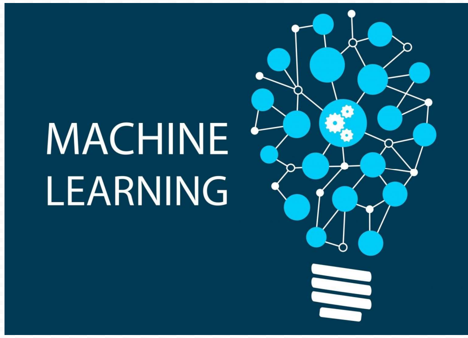
Local PLS with package Jchemo

matthieu.lesnoff@cirad.fr



SensorFINT 08/09/2022 Sète, France

https://julialang.org

julia

Download Documentation Learn Blog Community Research JSoC Sponsor

The Julia Programming Language

Download Documentation

Star 40,167

Julia in a Nutshell

Fast

Julia was designed from the beginning for high performance. Julia programs compile to efficient native code for multiple platforms via LLVM.

Dynamic

Julia is dynamically typed, feels like a scripting language, and has good support for interactive use.

Reproducible

Reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with pre-built binaries.

Composable

Julia uses multiple dispatch as a paradigm, making it easy to express many object-oriented and functional programming patterns. The talk on the Unreasonable Effectiveness of Multiple Dispatch explains why it works so well.

General

Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and more. One can build entire Applications and Microservices in Julia.

Open source

Julia is an open source project with over 1,000 contributors. It is made available under the MIT license. The source code is available on GitHub.

Download Julia

Star 40,198

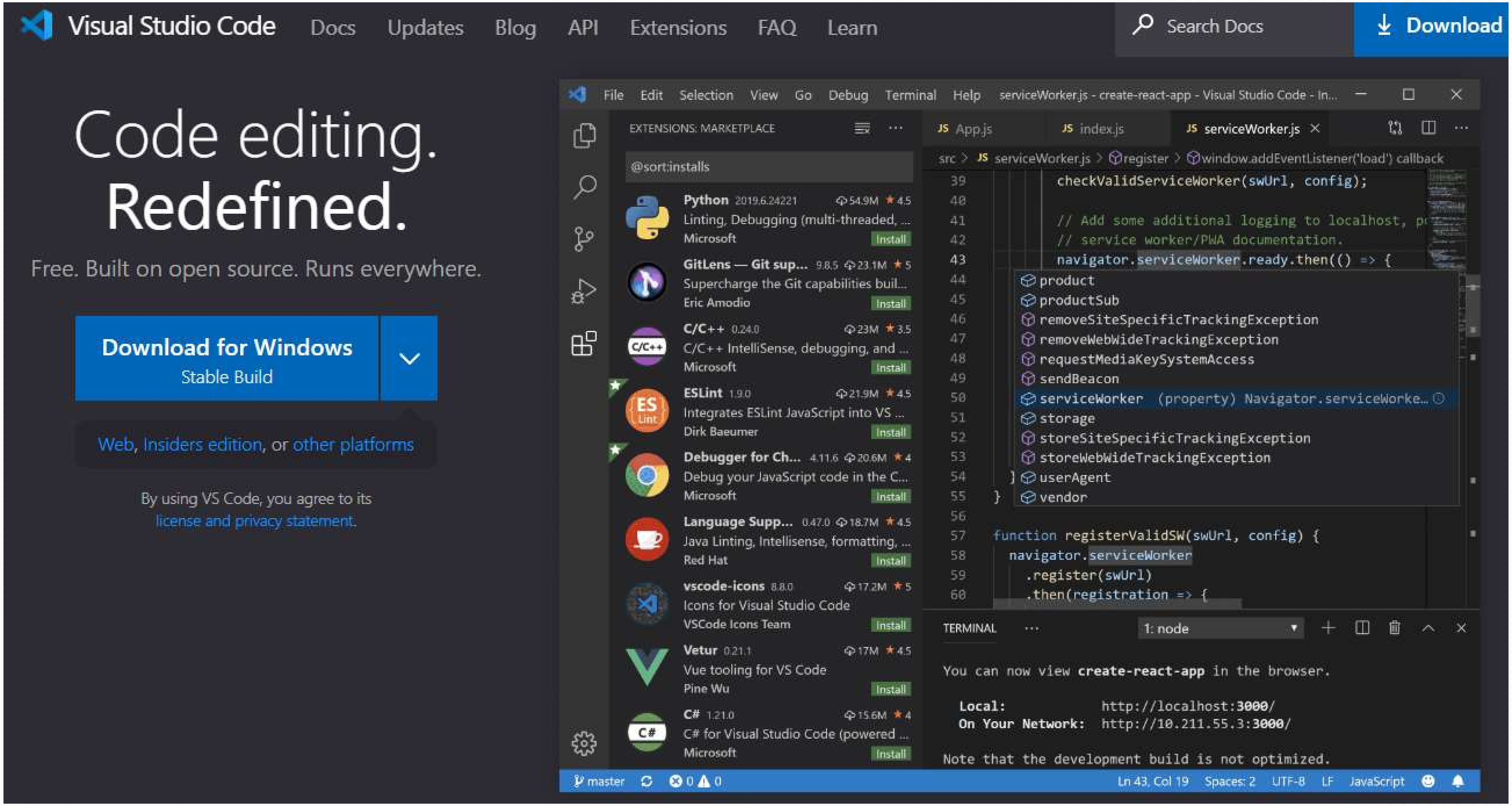
Please star us on GitHub. If you use Julia in your research, please cite us. If possible, do consider sponsoring us.

