Classification of Unlabeled observations

Emy Guilbault

28 March 2020

This document run through the different steps to use the functions ppmMixEngine and ppmLoopEngine for data classification. These functions as well as others are contained in the document functionTestsim.R. First, we load the different functions and package that we will need.

```
setwd("C:/Users/c3286500/Documents/SimulationProject1/Script")
source("functionTestsim160420-SH.R")

createConfusionMatrix <- function(act, pred) {
  pred <- pred[order(act)]
  act  <- act[order(act)]
  sapply(split(pred, act), tabulate, nbins=3)
}
```

We will also use some other functions from other packages, that we load here:

```
library(spatstat)
library(lattice)
library(sp)
library(maptools)
library(raster)
library(geostatsp)
library(rgdal)
library(latticeExtra)
library(caret)
library(rgeos)
library(scales)
library(gridExtra)
library(viridisLite)
```

Setting up environmental covariates

We define a grid and two vectors X and Y to define four envrionmental covariates.

```
# Set up data
# 1 #  Set up data.ppp, cov.list, ppmform and quads
# Generate XY grid
set.seed(10013)
XY = expand.grid(seq(0, 100, 1), seq(0, 100, 1))
X = XY[,1]
Y = XY[,2]

# Generate 2 covariates for PPM

v1 = (X - 30)^2 + (Y - 70)^2 - 0.5*X*Y
v2 = (X - 70)^2 + (Y - 60)^2 + 0.9*X*Y

v1 = -1*scale(v1)
v2 = -1*scale(v2)
```

```r
# Matrix of covariates
vmat = as.matrix(data.frame(1, v1, v1^2, v2, v2^2))
```

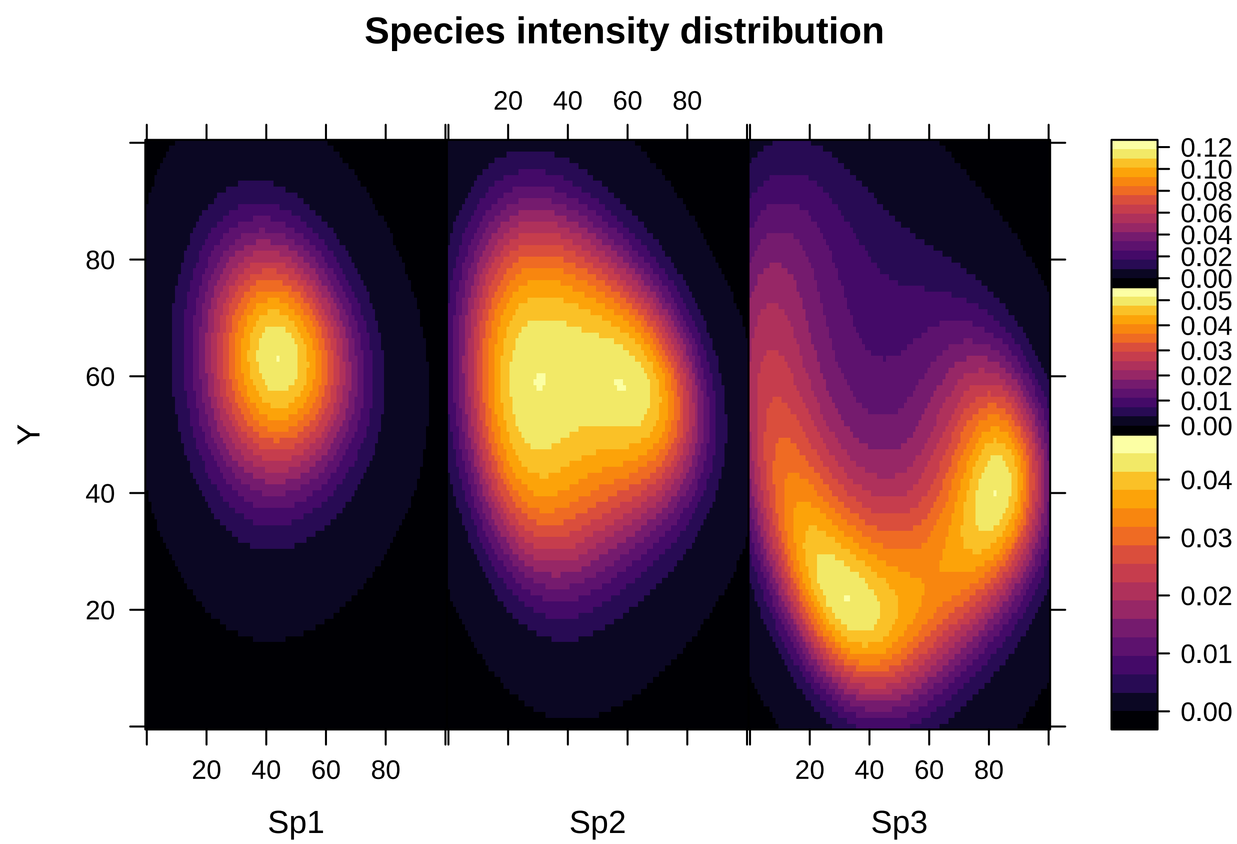
We define a list of envvironmental covariates images and the formula to fit the model.

```r
## Create list of coavriates
cov.list = list()
for (v in 1:4)
{
v.v = as.im(data.frame(x = X, y = Y, z = vmat[,(v + 1)]))
cov.list[[v]] = v.v
}
names(cov.list) = c("v1", "v1.2", "v2", "v2.2")

# set up model formula
cov.mat = vmat[,2:5]
ppmform = as.formula(paste("~", paste(colnames(cov.mat), collapse = "+")))
```

