

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

- The base installation
- From Singularity container (in progress, details pending)
- 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

```
(base) ~/:chmp git clone https://gitlab.moffitt.usf.edu:8000/Bios2Projects/NIPALS_PCA
Cloning into 'NIPALS_PCA'...
Username for 'https://gitlab.moffitt.usf.edu:8000': petterrh7
Password for 'https://petterrh7@gitlab.moffitt.usf.edu:8000':
Warning: redirecting to https://gitlab.moffitt.usf.edu:8000/Bios2Projects/NIPALS_PCA.git/
remote: Enumerating objects: 164, done.
remote: Counting objects: 100% (164/164), done.
remote: Compressing objects: 100% (182/182), done.
remote: Total 164 (delta 86), reused 106 (delta 54), pack-reused 0
Receiving objects: 100% (164/164), 97.56 KiB | 1.99 MiB/s, done.
Resolving deltas: 100% (86/86), done.
(base) ~/:chmp cd NIPALS_PCA
(base) ~/temp/NIPALS_PCA ~/:developer julia --project=.
```

Documentation: <https://docs.julialang.org>
Type "?" for help, "]" for Pkg help.
Version 1.5.3 (2020-11-09)
Official <https://julialang.org/> release

```
julia> using NIPALS_PCA
[ Info: Precompiling NIPALS_PCA [6d34e894-ae79-4b37-8bb1-324840f02c67]

julia> loadIrisData()
180x9 DataFrame
 Row | Column1  Column2  Column3  Column4  Column5  Column6  Column7  Column8  Column9
     | Float64  Float64  Float64  Float64  Float64  Float64  Float64  Float64  String
-----|-----|-----|-----|-----|-----|-----|-----|-----
  1  | 5.1  0.222222  3.5  0.625  1.4  0.0677966  0.2  0.0416667  setosa
```

On cluster using Singularity

(in progress, details pending) A prebundled singularity container for Julia can be accessed at [/share/data2/applications/singularity_images/bbsrTools.sif](#)

```
module load singularity/3.10
sudo singularity run --app plsnorm -C -B ../data:/data bbsrTools.sif [arguments]
```

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" \
--ycontinuous "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)
```

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`

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)

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

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