Current stable release: v1.8.0 (August 17, 2022)

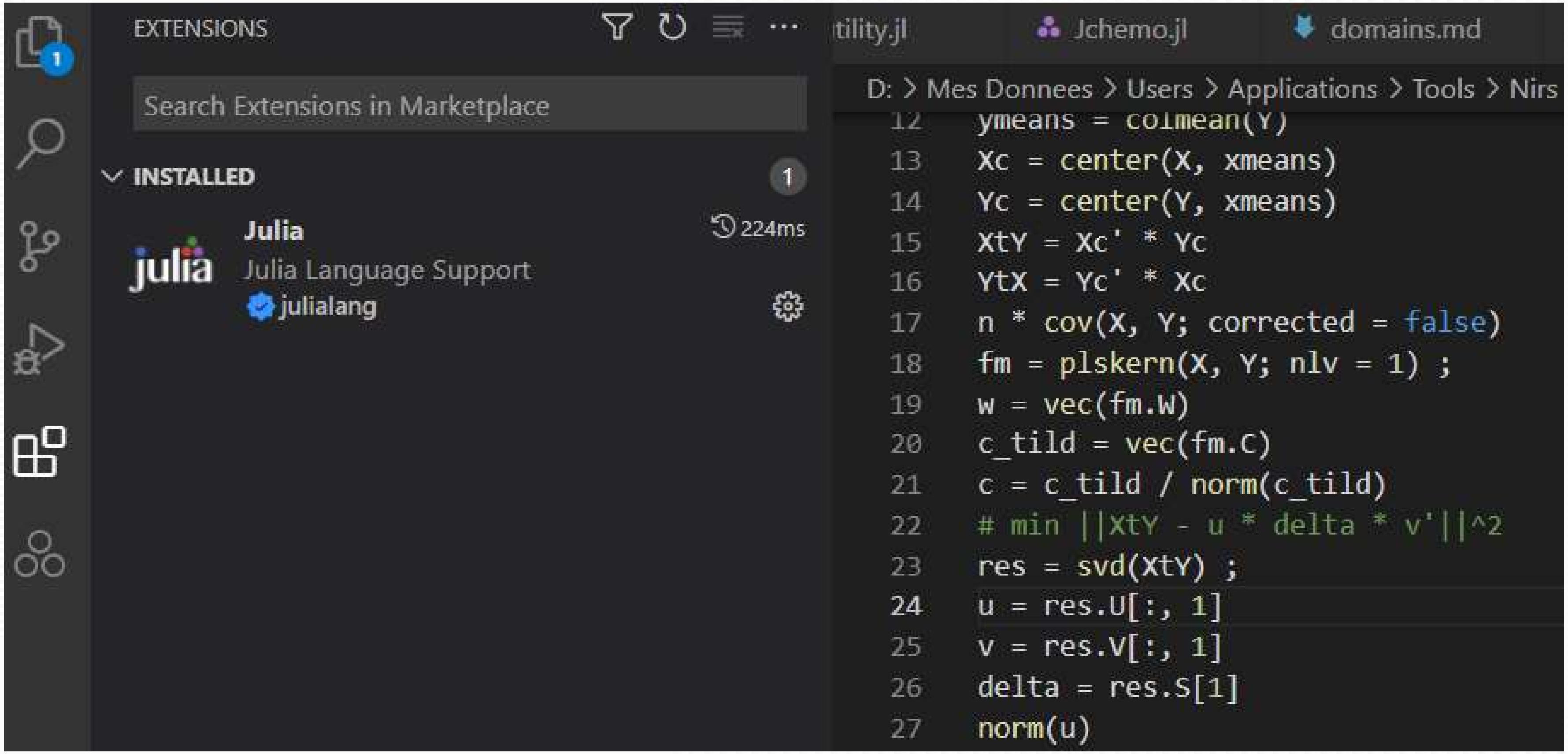
Checksums for this release are available in both MD5 and SHA256 formats.