Displaying data points

We define the true species coefficients for three species and we show the species true intensity.

```r
# Generate true PPM coefficients based on linear and quadratic terms for the covariates
sp1_coef = c(-6.5, 4, -1, 2, -0.6)
sp1_int = exp(vmat %*% sp1_coef)

sp2_coef = c(-4.4, 1.8, -1, 1.5, -0.9)
sp2_int = exp(vmat %*% sp2_coef)

sp3_coef = c(-3.5, -0.5, -0.8, 1, -0.8)
sp3_int = exp(vmat %*% sp3_coef)

sp_int.list = list(sp1_int, sp2_int, sp3_int)

# Plot the intensities created

Lsp1 = levelplot(sp1_int ~ X + Y, col.regions=inferno(50))
Lsp2 = levelplot(sp2_int ~ X + Y, col.regions=inferno(50))
Lsp3 = levelplot(sp3_int ~ X + Y, col.regions=inferno(50))
comb_levObj <- c(Lsp1, Lsp2, Lsp3,
                 layout = c(3, 1), merge.legends = T)
update(comb_levObj,  xlab = c("Sp1", "Sp2", "Sp3"),
       main="Species intensity distribution")
```




```
# Create pixel images of intensity surfaces for spatstat
sp1_int_im = as.im(data.frame(x = X, y = Y, z = sp1_int))
sp2_int_im = as.im(data.frame(x = X, y = Y, z = sp2_int))
sp3_int_im = as.im(data.frame(x = X, y = Y, z = sp3_int))
```

We generate the true species pattern for each of the tree species.

```
# Simulate species patterns
sp1_sim = rpoispp(sp1_int_im)
sp2_sim = rpoispp(sp2_int_im)
sp3_sim = rpoispp(sp3_int_im)

sp1_sim
```

```
## Planar point pattern: 138 points
## window: rectangle = [-0.5, 100.5] x [-0.5, 100.5] units
```

```
sp2_sim
```

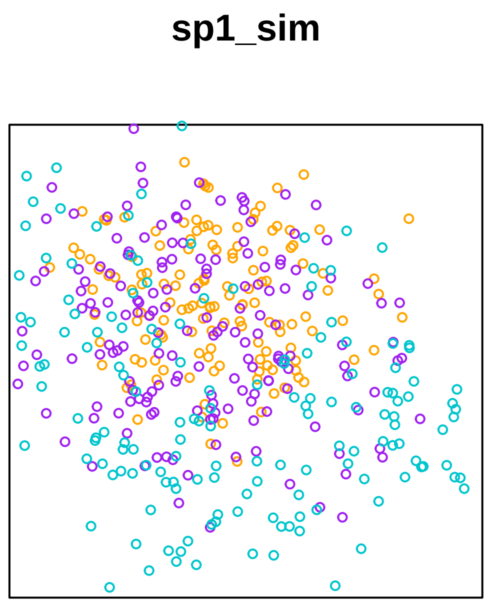
```
## Planar point pattern: 166 points
## window: rectangle = [-0.5, 100.5] x [-0.5, 100.5] units
```

```
sp3_sim
```

```
## Planar point pattern: 151 points
## window: rectangle = [-0.5, 100.5] x [-0.5, 100.5] units
```

```
plot(sp1_sim, cex = 0.6, col="white")
plot(sp1_sim, add = TRUE, col = "orange", cex = 0.6)
```

```r
plot(sp2_sim, add = TRUE, col = "purple", cex = 0.6)
plot(sp3_sim, add = TRUE, col = "turquoise3", cex = 0.6)
```



```r
sp_sim.list = list(sp1_sim, sp2_sim, sp3_sim)

# Look at the correlation between intensity surfaces
#all
cor1_2 = cor(as.vector(sp1_int), as.vector(sp2_int), use = "complete.obs")
cor1_3 = cor(as.vector(sp1_int), as.vector(sp3_int), use = "complete.obs")
cor2_3 = cor(as.vector(sp2_int), as.vector(sp3_int), use = "complete.obs")
```

Simulation test

We can call the function test sim and specify different parameters to give an example of the simulation steps. First, we set up the different parameters: which percentage of unknown point we want (we can have several values but the example show for 0.2), the number of simulations, the list of point patterns, the number of species and some other parameters of the model. k determines the number of k nearest neighbours we want to choose for the knn nearest neighbours initialisation method (Mixture methods). ks is similar to k but instead of looking at all the points we look at the points that belongs to a particular species separately. nstart represents the number random start to set for the kmeans initialization method. cov.list is the list of envrionmental covariates images. cov.bias is the environmental covariate set as the bias (its position in the covarite matrix). kval is the value used to correct for the bias in the predictions. we will set those to NULL for this simple example of the simulation. kAreaInt is set to NULL because it is an ongoing extension that needs more research. The different delta parameters are necessary for the LoopT method where delt_max is the maximum threshold membership probability to add points to the known point patterns. It is the starting threshold value. Delta_min it the lowest threshold value we can have to add points and delta_step is the value by which we decrease the threshold at each iteration. num_add represents the number of points to add at each iteration for the LoopE method. We also define the window and the quadrature points.

```
# Set up parameters
hidepct=c(0.8)
n.sims=50
sp_sim.list
```

```
## [[1]]
## Planar point pattern: 138 points
## window: rectangle = [-0.5, 100.5] x [-0.5, 100.5] units
##
## [[2]]
## Planar point pattern: 166 points
## window: rectangle = [-0.5, 100.5] x [-0.5, 100.5] units
##
## [[3]]
## Planar point pattern: 151 points
## window: rectangle = [-0.5, 100.5] x [-0.5, 100.5] units
```

```
n.sp=3
k = 1
ks=1:3
nstart=30
cov.list.=cov.list
cov.bias=NULL
kVal=NULL
kAreaInt=NULL
delta_max=0.5
delta_min=0.1
delta_step =0.1
num.add = 1

win   = owin(xrange = c(-0.5, 100.5), yrange = c(-0.5, 100.5))
quads.win = ppp(X, Y, window = win)
```

