Package `widals`: Fun with `fun.load()`

Dave Zes

December 7, 2019

1 Intro

`widals` Users are encouraged to create their own `fun.load` functions to suit particular situations.

Here, we will create a function called `fun.load.widals.ab`; very close kin to the stock `fun.load.widals.a`, except that it includes an *extra* ALS stage that will run in series with WIDALS. Since the stock WIDALS functions, e.g., `widals.snow`, fit/predict using an initial ALS stage followed in series with the stochastic adjustment, the `fun.load.widals.ab` solving method could be notated as

ALS1 $\longrightarrow$ ALS2 $\longrightarrow$ Crispify

When using the Ozone data, there is no real evidence that the additional ALS stage — as constructed — offers any improvement. In fact, the RMSE increases slightly. The advantage of including additional bases-transform covariates is more commonly realized when many sensors are present. In Section 3, where we simulate a composite system, we will find a small reduction in CV RMSE over the single stage ALS.

2 WIDALS Review

We'll work with the Californis Ozone demonstration data set obtained from CARB [2].

Important: It is intended that the User run through *all* the code in this Section.

Ready data and covariates:

```
options(stringsAsFactors=FALSE)
library(snowfall)
k.cpus <- 2 #### set the number of cpus for snowfall
library(widals)
data(O3)
Z.all <- as.matrix(O3$Z)[366:730, ]
locs.all <- O3$locs[ , c(2,1)]
hsa.all <- O3$helevs/500
xdate <- rownames(Z.all)
tau <- nrow(Z.all)
n.all <- ncol(Z.all)
xgeodesic <- TRUE
Z <- Z.all
locs <- locs.all
n <- n.all
dateDate <- strptime(xdate, "%Y%m%d")
doy <- as.integer(format(dateDate, "%j"))
Ht <- cbind( sin(2*pi*doy/365), cos(2*pi*doy/365) )
Hs.all <- cbind(matrix(1, nrow=n.all), hsa.all)
Hisa.ls <- H.Earth.solar(locs[ , 2], locs[ , 1], dateDate)
Hst.ls.all2 <- list()
for(tt in 1:tau) {
      Hst.ls.all2[[tt]] <- cbind(Hisa.ls[[tt]], Hisa.ls[[tt]]*hsa.all)
      colnames(Hst.ls.all2[[tt]]) <- c("ISA", "ISAxElev")
  }
Hst.ls <- Hst.ls.all2
Hs <- Hs.all
Ht.original <- Ht
train.rng <- 30:tau
test.rng <- train.rng
k.glob <- 10
```

```
run.parallel <- TRUE
```

Assign the necessary parameters:

```
FUN.source <- fun.load.widals.a
d.alpha.lower.limit <- 0
rho.upper.limit <- 100
rgr.lower.limit <- 10^(-7)
GP <- c(1/10, 1, 0.01, 3, 1)
############# pseudo cross-validation
rm.ndx <- 1:n
cv <- -2
lags <- c(0)
b.lag <- -1
sds.mx <- seq(2, 0.01, length=k.glob) * matrix(1, k.glob, length(GP))
ltco <- -10
stnd.d <- TRUE
FUN.GP <- NULL
sfInit(TRUE, k.cpus)
FUN.source()
set.seed(99999)
MSS.snow(FUN.source, NA, p.ndx.ls, f.d, sds.mx=sds.mx,
  k.glob, k.loc.coef=7, X = NULL)
sfStop()
#### 11.90536
```

```
k.glob <- 10
FUN.source <- fun.load.widals.a
GP <- c(1/10, 1, 0.01, 3, 1)
######### true spacial cross-validation
rm.ndx <- create.rm.ndx.ls(n, 14)
cv <- 2
lags <- c(0)
b.lag <- 0
sds.mx <- seq(2, 0.01, length=k.glob) * matrix(1, k.glob, length(GP))
ltco <- -10
FUN.GP <- NULL
sfInit(TRUE, k.cpus)
```

```
FUN.source()
set.seed(99999)
MSS.snow(FUN.source, NA, p.ndx.ls, f.d, sds.mx=sds.mx,
  k.glob, k.loc.coef=7, X=NULL)
sfStop()
#### 12.11686
```