Windows [help]	64-bit (installer), 64-bit (portable)	32-bit (installer), 32-bit (portable)	
macOS x86 (Intel or Rosetta) [help]	64-bit (.dmg), 64-bit (.tar.gz)		
macOS ARM (M-series Processor) [help]	64-bit		
Generic Linux on x86 [help]	64-bit (glibc) (GPG), 64-bit (musl)[1] (GPG)	32-bit (GPG)	
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)		
Generic FreeBSD on x86 [help]	64-bit (GPG)		
Source	Tarball (GPG)	Tarball with dependencies (GPG)	GitHub

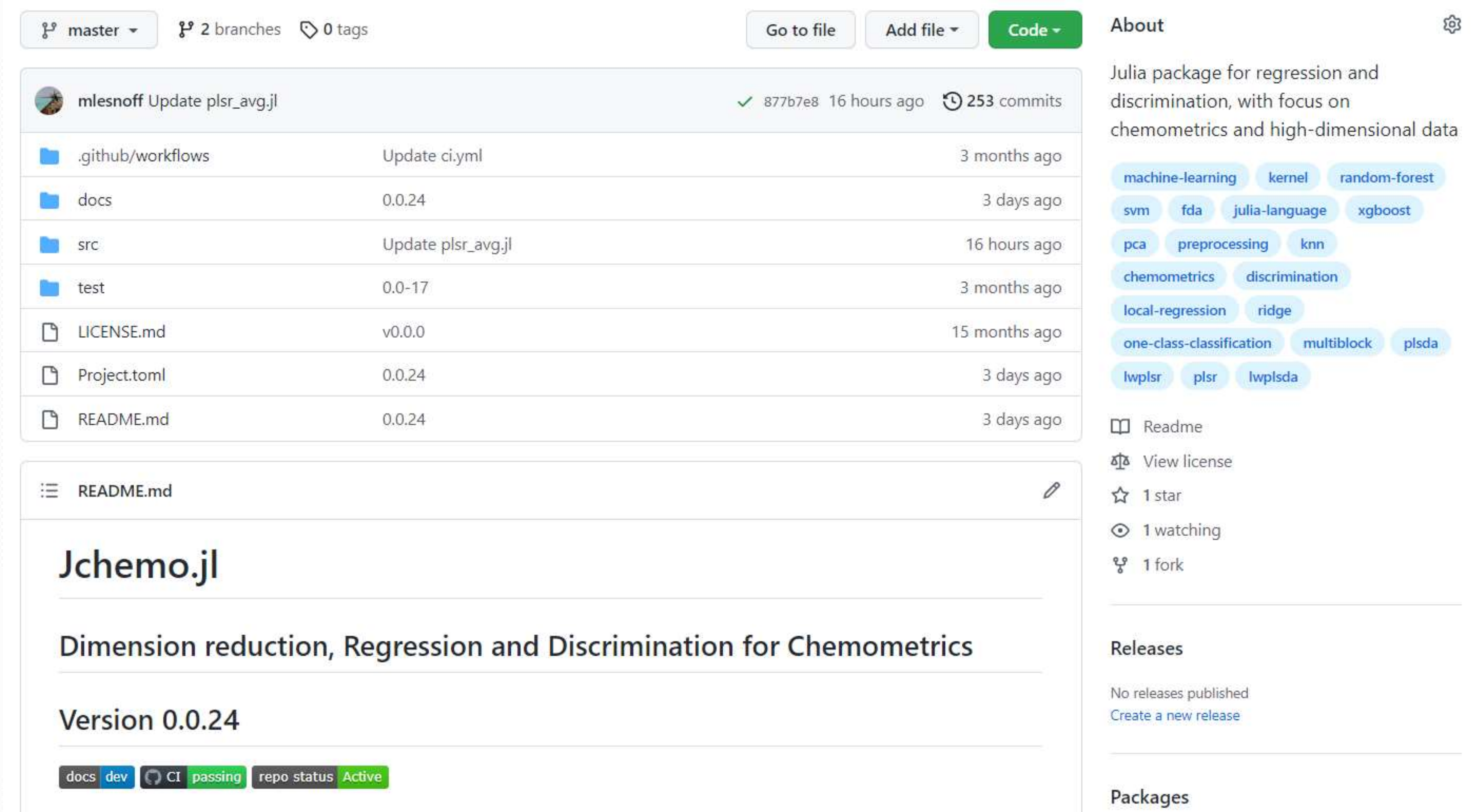
https://code.visualstudio.com



Install Julia extension under VsCode



Package Jchemo
https://github.com/mlesnoff/Jchemo.jl



Documentation

Jchemo.jl

Search docs

Home

Available methods

- PCA
- RANDOM PROJECTIONS
- REGRESSION
- DISCRIMINATION ANALYSIS (DA)
- TUNING MODELS
- DATA MANAGEMENT
- PLOTTING
- UTILITIES

Index of functions

News

Version dev

- **rpmat_li** Sparse random projection matrix

REGRESSION

Linear models

Multiple linear regression (MLR)