One simulation example

We start by hideing some information using the percentge of hidden observation we choose. In our simulation we choose different values and run the simulation for all of them. Here only the case of 80% of hidden observation is presented. We define some new object that corresponds to the known points, the points to test that have been hidden.

```
pct_hidden = hidepct

# hide some observations and prepare data objects
sp_hide.list = sp_sub.list = train.list = sp_test.list = list()
coordtestx.list = coordtesty.list = markshide.list = markstest.list = list()
coordsubx.list = coordsuby.list = marksub.list = list()

for (l in 1:n.sp) {
  sp_hide.list[[l]] = sample(1:sp_sim.list[[l]]$n, floor(pct_hidden*sp_sim.list[[l]]$n))
  sp_sub.list[[l]]  = sp_sim.list[[l]][-sp_hide.list[[l]]]
  train.list[[l]]   = ppp(x = sp_sub.list[[l]]$x, y = sp_sub.list[[l]]$y, window = win)
  sp_test.list[[l]] = sp_sim.list[[l]][sp_hide.list[[l]]]
```

```r
  coordtestx.list[[l]] = sp_test.list[[l]]$x
  coordtesty.list[[l]] = sp_test.list[[l]]$y
  markshide.list[[l]] = rep(paste("Hidden", l, sep = ""), sp_test.list[[l]]$n)
  markstest.list[[l]] = rep(paste("Sp", l, sep = ""), sp_test.list[[l]]$n)

  coordsubx.list[[l]] = sp_sub.list[[l]]$x
  coordsuby.list[[l]] = sp_sub.list[[l]]$y
  marksub.list[[l]] = rep(paste("Sp", l, sep = ""), sp_sub.list[[l]]$n)

  l=l+1
}

sp.true = sp_hide.list

all_test = ppp(x = c(unlist(coordtestx.list)),
               y = c(unlist(coordtesty.list)), window = win,
               marks = c(unlist(markshide.list)))

all_test2 = ppp(x = c(unlist(coordtestx.list)),
                y = c(unlist(coordtesty.list)), window = win,
                marks = c(rep("Unknown", all_test$n)))

test_labels = as.vector(unlist(markstest.list))

all_true = ppp(x = c(unlist(coordsubx.list)),
               y = c(unlist(coordsuby.list)), window = win,
               marks = c(unlist(marksub.list)))


datappp = superimpose.ppp(all_true, all_test2)
```

If we want to correct for bias we set up the correction following these lines:

```r
if(is.null(cov.bias)){
      cov.list. = cov.list.
    }else{#--- Set observer bias variables to kVal
      pred.list = cov.list.
      set.Val = cov.bias #Variables to set to a certain value
      for (v in set.Val){
        pred.list[[v]]$v = kVal*pred.list[[v]]$v
      }
    }
```

We first use the mixture model initialisation tests to reclassify our data. We can choose the type of classification method. The argument classif = “soft” allow to choose a soft classification, a hard classification is chosen using classif = “hard”. We run both to compare results.

```r
simknn = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                      all_test=all_test, initweights = "knn", sp_int_im = sp1_int_im,
                      k=k, ks=ks, nstart=nstart, ppmform = ppmform, cov.list. = cov.list,
                      cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                      verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                      classif = "soft")
```

```r
simkmeans = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                         all_test=all_test, initweights = "kmeans", sp_int_im = sp1_int_im,
                         k=k, ks=ks, nstart=nstart, ppmform = ppmform, cov.list. = cov.list,
                         cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                         verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                         classif = "soft")

simrandom = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                         all_test=all_test, initweights = "random", sp_int_im = sp1_int_im,
                         k=k, ks=ks, nstart=nstart, ppmform = ppmform, cov.list. = cov.list,
                         cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                         verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                         classif = "soft")

simCF = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                      all_test=all_test, initweights = "CoinF", sp_int_im = sp1_int_im,
                      k=NULL, ks=NULL, nstart=NULL, ppmform = ppmform, cov.list. = cov.list,
                      cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                      verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                      classif = "soft")

simkps = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                      all_test=all_test, initweights = "kps", sp_int_im = sp1_int_im,
                      k=NULL, ks=ks, nstart=nstart, ppmform = ppmform, cov.list. = cov.list,
                      cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                      verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                      classif = "soft")

simknn2 = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                      all_test=all_test, initweights = "knn", sp_int_im = sp1_int_im,
                      k=k, ks=ks, nstart=nstart, ppmform = ppmform, cov.list. = cov.list,
                      cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                      verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                      classif = "hard")

simkmeans2 = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                         all_test=all_test, initweights = "kmeans", sp_int_im = sp1_int_im,
                         k=k, ks=ks, nstart=nstart, ppmform = ppmform, cov.list. = cov.list,
                         cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                         verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                         classif = "hard")

simrandom2 = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                         all_test=all_test, initweights = "random", sp_int_im = sp1_int_im,
                         k=k, ks=ks, nstart=nstart, ppmform = ppmform, cov.list. = cov.list,
                         cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                         verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                         classif = "hard")

simCF2 = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                      all_test=all_test, initweights = "CoinF", sp_int_im = sp1_int_im,
                      k=NULL, ks=NULL, nstart=NULL, ppmform = ppmform, cov.list. = cov.list,
                      cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
```

```
                        verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                        classif = "hard")

simkps2 = ppmMixEngine(datappp = datappp, quads. = quads.win, all_true=all_true, n.sp=n.sp,
                        all_test=all_test, initweights = "kps", sp_int_im = sp1_int_im,
                        k=NULL, ks=ks, nstart=nstart, ppmform = ppmform, cov.list. = cov.list,
                        cov.bias = cov.bias, kVal = kVal, kAreaInt = kAreaInt,
                        verbose = TRUE, tol = 0.000001, maxit = 50, plots = FALSE,
                        classif = "hard")
```

We then calculate and store the performances measures: Weights, coefficients, predictions, accuracy, meanRSS, IMSE and sumcor for each method.

```
# for performance measures

knn.perf = Perffunc(simknn, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"),  LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "soft")


kmeans.perf = Perffunc(simkmeans, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"), LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "soft")


random.perf = Perffunc(simrandom, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"), LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "soft")


CF.perf = Perffunc(simCF, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"), LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "soft")


kps.perf = Perffunc(simkps, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"), LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "soft")

knn2.perf = Perffunc(simknn2, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"),  LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "hard")


kmeans2.perf = Perffunc(simkmeans2, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"), LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "hard")
```

```r
random2.perf = Perffunc(simrandom2, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"), LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "hard")


CF2.perf = Perffunc(simCF2, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"), LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "hard")


kps2.perf = Perffunc(simkps2, sp_int.list, datappp, fun = "log", rescale = TRUE,
                    method=c("pearson", "kendall", "spearman"), LoopM=FALSE,
                    mu.min = 1.e-5, all_true, test_labels, n.sp, pf = c(NULL),
                    classif = "hard")

# for intensity plots
knnpred = as.matrix(unlist(Predlist(simknn$pred.loc, n.sp)))
kmeanspred =as.matrix(unlist(Predlist(simkmeans$pred.loc, n.sp)))
randpred = as.matrix(unlist(Predlist(simrandom$pred.loc, n.sp)))
CFpred = as.matrix(unlist(Predlist(simCF$pred.loc, n.sp)))
kpspred = as.matrix(unlist(Predlist(simkps$pred.loc, n.sp)))
```