We will now invite the requirement of an additional ALS stage by introducing spacial covariates in the form of a bases expansion (generally, see [6, 4, 1], specifically, [3]). For this we'll call upon the excellent package `LatticeKrig` by Professor Nychka and friends [5].

```
library(LatticeKrig)
xxb <- 4
yyb <- 4
center <- as.matrix(expand.grid( seq(0, 1, length=xxb), seq(0, 1, length=yyb)))
###### run together
ffunit <- function(x) { return( (x-min(x)) / (max(x)-min(x)) ) }
locs.unit <- apply(locs, 2, ffunit)
locs.unit <- matrix(as.vector(locs.unit), ncol=2)
###### run togther

xPHI <- Radial.basis(as.matrix(locs.unit), center, 0.5)
Hs.lkrig <- matrix(NA, nrow(locs), xxb*yyb)
for(i in 1:nrow(locs)) {
      Hs.lkrig[i, ] <- xPHI[i]
  }
Hs.lkrig <- Hs.lkrig[ , -which( apply(Hs.lkrig, 2, sum) < 0.1 ), drop=FALSE ]
```

We now create two sets of covariates.

The first (prefixed `XA.`) will house the usual goods: constant term, elevation, seasonal, ISA, and the ISA-elevation interaction.

The second (prefixed `XB.`) will house the bases transform.

```
XA.Hs <- cbind(rep(1,n), hsa.all)
XA.Ht <- Ht.original
XA.Hst.ls <- Hst.ls
```

```
Hst.sumup(XA.Hst.ls, XA.Hs, XA.Ht)
XB.Hs <- 10*Hs.lkrig
XB.Ht <- NULL
XB.Hst.ls <- NULL
Hst.sumup(XB.Hst.ls, XB.Hs, XB.Ht)
```