- **mlr** QR algorithm
- **mlrchol** Normal equations and Choleski factorization
- **mlrpinv** Pseudo-inverse
- **mlrpinv_n** Normal equations and pseudo-inverse
- **mlrvec** Simple linear regression (Univariate x)

Anova

- **aov1** One factor ANOVA

Partial least squares (PLSR)

- **plskern** "Improved kernel #1" *Dayal & McGregor 1997*
- **plsnipals** NIPALS
- **plsrosa** ROSA *Liland et al. 2016*
- **plssimp** SIMPLS *de Jong 1993*

Variants

- **cglsr** Conjugate gradient for the least squares normal equations (CGLS)
- **covselr** MLR on variables selected from partial correlation or covariance (Covsel)
- **pcr** SVD factorization

Non linear

Benchmark

```
n = 10^6 ; p = 500 ; q = 10
X = rand(n, p)
Y = rand(n, q)
nlv = 25

@time plskern(X, Y[:, 1]; nlv = nlv) ;

 7.1 seconds (50 allocations: 4.001 GiB, 3.45% gc time)

@time plskern(X, Y; nlv = nlv) ;

 7.3 seconds (9.76 k allocations: 4.130 GiB, 7.71% gc time, 0.17% compilation time)
```

Examples of syntax

Fitting a model

```
using Jchemo

n = 150 ; p = 200 ; q = 2 ; m = 50
Xtrain = rand(n, p) ; Ytrain = rand(n, q) ;
Xtest = rand(m, p) ; Ytest = rand(m, q) ;

nlv = 5
fm = plskern(Xtrain, Ytrain; nlv = nlv) ;
pnames(fm)

summary(fm, Xtrain, Ytrain)

transform(fm, Xtest)
transform(fm, Xtest; nlv = 1)

coef(fm)
coef(fm; nlv = 2)

res = predict(fm, Xtest) ;
res.pred
rmsep(res.pred, Ytest)
mse(res.pred, Ytest)
```

Fictive data

Model fitting

LVs computations (projections)

b-coefs

Predictions

Error rates (scores)

Tuning models by grid search

Runs over all the combinations of the model parameters
and compute the error rate (RMSEP etc.) ⇒ Model selection

```
par1  = [.1 ; .5 ; 1]
par2 = ["a" ; "e"]
```

par1	par2
0.1	"a"
0.5	"a"
1.0	"a“
0.1	“e"
0.5	"e“
1.0	"e"

Function **mpar** builds the grid

```
pars = mpar(par1 = par1, par2 = par2)
```

- Functions gridscore, gridscorelv, gridscorelb

Train = Cal + Val

```
gridscore(Xcal, Ycal, Xval, Yval ; ....)
```

- Functions gridcv, gridcvlv, gridcvlb

Cross-validation

```
segm = segmkf(ntrain, 5; rep = 5)        # Replicated K-fold cross-validation
#segm = segmts(ntrain, 30; rep = 5)   # Replicated test-set validation

gridcv(Xtrain, Ytrain ; ....)
```

- **With gridscore**

```julia
using Jchemo, CairoMakie

n = 150 ; p = 200 ; m = 50
Xtrain = rand(n, p) ; ytrain = rand(n)
Xval = rand(m, p) ; yval = rand(m)

nlv = 0:10
pars = mpar(nlv = nlv)
res = gridscore(
    Xtrain, ytrain, Xval, yval;
    score = rmsep, fun = plskern, pars = pars)

plotgrid(res.nlv, res.y1,
    xlabel = "Nb. LVs", ylabel = "RMSEP").f

u = findall(res.y1 .== minimum(res.y1))[1]
res[u, :]
fm = plskern(Xval, yval; nlv = res.nlv[u]) ;
res = Jchemo.predict(fm, Xval) ;
rmsep(res.pred, yval)

## For PLSR models, using gridscorelv is much faster than gridscore!!!

res = gridscorelv(
    Xtrain, ytrain, Xval, yval;
    score = rmsep, fun = plskern, nlv = nlv)
```

- **With gridcv**

```
using Jchemo

n = 150 ; p = 200 ; m = 50
Xtrain = rand(n, p) ; ytrain = rand(n)
Xval = rand(m, p) ; yval = rand(m)

segm = segmkf(n, 5; rep = 5)     # Replicated K-fold cross-validation
#segm = segmts(n, 30; rep = 5)   # Replicated test-set validation

nlv = 0:10
pars = mpar(nlv = nlv)
zres = gridcv(
    Xtrain, ytrain; segm,
    score = rmsep, fun = plskern, pars = pars) ;
pnames(zres)
res = zres.res

plotgrid(res.nlv, res.y1,
    xlabel = "Nb. LVs", ylabel = "RMSEP").f

u = findall(res.y1 .== minimum(res.y1))[1]
res[u, :]
fm = plskern(Xval, yval; nlv = res.nlv[u]) ;
res = Jchemo.predict(fm, Xval) ;
rmsep(res.pred, yval)

## For PLSR models, using gridcvlv is much faster than gridcv!!!

zres = gridcvlv(
    Xtrain, ytrain; segm,
    score = rmsep, fun = plskern, nlv = nlv) ;
zres.res
```