We can do a similar work with the Loop methods:

```r
simLoopA = ppmLoopEngine(datappp, all_test, n.sp, addpt = "LoopA", quads.= quads.win,
                          ppmform=ppmform, delta_max=NULL, delta_min=NULL,
                          delta_step=NULL, win. = win, num.add = NULL,
                          cov.list.=cov.list, cov.bias=NULL, kVal =NULL,
                          kAreaInt=NULL, maxit = 50, tol=0.000001,
                          verbose = TRUE, plots = FALSE)

simLoopT = ppmLoopEngine(datappp, all_test, n.sp, addpt = "LoopT", quads.= quads.win,
                          ppmform= ppmform, delta_max=delta_max, delta_min=delta_min,
                          delta_step=delta_step, win. = win, num.add = NULL,
                          cov.list.=cov.list, cov.bias=NULL, kVal =NULL,
                          kAreaInt=NULL, maxit = 50, tol=0.000001,
                          verbose = TRUE, plots = FALSE)

simLoopE = ppmLoopEngine(datappp, all_test, n.sp, addpt = "LoopE", quads.= quads.win,
                          ppmform= ppmform, delta_max=NULL, delta_min=NULL,
                          delta_step=NULL,  win. = win, num.add = num.add,
                          cov.list.=cov.list, cov.bias=NULL, kVal =NULL,
                          kAreaInt=NULL, maxit = 50, tol=0.000001,
                          verbose = TRUE, plots = FALSE)
```

We again calculate the performance measures.

```r
LA.perf = Perffunc(simLoopA, sp_int.list, datappp, fun = "log", rescale = TRUE,
                   all_true, LoopM=TRUE, classif=NULL, method="pearson",
                   mu.min = 1.e-5, test_labels, n.sp, pf = NULL)
```

```
LT.perf = Perffunc(simLoopT, sp_int.list, datappp, fun = "log", rescale = TRUE,
                   all_true, LoopM=TRUE, classif=NULL, method="pearson",
                   mu.min = 1.e-5, test_labels, n.sp, pf = NULL)

LE.perf = Perffunc(simLoopE, sp_int.list, datappp, fun = "log", rescale = TRUE,
                   all_true, LoopM=TRUE, classif=NULL, method="pearson",
                   mu.min = 1.e-5, test_labels, n.sp, pf = NULL)


# for intensity plots
LoopApred = matrix(unlist(simLoopA$pred.loc),
                   nrow=length(simLoopA$pred.loc[[1]]), byrow=F)

LoopTpred = matrix(unlist(simLoopT$pred.loc),
                   nrow=length(simLoopT$pred.loc[[1]]), byrow=F)

LoopEpred = matrix(unlist(simLoopE$pred.loc),
                   nrow=length(simLoopE$pred.loc[[1]]), byrow=F)
```

We can then compare the diferent method performances values. If we have many simulation, we compare boxplots of the different values.

```
# Comparison between hard and soft classification
ACCvec2 = c(knn.perf$accmat, kmeans.perf$accmat, random.perf$accmat, CF.perf$accmat,
           kps.perf$accmat, knn2.perf$accmat, kmeans2.perf$accmat, random2.perf$accmat,
           CF2.perf$accmat, kps2.perf$accmat)
meanRSSvec2 = c(knn.perf$meanRSS, kmeans.perf$meanRSS, random.perf$meanRSS, CF.perf$meanRSS,
           kps.perf$meanRSS, knn2.perf$meanRSS, kmeans2.perf$meanRSS, random2.perf$meanRSS,
           CF2.perf$meanRSS, kps2.perf$meanRSS)
IMSEvec2 = c(knn.perf$IMSE, kmeans.perf$IMSE, random.perf$IMSE, CF.perf$IMSE,
           kps.perf$IMSE, knn2.perf$IMSE, kmeans2.perf$IMSE, random2.perf$IMSE,
           CF2.perf$IMSE, kps2.perf$IMSE)
sumcorvec2 = c(knn.perf$sumcor1, kmeans.perf$sumcor1, random.perf$sumco1, CF.perf$sumcor1,
           kps.perf$sumcor1, knn2.perf$sumcor1, kmeans2.perf$sumcor1, random2.perf$sumco1,
           CF2.perf$sumcor1, kps2.perf$sumcor1)

Perfmixt = cbind(ACCvec2, meanRSSvec2, IMSEvec2, sumcorvec2)
rownames(Perfmixt) = c("knn", "kmeans", "rand", "CoinF", "kps",
                       "knn-hard", "kmeans-hard", "rand-hard", "CoinF-hard", "kps-hard")
Perfmixt
```

```
##                ACCvec2 meanRSSvec2  IMSEvec2 sumcorvec2
## knn          0.4337017   0.4266574  9261.848  0.8919063
## kmeans       0.4337017   0.4266572  8928.320  0.8919063
## rand         0.4337017   0.4266572  8928.320  0.8919063
## CoinF        0.4337017   0.4266570  9191.452  0.8919063
## kps          0.4337017   0.4266572  8928.320  0.7654013
## knn-hard     0.3038674   0.6800550 35803.257  0.8919063
## kmeans-hard  0.3314917   0.6568788  9182.328  0.7653695
## rand-hard    0.3038674   0.6800550 35803.257  0.7653695
## CoinF-hard   0.3646409   0.6214036 35810.015  0.8919063
## kps-hard     0.3646409   0.6214036 35810.015  0.8919063
```

```
# Comparison between Mixture and Loop classsifcation
ACCvec = c(knn.perf$accmat, kmeans.perf$accmat, random.perf$accmat, CF.perf$accmat,
```