Our custom function:

```
fun.load.widals.ab <- function() {

      if( run.parallel ) {
          sfExport("Z", "XA.Hs", "XA.Ht", "XA.Hst.ls", "XB.Hs", "XB.Ht", "XB.Hst.ls",
          "locs", "lags", "b.lag", "cv", "rm.ndx", "train.rng", "test.rng", "xgeodesic",
          "ltco", "stnd.d")
          suppressWarnings(sfLibrary(widals))
      }

      if( length(lags) == 1 & lags[1] == 0 ) {
          p.ndx.ls <- list( c(1,2), c(3,4), c(5,7) )
      } else {
          p.ndx.ls <- list( c(1,2), c(3,4), c(5,6,7) )
      }
      assign( "p.ndx.ls", p.ndx.ls, pos=globalenv() )

      f.d <- list( dlog.norm, dlog.norm, dlog.norm, dlog.norm, dlog.norm,
      dlog.norm, dlog.norm )
      assign( "f.d", f.d, pos=globalenv() )

      FUN.MH <- function(jj, GP.mx, X) {

          if(cv==2) { ZhalsA <- Hals.fastcv.snow(jj, rm.ndx, Z, XA.Hs, XA.Ht, XA.Hst.ls, GP.mx) }
          if(cv==-2) { ZhalsA <- Hals.snow(jj, Z, XA.Hs, XA.Ht, XA.Hst.ls, b.lag, GP.mx) }
          Z.resids.A <- Z - ZhalsA

          Z.wid <- widals.snow(jj, rm.ndx=rm.ndx, Z=Z.resids.A, Hs=XB.Hs, Ht=XB.Ht, Hst.ls=XB.Hst.ls,
          locs=locs, lags=lags, b.lag=b.lag, cv=cv, geodesic=xgeodesic,
          wrap.around=NULL, GP.mx[ , c(3:7), drop=FALSE], stnd.d=stnd.d, ltco=ltco)

          Z.wid <- Z.wid + ZhalsA

          if( min(Z, na.rm=TRUE) >= 0 ) { Z.wid[ Z.wid < 0 ] <- 0 } ####### DZ EDIT

          Z.wid <- Z.clean.up(Z.wid)

          resids <- Z[ , unlist(rm.ndx)] - Z.wid[ , unlist(rm.ndx)]
          our.cost <- sqrt( mean( resids[ train.rng, ]^2 ) )

          if( is.nan(our.cost) ) { our.cost <- Inf }

          return( our.cost )
      }
      assign( "FUN.MH", FUN.MH, pos=globalenv() )

      #FUN.GP <- NULL
      FUN.GP <- function(GP.mx) {
          GP.mx[ GP.mx[ , 1] > rho.upper.limit, 1 ] <- rho.upper.limit
          GP.mx[ GP.mx[ , 2] < rgr.lower.limit, 2 ] <- rgr.lower.limit
          GP.mx[ GP.mx[ , 2] > rgr.upper.limit, 2 ] <- rgr.upper.limit

          GP.mx[ GP.mx[ , 3] > rho.upper.limit, 3 ] <- rho.upper.limit
          GP.mx[ GP.mx[ , 4] < rgr.lower.limit, 4 ] <- rgr.lower.limit
```

```
        GP.mx[ GP.mx[ , 4] > rgr.upper.limit, 4 ] <- rgr.upper.limit

        GP.mx[ GP.mx[ , 5] < d.alpha.lower.limit, 5 ] <- d.alpha.lower.limit
        xperm <- order(GP.mx[ , 5,  drop=FALSE])
        GP.mx <- GP.mx[ xperm,  ,  drop=FALSE]
        return(GP.mx)
    }
    assign( "FUN.GP", FUN.GP, pos=globalenv() )

    FUN.I <- function(envmh, X) {
        cat( "Improvement ---> ", envmh$current.best, " ---- " , envmh$GP, "\n" )
    }
    assign( "FUN.I", FUN.I, pos=globalenv() )

    FUN.EXIT <- function(envmh, X) {

        GP.mx <- matrix(envmh$GP, 1, length(envmh$GP))

        if(cv==2) { ZhalsA <- Hals.fastcv.snow(1, rm.ndx, Z, XA.Hs, XA.Ht, XA.Hst.ls, GP.mx) }
        if(cv==-2) { ZhalsA <- Hals.snow(1, Z, XA.Hs, XA.Ht, XA.Hst.ls, b.lag, GP.mx) }

        Z.resids.A <- Z - ZhalsA

        Z.wid <- widals.snow(1, rm.ndx=rm.ndx, Z=Z.resids.A, Hs=XB.Hs, Ht=XB.Ht, Hst.ls=XB.Hst.ls,
        locs=locs, lags=lags, b.lag=b.lag, cv=cv, geodesic=xgeodesic,
        wrap.around=NULL, GP.mx[ , c(3:7), drop=FALSE], stnd.d=stnd.d, ltco=ltco)

        Z.wid <- Z.wid + ZhalsA

        if( min(Z, na.rm=TRUE) >= 0 ) { Z.wid[ Z.wid < 0 ] <- 0 } ############ DZ EDIT

        assign( "Z.wid", Z.wid, envir=globalenv() )

        Z.wid <- Z.clean.up(Z.wid)

        resids <- Z[ , unlist(rm.ndx)] - Z.wid[ , unlist(rm.ndx)]
        our.cost <- sqrt( mean( resids[ test.rng, ]^2 ) )

        if( is.nan(our.cost) ) { our.cost <- Inf }

        cat( envmh$GP, " -- ", our.cost,   "\n" )

        assign( "our.cost", our.cost, pos=globalenv() )
        assign( "GP", envmh$GP, pos=globalenv() )
        cat( paste( "GP <- c(", paste(format(GP,digits=5), collapse=", "), ") ### ",
        format(our.cost, width=6), "\n", sep="" ) )
    }
    assign( "FUN.EXIT", FUN.EXIT, pos=globalenv() )
}
```