Pipeline function lwplsr

Data
X, Y

Pre-processing

Xp, Y

x_{new}

Compute dissimilarities between x_{new} and rows of Xp to find the k nearest neighbors of x_{new}

Options:

(1) Xp
- Euclidean

or

(2) Global PLSR Xp, y ➔ T
- Euclidean
- Mahalanobis

parameters
- nlvdis
- metric = “eucl”, “mahal”

Set of k nearest neighbors
$= Xp[x_{new}]$
$d = \{d_1, \ldots, d_k\}$

parameters
- k

Weight function
Compute the weights from d

Weights
$w = \{w_1, \ldots, w_k\}$

parameters
- h

Model building WPLSR
$Xp[x_{new}]$, $y[x_{new}]$, w

Prediction y_{new}

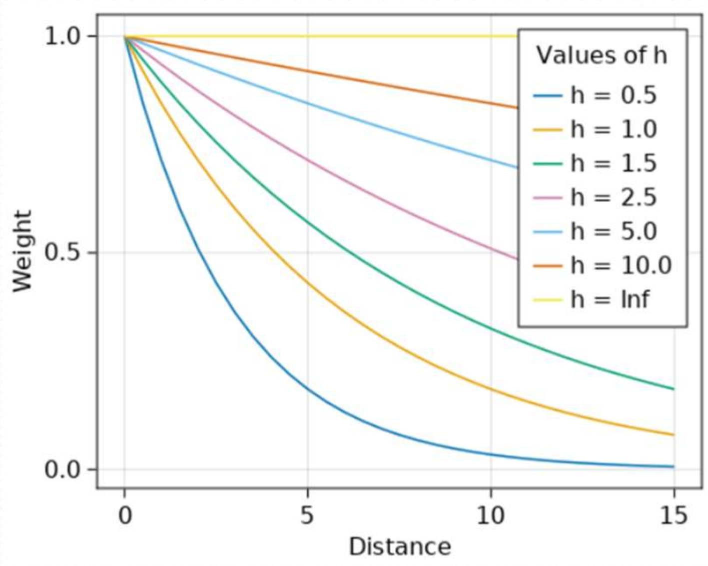
SYNTAX

```
nlvdis = 20 ; metric = "mahal"        # Mahalanobis distance on 20 global PLS scores
h = 1 ; k = 100                       # k neighbors, weight function h = 1
nlv = 15                              # Nb. LVs in local models

fm = lwplsr(Xtrain, ytrain; nlvdis = nlvdis,
            metric = metric, h = h, k = k, nlv = nlv) ;

pred = predict(fm, Xtest).pred

rmsep(pred, ytest)
```



- GRIDSCORE

```
nlvdis = [25] ; metric = ["mahal"]
h = [1; 2; 5] ; k = [100; 250; 500]

pars = mpar(nlvdis = nlvdis, metric = metric, h = h, k = k)        # Build the grid
nlv = 0:25

res = gridscorelv(Xcal, ycal, Xval, yval; score = rmsep,
    fun = lwplsr, nlv = nlv, pars = pars)
u = findall(res.y1 .== minimum(res.y1))[1]

fm = lwplsr(Xtrain, ytrain; nlvdis = res.nlvdis[u],
    metric = res.metric[u], h = res.h[u], k = res.k[u],
    nlv = res.nlv[u], verbose = true) ;

pred = predict(fm, Xtest).pred

rmsep(pred, ytest)
```

- GRIDCV

```
nlvdis = [25] ; metric = ["mahal"]
h = [1; 2; 5] ; k = [100; 250; 500]

pars = mpar(nlvdis = nlvdis, metric = metric, h = h, k = k)       # Build the grid
nlv = 0:25

m = 100 ; segm = segmts(ntrain, m; rep = 6)      # Test-set CV
#K = 3 ; segm = segmkf(ntrain, K; rep = 2)       # K-fold CV

res = gridcvlv(Xtrain, ytrain; segm = segm, score = rmsep,
    fun = lwplsr, nlv = nlv, pars = pars).res
u = findall(res.y1 .== minimum(res.y1))[1]
res[u, :]

fm = lwplsr(Xtrain, ytrain; nlvdis = res.nlvdis[u],
    metric = res.metric[u], h = res.h[u], k = res.k[u],
    nlv = res.nlv[u]) ;

pred = predict(fm, Xtest).pred

rmsep(pred, ytest)
```