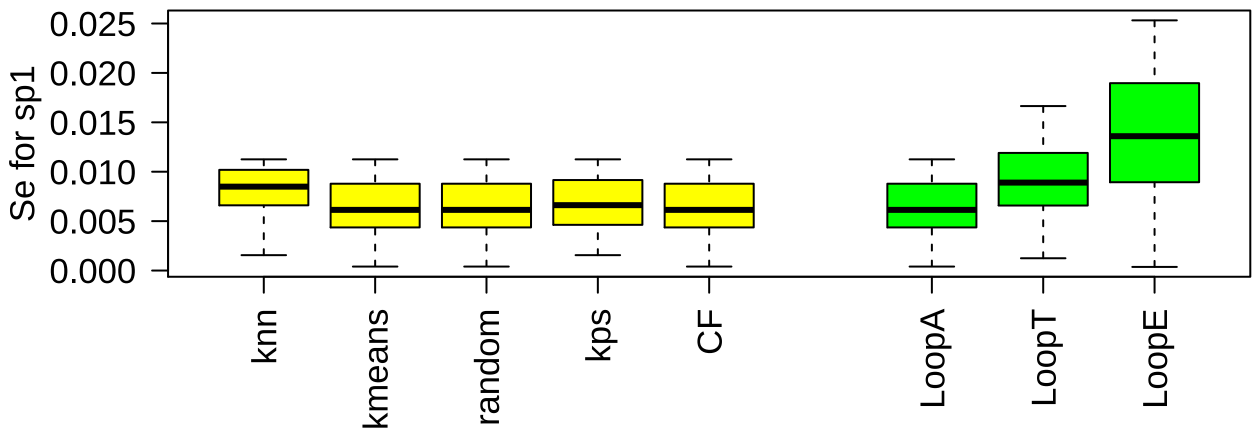
```
           kps.perf$accmat, LA.perf$accmat, LT.perf$accmat, LE.perf$accmat)
meanRSSvec = c(knn.perf$meanRSS, kmeans.perf$meanRSS, random.perf$meanRSS, CF.perf$meanRSS,
           kps.perf$meanRSS, LA.perf$meanRSS, LT.perf$meanRSS, LE.perf$meanRSS)
IMSEvec = c(knn.perf$IMSE, kmeans.perf$IMSE, random.perf$IMSE, CF.perf$IMSE,
           kps.perf$IMSE, LA.perf$IMSE, LT.perf$IMSE, LE.perf$IMSE)
sumcorvec = c(knn.perf$sumcor1, kmeans.perf$sumcor1, random.perf$sumco1, CF.perf$sumcor1,
           kps.perf$sumcor1, LA.perf$sumcor1, LT.perf$sumcor1, LE.perf$sumcor1)

Perfmat = cbind(ACCvec, meanRSSvec, IMSEvec, sumcorvec)
rownames(Perfmat) = c("knn", "kmeans", "rand", "CoinF", "kps", "LoopA", "LoopT", "LoopE")
Perfmat
```

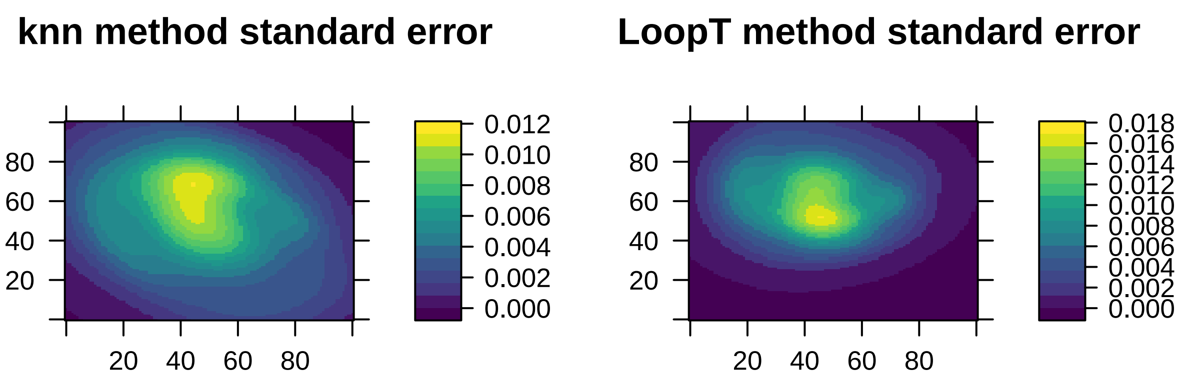
```
##           ACCvec meanRSSvec   IMSEvec sumcorvec
## knn    0.4337017  0.4266574  9261.848 0.8919063
## kmeans 0.4337017  0.4266572  8928.320 0.8919063
## rand   0.4337017  0.4266572  8928.320 0.8919063
## CoinF  0.4337017  0.4266570  9191.452 0.8919063
## kps    0.4337017  0.4266572  8928.320 0.8919063
## LoopA  0.5276243  0.4283530  8928.320 0.9352549
## LoopT  0.4889503  0.3332432 18011.221 0.8202568
## LoopE  0.4972376  0.4019125 39401.461 0.8919063
```