Pseudo CV:

```
GP <- c(1/10, 1, 1/10, 1,       5, 3, 1)
```

```
sds.mx <- seq(2, 0.01, length=k.glob) * matrix(1, k.glob, length(GP))
ltco <- -10
stnd.d <- TRUE
rm.ndx <- I(1:n)
cv <- -2
lags <- c(0)
b.lag <- -1
d.alpha.lower.limit <- 0
rho.upper.limit <- 100
rgr.lower.limit <- 10^(-7)
rgr.upper.limit <- 500
FUN.GP <- NULL
sfInit(TRUE, k.cpus)
FUN.source <- fun.load.widals.ab
FUN.source()
set.seed(9999)
MSS.snow(FUN.source, NA, p.ndx.ls, f.d, sds.mx=sds.mx,
  k.glob, k.loc.coef=7, X = NULL)
sfStop()
#### 11.5234
```

Real sitewise CV:

```
GP <- c(1/10, 1, 1/10, 1,      0.5, 3, 1)
sds.mx <- seq(2, 0.01, length=k.glob) * matrix(1, k.glob, length(GP))
ltco <- -10
stnd.d <- TRUE
rm.ndx <- create.rm.ndx.ls(n, 14)
cv <- 2
lags <- c(0)
b.lag <- 0
d.alpha.lower.limit <- 0
rho.upper.limit <- 100
rgr.lower.limit <- 10^(-7)
rgr.upper.limit <- 500
FUN.GP <- NULL
sfInit(TRUE, k.cpus)
FUN.source <- fun.load.widals.ab
FUN.source()
```

```
set.seed(9999)
MSS.snow(FUN.source, NA, p.ndx.ls, f.d, sds.mx=sds.mx,
  k.glob, k.loc.coef=9, X = NULL)
sfStop()
```

3 Simulation

```
options(stringsAsFactors=FALSE)
k.cpus <- 2 #### set the number of cpus for snowfall
library(widals)
tau <- 210
n.all <- 300
set.seed(77777)
locs.all <- cbind(runif(n.all), runif(n.all))
D.mx <- distance(locs.all, locs.all, FALSE)
Q <- 0.03*exp(-2*D.mx)
F <- 0.99
R <- diag(1, n.all)
beta0 <- rep(0, n.all)
H <- diag(1, n.all)
xsssim <- SS.sim(F, H, Q, R, length.out=tau, beta0=beta0)
Y1 <- xsssim$Y
Hst.ss <- list()
for(tt in 1:tau) {
  Hst.ss[[tt]] <- cbind( rep(sin(tt*2*pi/tau), n.all), rep(cos(tt*2*pi/tau), n.all) )
  colnames(Hst.ss[[tt]]) <- c("sinet", "cosinet")
  }
Ht.original <- cbind( sin((1:tau)*2*pi/tau), cos((1:tau)*2*pi/tau) )
Q2 <- diag(0.03, ncol(Hst.ss[[1]]))
F2 <- 0.99
beta20 <- rep(0, ncol(Hst.ss[[1]]))
R2 <- 1*exp(-3*D.mx) + diag(0.001, n.all)
xsssim2 <- SS.sim.tv(F2, Hst.ss, Q2, R, length.out=tau, beta0=beta20)
Z2 <- xsssim2$Z
Z.all <- Y1 + Z2
#################### plot
z.min <- min(Z.all)
for(tt in 1:tau) {
```

```
    plot(locs.all, cex=(Z.all[ tt, ]-z.min)*0.3,   main=tt)
    Sys.sleep(0.1)
  }
```