Practical work

forages2

NIRS data from mixed forages (dried and grounded): stems, leaves etc. Origin: mainly tropical African areas. FOSS NiRSystem Instruments 1100-2498 nm (step = 2 nm). Data being private, spectra have been preprocessed with Savitzky-Golay (d = 2). Response variables:

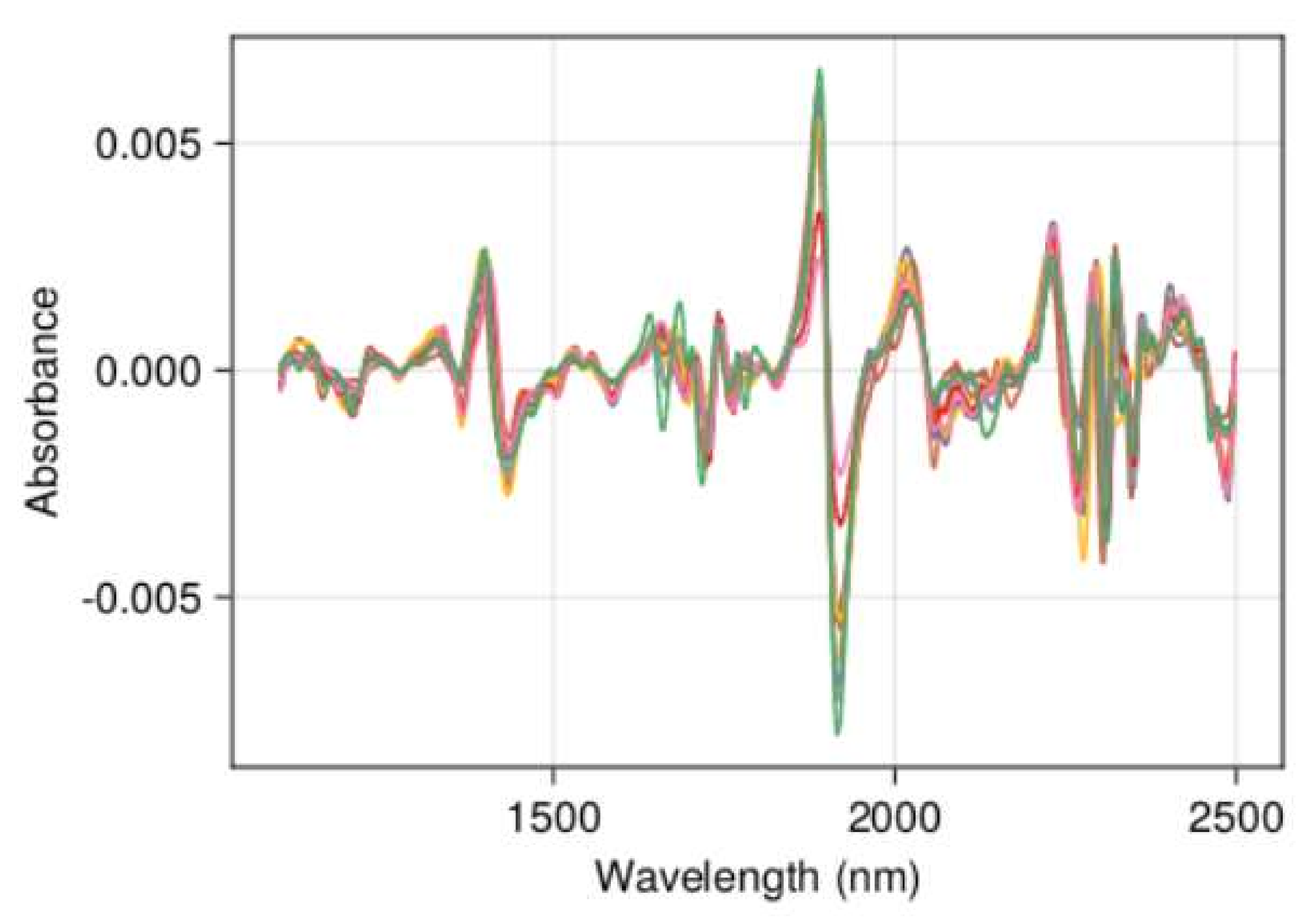
- DM: dry matter content
- NDF: fibers content