We can also calculate uncertainties. We choose to display and compare the prediction standard errors for species 1 only and compare standard error map from 3 methods: knn, LoopT and indiv method.

```
# standard error values for species 1
se.knn1 <- predict(simknn$fit.final[[1]], locations=simknn$sp_aug.list[[1]], se=TRUE)
se.kmeans1 <- predict(simkmeans$fit.final[[1]], locations=simkmeans$sp_aug.list[[1]], se=TRUE)
se.rand1 <- predict(simrandom$fit.final[[1]], locations=simrandom$sp_aug.list[[1]], se=TRUE)
se.CF1 <- predict(simCF$fit.final[[1]], locations=simCF$sp_aug.list[[1]], se=TRUE)
se.kps1 <- predict(simkps$fit.final[[1]], locations=simkps$sp_aug.list[[1]], se=TRUE)
se.LA1 <- predict(simLoopA$ppm_list[[1]], locations=simLoopA$sp_aug_ppp.list[[1]], se=TRUE)
se.LT1 <- predict(simLoopT$ppm_list[[1]], locations=simLoopT$sp_aug_ppp.list[[1]], se=TRUE)
se.LE1 <- predict(simLoopE$ppm_list[[1]], locations=simLoopE$sp_aug_ppp.list[[1]], se=TRUE)

SEMmix.sp1 = cbind(se.knn1$se, se.kmeans1$se, se.rand1$se, se.CF1$se, se.kps1$se)

# boxplots
yrangeSE = range(c(min(SEMmix.sp1, se.LA1$se, se.LT1$se, se.LE1$se),
                   max(SEMmix.sp1, se.LA1$se, se.LT1$se, se.LE1$se)))

layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
par(mar=c(5,5,0,0)+0.1,mgp=c(4,1,0))

boxplot(se.knn1$se, se.kmeans1$se, se.rand1$se, se.CF1$se, se.kps1$se,
        se.LA1$se, se.LT1$se, se.LE1$se,
        col = c("yellow","yellow","yellow","yellow","yellow",
                "green","green","green"),
        names = c("knn", "kmeans", "random", "kps", "CF", "LoopA",
                  "LoopT", "LoopE"),
        at = c(1,2,3,4,5, 7,8,9),
        ylab = "Se for sp1", ylim=yrangeSE, las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)

# standard error plots
seplot.knn <- predict(simknn$fit.final[[1]], se=TRUE)
```

```r
seplot.LT <- predict(simLoopT$ppm_list[[1]], se=TRUE)  # for sp1

Xplot = as.data.frame(simknn$fitaft.pred[[1]])$x
Yplot = as.data.frame(simknn$fitaft.pred[[1]])$y

print(levelplot(as.vector(seplot.knn$se$v) ~ Xplot + Yplot, col.regions=viridis(20),
                main="knn method standard error", ylab=NULL, xlab=NULL),
      split=c(1, 2, 2, 2), newpage=FALSE)
print(levelplot(as.vector(seplot.LT$se$v) ~ Xplot + Yplot, col.regions=viridis(20),
                main="LoopT method standard error", ylab=NULL, xlab=NULL),
      split=c(2, 2, 2, 2), newpage=FALSE)
```





Multiple simulation example

We can also play with multiple simulation (or several percentage of hidden observation too) and compare the results.

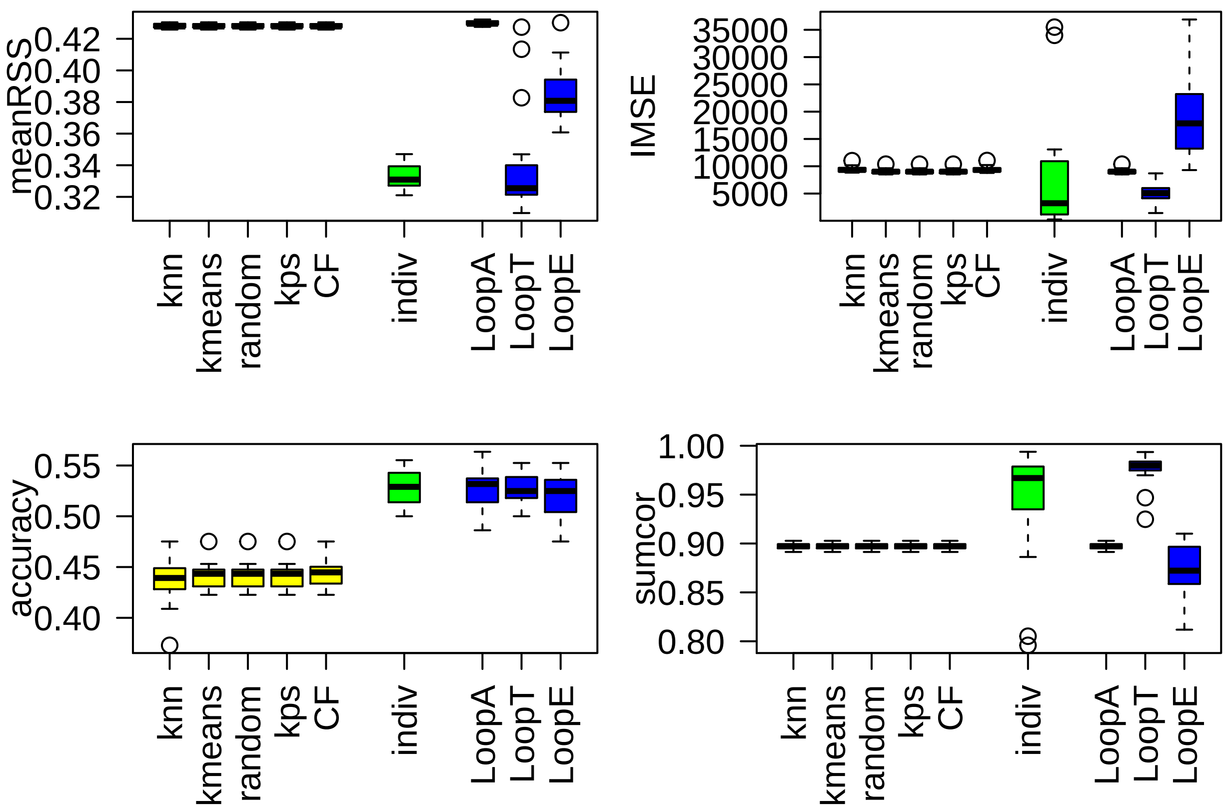
```r
QuickTest = Testsims(hidepct=c(0.8), n.sims=20, sp_sim.list, sp1_int_im, n.sp=3, k = 1, ks=1,
                    nstart=30, cov.list=cov.list, cov.bias=NULL, kVal=NULL, kAreaInt=NULL,
                                    delta_max=0.5, delta_min=0.1, delta_step =0.1, num.add = 5)
```

We can created some boxplots to compare the results of the different methods:

```r
par(mfrow=c(2,2))
par(mar=c(5,4,2,1))

####################################################
```