Just using temporal covariates:

```
train.rng <- 30:tau
test.rng <- train.rng
xgeodesic <- FALSE
Z <- Z.all
locs <- locs.all
n <- n.all
Ht <- Ht.original
Hs <- matrix(1, nrow=n)
Hst.ls <- NULL
rm.ndx <- create.rm.ndx.ls(n, 14)
k.glob <- 10
run.parallel <- TRUE
FUN.source <- fun.load.widals.a
d.alpha.lower.limit <- 0
rho.upper.limit <- 100
rgr.lower.limit <- 10^(-7)
GP <- c(1/10, 1, 0.01, 3, 1)
cv <- 2
lags <- c(0)
b.lag <- 0
sds.mx <- seq(2, 0.01, length=k.glob) * matrix(1, k.glob, length(GP))
ltco <- -10
stnd.d <- TRUE
FUN.GP <- NULL
sfInit(TRUE, k.cpus)
FUN.source()
set.seed(99999)
MSS.snow(FUN.source, NA, p.ndx.ls, f.d, sds.mx=sds.mx,
  k.glob, k.loc.coef=7, X = NULL)
sfStop()
```

Now, using a second ALS stage for the bases transformation:

```
library(LatticeKrig)
xxb <- 7
yyb <- 7
center <- as.matrix(expand.grid( seq( 0, 1, length=xxb), seq( 0, 1, length=yyb)))
###### run together
ffunit <- function(x) { return( (x-min(x)) / (max(x)-min(x)) ) }
locs.unit <- apply(locs, 2, ffunit)
locs.unit <- matrix(as.vector(locs.unit), ncol=2)
###### run togther

xPHI <- Radial.basis(as.matrix(locs.unit), center, 0.5)
Hs.lkrig <- matrix(NA, nrow(locs), xxb*yyb)
for(i in 1:nrow(locs)) {
      Hs.lkrig[i, ] <- xPHI[i]
  }
XA.Hs <- Hs
XA.Ht <- Ht.original
XA.Hst.ls <- NULL
Hst.sumup(XA.Hst.ls, XA.Hs, XA.Ht)
XB.Hs <- 10*Hs.lkrig
XB.Ht <- NULL
XB.Hst.ls <- NULL
Hst.sumup(XB.Hst.ls, XB.Hs, XB.Ht)
GP <- c(1/10, 1, 1/10, 1,      5, 3, 1)
sds.mx <- seq(2, 0.01, length=k.glob) * matrix(1, k.glob, length(GP))
rm.ndx <- create.rm.ndx.ls(n, 14)
cv <- 2
lags <- c(0)
b.lag <- 0
d.alpha.lower.limit <- 0
rho.upper.limit <- 100
rgr.lower.limit <- 10^(-7)
rgr.upper.limit <- 100
FUN.GP <- NULL
sfInit(TRUE, k.cpus)
FUN.source <- fun.load.widals.ab
FUN.source()
set.seed(9999)
MSS.snow(FUN.source, NA, p.ndx.ls, f.d, sds.mx=sds.mx,
```

```
  k.glob, k.loc.coef=7, X = NULL)
sfStop()
```

References

[1] F. Abramovich, T. C. Bailey, and T. Sapatinas. Wavelet analysis and its statistical applications. *The Statistician*, 49:1–29, 2000. Part 1.

[2] California Air Resources Board. California Air Resources Board DVD-ROM. http://www.arb.ca.gov/aqd/aqdcd/aqdcd.htm, 2011. [Online; last accessed 2012-11-04].

[3] N. Cressie and G. Johannesson. Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70, 2008.

[4] S. Efromovich. *Nonparametric Curve Estimation (Springer Series in Statistics)*, volume 1862. Springer U.S., New York, 1999.

[5] D. Nychka, D. Hammerling, S. Sain, and T. Lerud. *LatticeKrig: Multiresolution Kriging based on Markov random fields*, 2012. R package version 2.2.1.

[6] L. Wasserman. *All of Nonparametric Statistics (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.