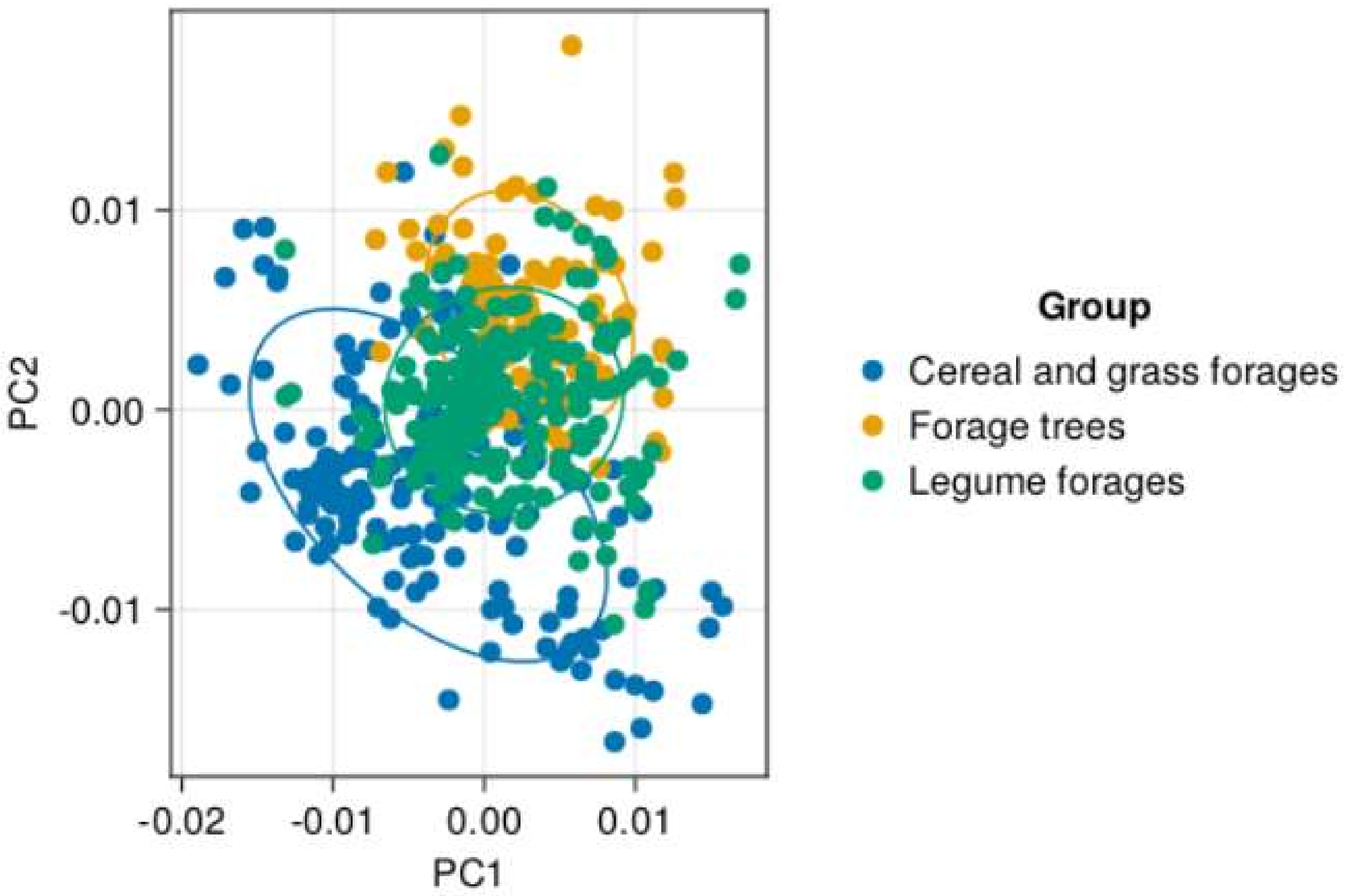
Source: CIRAD, Selmet research unit

```
julia> Y = dat.Y
485×4 DataFrame
 Row │ dm         ndf        typ                        test
     │ Float64?   Float64?   String                     Int64
─────┼────────────────────────────────────────────────────────
   1 │  92.23      37.58     Legume forages                1
   2 │  93.26      49.6462   Legume forages                0
   3 │  92.9       63.2939   Forage trees                  0
   4 │  94.44      51.6413   Cereal and grass forages      0
   5 │  93.16      46.5114   Legume forages                0
   6 │  93.44      40.8176   Legume forages                0
  ⋮       ⋮          ⋮                 ⋮                  ⋮
 481 │  93.7303    55.5358   Legume forages                1
 482 │  93.2967    68.9812   Legume forages                0
 483 │  93.1954    39.1038   Legume forages                0
 484 │  93.17      48.8868   Legume forages                0
 485 │  93.2842    34.6457   Legume forages                1
                                            474 rows omitted
```