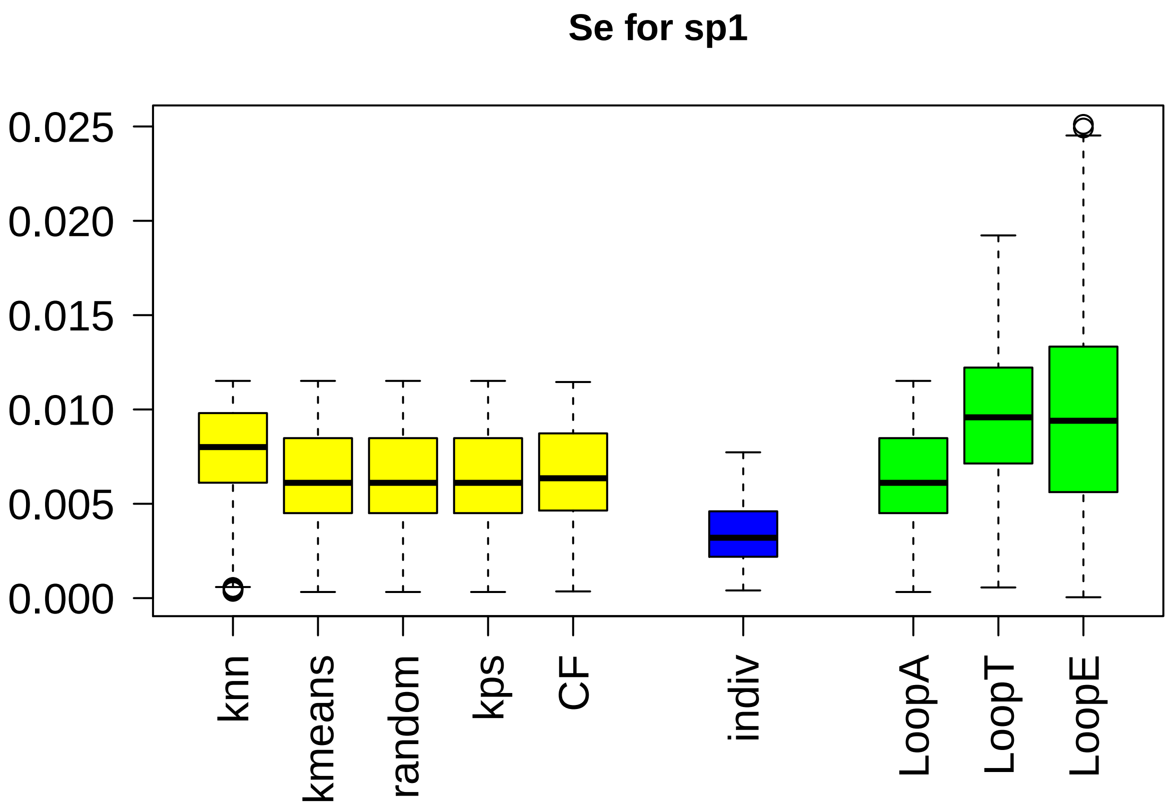
```r
## meanRSS #-----------------------------------------------------------------------------
yrangemeanRSS = range(c(min(QuickTest$meanRSSknn, QuickTest$meanRSSkmeans,
                            QuickTest$meanRSSrand, QuickTest$meanRSSkps,
                            QuickTest$meanRSSCF, QuickTest$meanRSSindiv, QuickTest$meanRSSLoopA,
                            QuickTest$meanRSSLoopT, QuickTest$meanRSSLoopE),
                         max(QuickTest$meanRSSknn, QuickTest$meanRSSkmeans,
                             QuickTest$meanRSSrand, QuickTest$meanRSSkps,
                             QuickTest$meanRSSCF, QuickTest$meanRSSindiv, QuickTest$meanRSSLoopA,
                             QuickTest$meanRSSLoopT, QuickTest$meanRSSLoopE)))

meanRSSmat = cbind(QuickTest$meanRSSknn, QuickTest$meanRSSkmeans,
                   QuickTest$meanRSSrand, QuickTest$meanRSSkps,
                   QuickTest$meanRSSCF, QuickTest$meanRSSindiv, QuickTest$meanRSSLoopA,
                   QuickTest$meanRSSLoopT, QuickTest$meanRSSLoopE)


boxplot(meanRSSmat, col = c("yellow","yellow","yellow","yellow","yellow",
                               "green", "blue", "blue", "blue"),
        names = c("knn", "kmeans", "random", "kps", "CF","indiv",
                  "LoopA","LoopT","LoopE"),
        at = c(1,2,3,4,5, 7, 9,10,11),
        ylab = "meanRSS", ylim=yrangemeanRSS, las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)

## IMSE #--------------------------------------------------------------------------------
par(mar=c(5,6,2,1),mgp=c(5,1,0))
yrangeIMSE = range(c(min(QuickTest$IMSEknn, QuickTest$IMSEkmeans,
                         QuickTest$IMSErand, QuickTest$IMSEkps,
                         QuickTest$IMSECF, QuickTest$IMSELoopA,
                         QuickTest$IMSELoopT, QuickTest$IMSELoopE, na.rm=T),
                      max(QuickTest$IMSEknn, QuickTest$IMSEkmeans,
                          QuickTest$IMSErand, QuickTest$IMSEkps,
                          QuickTest$IMSECF, QuickTest$IMSELoopA,
                          QuickTest$IMSELoopT, QuickTest$IMSELoopE, na.rm=T)))

IMSEmat = cbind(QuickTest$IMSEknn, QuickTest$IMSEkmeans,
                   QuickTest$IMSErand, QuickTest$IMSEkps, QuickTest$IMSECF,
                   QuickTest$IMSEindiv, QuickTest$IMSELoopA,
                   QuickTest$IMSELoopT, QuickTest$IMSELoopE)


boxplot(IMSEmat, col = c("yellow","yellow","yellow","yellow","yellow",
                                "green","blue", "blue", "blue"),
        names = c("knn", "kmeans", "random", "kps", "CF", "indiv",
                  "LoopA","LoopT","LoopE"),
        at = c(1,2,3,4,5, 7, 9,10,11),
        ylab = "IMSE", ylim=yrangeIMSE, las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)

par(mar=c(5,4,2,1),mgp=c(3,1,0))

## acc #---------------------------------------------------------------------------------
yrangeaccmat = range(c(min(QuickTest$accmatknn, QuickTest$accmatkmeans,
                               QuickTest$accmatrand, QuickTest$accmatkps, QuickTest$accmatCF,
                               QuickTest$accmatindiv, QuickTest$accmatLoopA,
```

```r
                            QuickTest$accmatLoopT, QuickTest$accmatLoopE, na.rm=T),
                        max(QuickTest$accmatknn, QuickTest$accmatkmeans,
                            QuickTest$accmatrand, QuickTest$accmatkps, QuickTest$accmatCF,
                            QuickTest$accmatindiv, QuickTest$accmatLoopA,
                            QuickTest$accmatLoopT, QuickTest$accmatLoopE, na.rm=T)))

accmatmat = cbind(QuickTest$accmatknn, QuickTest$accmatkmeans,
                  QuickTest$accmatrand, QuickTest$accmatkps, QuickTest$accmatCF,
                  QuickTest$accmatindiv, QuickTest$accmatLoopA,
                  QuickTest$accmatLoopT, QuickTest$accmatLoopE)


boxplot(accmatmat, col = c("yellow","yellow","yellow","yellow","yellow",
                                "green","blue", "blue", "blue"),
        names = c("knn", "kmeans", "random", "kps", "CF", "indiv",
                  "LoopA","LoopT","LoopE"),
        at = c(1,2,3,4,5, 7, 9,10,11),
        ylab = "accuracy", ylim=yrangeaccmat, las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)

## sumcor #-----------------------------------------------------------------------------

yrangesumcor = range(c(min(QuickTest$sumcorknn1, QuickTest$sumcorkmeans1,
                            QuickTest$sumcorrand1, QuickTest$sumcorkps1, QuickTest$sumcorCF1,
                            QuickTest$sumcorindiv1, QuickTest$sumcorLoopA1,
                            QuickTest$sumcorLoopT1, QuickTest$sumcorLoopE1, na.rm=T),
                        max(QuickTest$sumcorknn1, QuickTest$sumcorkmeans1,
                            QuickTest$sumcorrand1, QuickTest$sumcorkps1, QuickTest$sumcorCF1,
                            QuickTest$sumcorindiv1, QuickTest$sumcorLoopA1,
                            QuickTest$sumcorLoopT1, QuickTest$sumcorLoopE1, na.rm=T)))

sumcormat = cbind(QuickTest$sumcorknn1, QuickTest$sumcorkmeans1,
                  QuickTest$sumcorrand1, QuickTest$sumcorkps1, QuickTest$sumcorCF1,
                  QuickTest$sumcorindiv1, QuickTest$sumcorLoopA1,
                  QuickTest$sumcorLoopT1, QuickTest$sumcorLoopE1)


boxplot(sumcormat, col = c("yellow","yellow","yellow","yellow","yellow",
                                "green","blue", "blue", "blue"),
        names = c("knn", "kmeans", "random", "kps", "CF", "indiv",
                  "LoopA","LoopT","LoopE"),
        at = c(1,2,3,4,5, 7, 9,10,11),
        ylab = "sumcor", ylim=yrangesumcor, las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)
```



