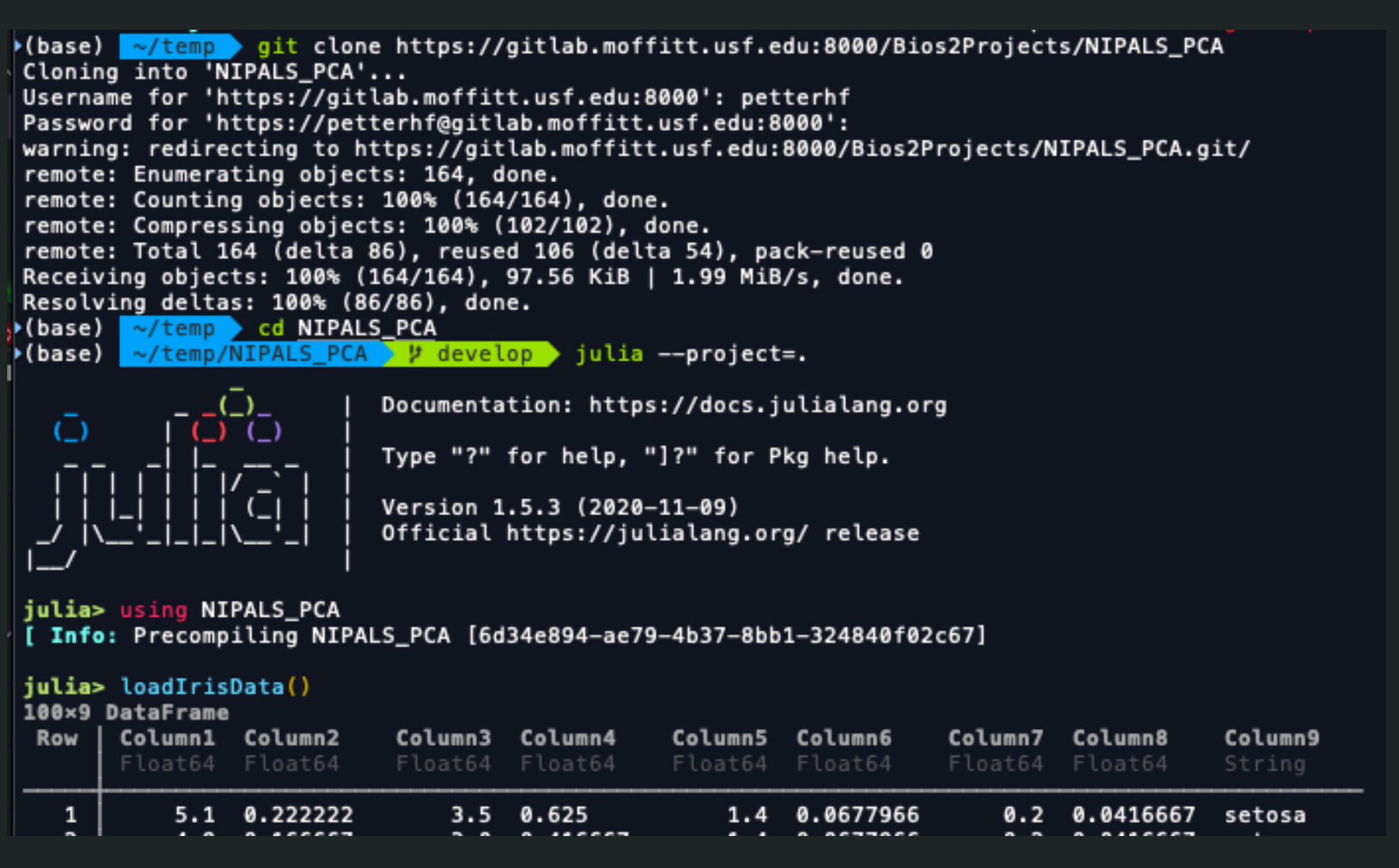
To activate NIPALS_PCA as local environment

```
cd path/to/cloned/NIPALS_PCA
julia --project=.
```

Loading package

```
using NIPALS_PCA
```

Example



On cluster using Singularity

A prebundled singularity container for Julia can be accessed at /share/data2/applications/singularity_images/bbsrTools.sif

-C, is required to utilize packages installed in the container. If this flag is not used, Julia will load packages from the users home directory

-B, binds folders to be accessible to the container. Repeated -B flags can be entered for multiple binds

```
module load singularity/3.10
singularity run -C -B ../data:/data  path/to/bbsrJuliaTools2.sif
```

or if running script

```
module load singularity/3.10
singularity run -C -B ../data:/data  path/to/bbsrJuliaTools2.sif myScript.jl args
```

Tutorial

PCA modelling

From Julia REPL

1)Load package

```
using NIPALS_PCA
```

2)Load dataset from .csv file to DataFrame

```
x_df = loadIrisData()
```

3)Create dataset and apply normalize to mean center data

```
xdataset = parseDataFrame(x_df) |> normalize
```