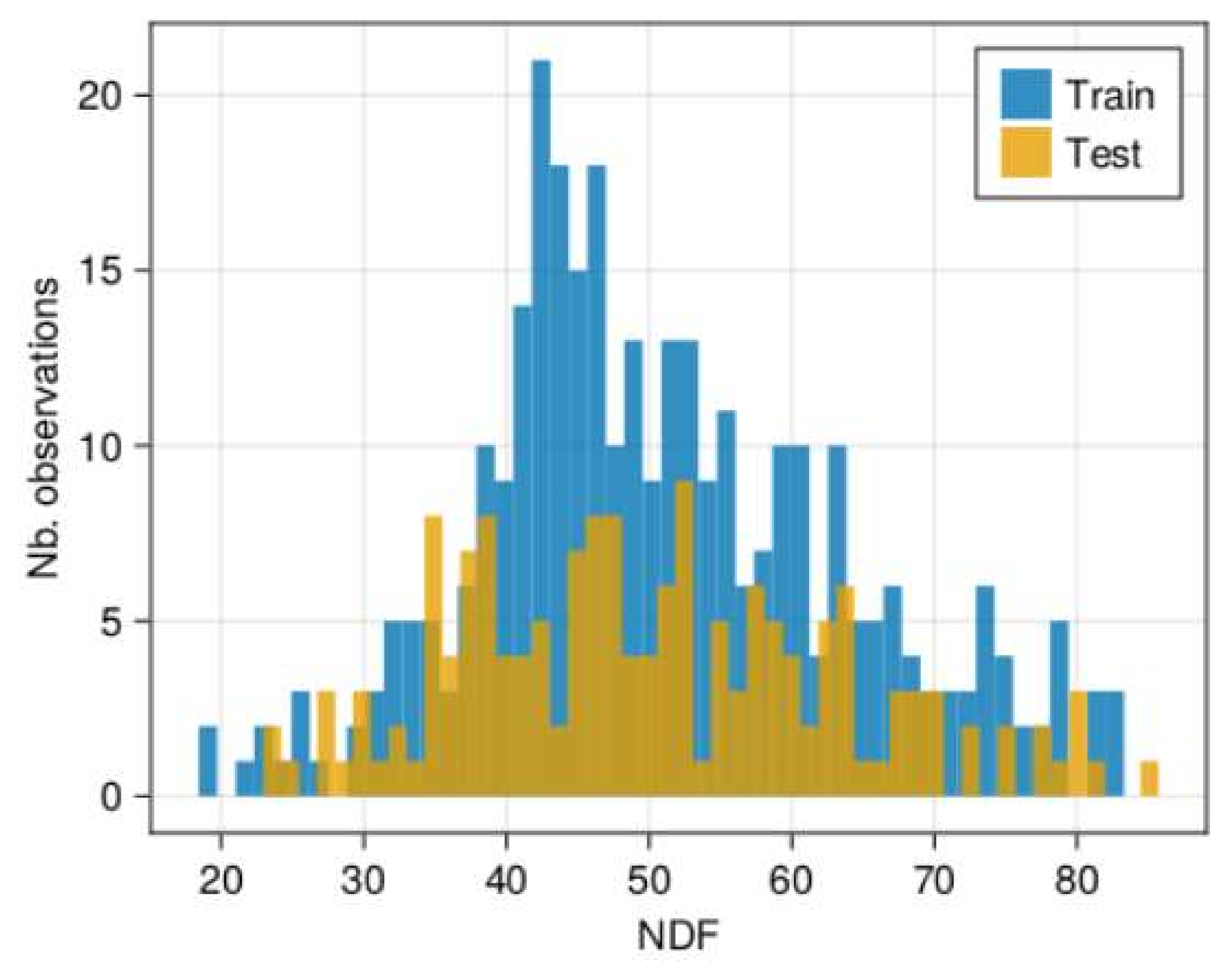
```
julia> X = dat.X
485×700 DataFrame
 Row │ 1100          1102          1104          1106
     │ Float64       Float64       Float64       Float64
─────┼─────────────────────────────────────────────────────────
   1 │ -0.000231591  -0.000175945  -8.48176e-5    2.05217e-5
   2 │ -9.66352e-5   -3.30928e-5    5.64966e-5    0.000154135
   3 │ -0.000131769  -7.8398e-5     7.92223e-7    8.90044e-5
   4 │ -5.27109e-5   -1.38838e-5    3.73562e-5    8.87785e-5
   5 │ -4.13284e-5    1.46607e-5    8.7976e-5     0.000161825
   6 │ -0.000562333  -0.000451182  -0.000256137  -1.91839e-5
  ⋮        ⋮             ⋮             ⋮             ⋮
 481 │ -0.000790809  -0.000707089  -0.000525597  -0.000272477
 482 │ -0.000866662  -0.000767647  -0.000559022  -0.000274144
 483 │ -0.000788774  -0.000703783  -0.000521761  -0.000270395
 484 │ -0.00033791   -0.00028799   -0.000190344  -5.73166e-5
 485 │ -0.000790261  -0.000703107  -0.000519605  -0.000268478
```

SNV + SAV-GOL d=2




0.01
0.00
-0.01
PC2
-0.02
-0.01
0.00
0.01
PC1
Group
Cereal and grass forages
Forage trees
Legume forages

Predict NDF



Script files

forages2_descri.jl

gridscore_lwplsr_forages2.jl

gridcv_lwplsr_forages2.jl