Similarly we can also calculate some standard errors from the multiple simulation test:

```
# standard error values for species 1
# boxplots
yrangeSE = range(c(min(unlist(QuickTest$se.knn1), unlist(QuickTest$se.kmeans1),
                       unlist(QuickTest$se.rand1), unlist(QuickTest$se.kps1),
                       unlist(QuickTest$se.CF1), unlist(QuickTest$se.LA1),
                       unlist(QuickTest$se.LT1), unlist(QuickTest$se.LE1),
                       unlist(QuickTest$se.indiv1)),
                   max(unlist(QuickTest$se.knn1), unlist(QuickTest$se.kmeans1),
                       unlist(QuickTest$se.rand1), unlist(QuickTest$se.kps1),
                       unlist(QuickTest$se.CF1), unlist(QuickTest$se.LA1),
                       unlist(QuickTest$se.LT1), unlist(QuickTest$se.LE1),
                       unlist(QuickTest$se.indiv1))))

boxplot(unlist(QuickTest$se.knn1), unlist(QuickTest$se.kmeans1), unlist(QuickTest$se.rand1),
        unlist(QuickTest$se.kps1), unlist(QuickTest$se.CF1), unlist(QuickTest$se.indiv1),
        unlist(QuickTest$se.LA1), unlist(QuickTest$se.LT1), unlist(QuickTest$se.LE1),
        col = c("yellow","yellow","yellow","yellow","yellow", "blue","green","green","green"),
        names = c("knn", "kmeans", "random", "kps", "CF", "indiv", "LoopA","LoopT", "LoopE"),
        at = c(1,2,3,4,5, 7, 9,10,11),
        main = "Se for sp1", ylim=yrangeSE, las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)
```




```r
# average standard error plots
SEwknn1 = SEwLT1 = SEwindiv1 = list()
for (i in 1:length(QuickTest$sew.knn1)) {
  SEwknn1[[i]] = QuickTest$sew.knn1[[i]]$v
  SEwLT1[[i]] = QuickTest$sew.LT1[[i]]$v
  SEwindiv1[[i]] = QuickTest$sew.indiv1[[i]]$v
}
SEknn1sim = Reduce(`+`, SEwknn1)/length(SEwknn1)
SELT1sim = Reduce(`+`, SEwLT1)/length(SEwLT1)
SEindiv1sim = Reduce(`+`, SEwindiv1)/length(SEwindiv1)

Lseknn = levelplot(SEknn1sim ~ Xplot + Yplot, col.regions=viridis(20))
LseLT = levelplot(SELT1sim ~ Xplot + Yplot, col.regions=viridis(20))
Lseindiv = levelplot(SEindiv1sim ~ Xplot + Yplot, col.regions=viridis(20))
comb_levObj2 <- c(Lseknn, LseLT, Lseindiv,
                  layout = c(3, 1), merge.legends = T)
update(comb_levObj2,  xlab = c("knn", "LoopT", "indiv"),
       main="Standard error predictions for sp1")
```

Standard error predictions for sp1