4)Calculate PCA model

```
pca = calcPCA(xdataset, 3)
```

5)Calculate variances for model

```
calcVariances(xdataset,pca)
```

PLS normalization

The PLS normalization workflow can either be run from a script or from an interactive Julia session. The default script can be find in src/scripts/plsnorm.jl

Run from script

```
julia src/scripts/plsnorm.jl \
--xfile /path/to/xmatrix.txt \
--yfile /path/to/ymatrix.txt \
--ycategorical "colname" \
--ycontinous "colname1;colname2" \
--mode calibrate \
--modelfile model.jld2 \
--outfile output_file.csv
```

Run from interactive session

```
using NIPALS_PCA

parsed_args=Dict{String,Any}("xfile" => "/path/to/xmatrix.txt","ycategorical" => "colname","yfi

#to calibrate
calibrate_model(parsed_args)

#to correct
correct(parsed_args)
```

Run on cluster using Singularity image

The folder(s) used for reading and writing needs a Bind. In the example below
 -B ../data/:/data
 data will be accessible from the root level within the container. Multiple folders can be bound.

```
module load singularity/3.10

singularity run --app plsnorm  -C -B ../data/:/data bbsrJuliaTools2.sif \
--xfile /data/xmatrix.txt \
--yfile /data/ymatrix.txt \
--ycategorical "colname" \
--ycontinous "colname1;colname2" \
--mode calibrate \
--modelfile model.jld2 \
--outfile output_file.csv
```

Get help

```
julia plsnorm.jl --help
```

Structures

`NIPALS_PCA.Dataset` — Type

```
struct Dataset
```

- `X::Array{Union{Missing, Float64},2}`
- `means::Array{Float64,1}`
- `stdevs::Array{Float64,1}`
- `value_columns::Array{String,1}`
- `xmask::BitArray{2}`
- `mv::Bool`
- `mvs::Array{Float64,1}`
- `ranges::Array{Float64,1}`

`NIPALS_PCA.PCA` — Type

```
struct PCA <: NIPALS_PCA.MultivariateModel
```

- `T::DataFrames.DataFrame`
- `P::DataFrames.DataFrame`

`NIPALS_PCA.PLS` — Type

```
struct PLS <: NIPALS_PCA.MultivariateModel
```

- `T::DataFrames.DataFrame`
- `P::DataFrames.DataFrame`
- `C::DataFrames.DataFrame`
- `W::DataFrames.DataFrame`
- `U::DataFrames.DataFrame`

Functions

General functionality

`NIPALS_PCA.calcPCA` — Function

calcPCA(dataset::Dataset, comps::Int64=3; dCrit = 1e-23, maxIter = 1000)::PCA

```
Calculates a PCA model
```

Examples

```
julia> calcPCA(datset,3)
```

`NIPALS_PCA.calcPLS` — Function

calcPLS(xdataset::Dataset,ydataset::Dataset,comps::Int64,incsamples::Array{Int64,1} = collect(1:size(xdataset.X)[1]))

```
Calculates a PLS model
```

`NIPALS_PCA.calcVariances` — Function

calcVariances(dataset::Dataset, model::PCA)::NamedTuple

```
Calculates r2x, r2x_cum and eigenvalues for all components in PCA model
```

Examples

```
julia> r2x,r2x_cum,eigenvalues = calcPCA(datset,model)
```

`NIPALS_PCA.loadmodel` — Function

loadmodel(path::String)::Tuple{MultivariateModel,Array{Float64,1},Array{Float64,1}}

```
Load PCA or PLS model from JLD2 file into a tuple containing the model, variable standard d
```

`NIPALS_PCA.savemodel` — Function

savemodel(model::T, dataset::Dataset, name::String) where T <: MultivariateModel

```
Save PCA or PLS model as JLD2 file
```

PLS normalization

`NIPALS_PCA.calibrate_model` — Function

calibrate_model(x::DataFrame,y::DataFrame,A::Int64, modelfile::String)

```
Calibrates PLS model based on datatypes in DataFrame for y

Columns of type CategoricalArray is handled by one-hot precedure

The calibrated model is saved to specified locations
```

calibrate*model(parsed*args::Dict{String,Any})

```
Calibrates PLS model based on datatypes in DataFrame for y

Columns of type CategoricalArray is handled by one-hot precedure

The calibrated model is saved to specified locations
```

`NIPALS_PCA.correct` — Function

correct(model::T, dataset::Dataset, name::String) where T <: MultivariateModel

```
Save PCA or PLS model as JLD2 file
```

`NIPALS_PCA.predict_xres` — Function

predict_xres(modelfile::String,xfile::String, outfile::String)

```
Loads model from jld2 file, predicts using xfile and exports residual matrix into .csv file
```