Classification of Unlabeled observations

Emy Guilbault

28 March 2020

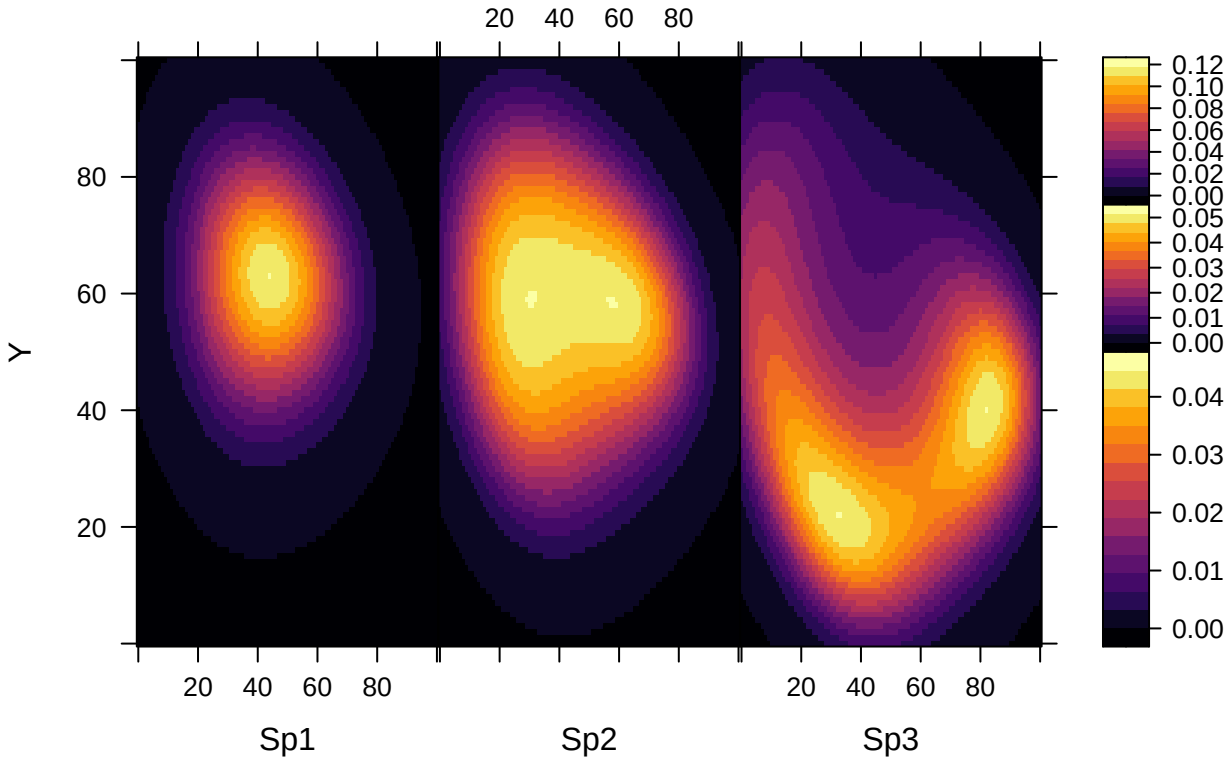
This document runs through the different steps to use the functions ppmMixEngine and ppmLoopEngine for data classification. These functions and others are contained in functionTestsim160420-SH.R. First, we load the various functions and packages we will need.

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

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)
```

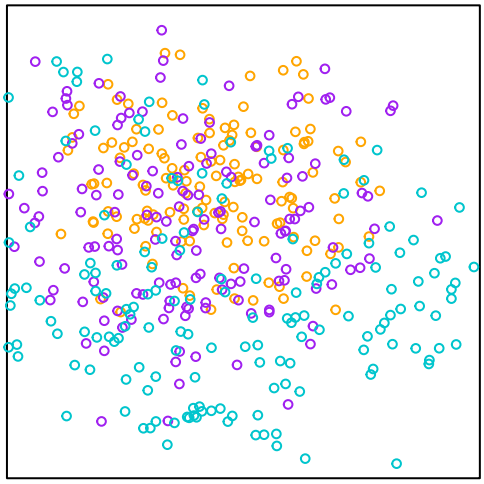
Data and environmental covariates

We load simulated data points for three species and environmental covariates. We display the species true intensity as well as the three point patterns.

```
load("PrepData.RDATA")

# Species 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")
```




```
# point patterns
plot(sp1_sim, cex = 0.6, col="white")
plot(sp1_sim, add = TRUE, col = "orange", cex = 0.6)
plot(sp2_sim, add = TRUE, col = "purple", cex = 0.6)
plot(sp3_sim, add = TRUE, col = "turquoise3", cex = 0.6)
```



One simulation example

Mixture methods

We use a simulated dataset where we hide some label information. In the article, we choose three values and run the algorithm for all of them. Here we only present the case of 60% of hidden observations. The points with hidden observations compose the Unknown.ppp pattern. The Known.ppp pattern is a marked point pattern with each marks representing a known species. quadsenv is a matrix with information on the quadrature points: coordinates x and y and environmental covariates information at those points. ppmform contains the spatial trend we want the model to fit our data. The initweights arguments allow to decide the method to calculate the initial weights: knn (we use the distance to the k nearest points), kmeans (we use the distance to the k centroids of the known species), random (we randomly attribute initial weight to the unknown points.), kps (similar to knn but the k nearest distances calculated within each species), coinF (we randomly attribute a label to the unknown points).

We set up the parameters to use in the initialization of the method: k determines the number of k nearest neighbors we want to choose for the knn nearest neighbors initialization method. We can choose the type of classification method. The argument classif = "soft" allows us to choose a soft classification. A hard classification is chosen using classif = "hard". We run both to compare the results.

```
# arguments
n.sp=3
k = 1

# models
simknn = ppmMixEngine(Known.ppp, Unknown.ppp, quadsenv = Quadmat, n.sp=n.sp,
```

```
                        initweights = "knn", k=k, ppmform = ppmform,
                        classif = "soft")

simCF = ppmMixEngine(Known.ppp, Unknown.ppp, quadsenv = Quadmat, n.sp=n.sp,
                       initweights = "CoinF", k=NULL, ppmform = ppmform,
                        classif = "soft")

simknn2 = ppmMixEngine(Known.ppp, Unknown.ppp, quadsenv = Quadmat, n.sp=n.sp,
                        initweights = "knn", k=k, ppmform = ppmform,
                        classif = "hard")

simCF2 = ppmMixEngine(Known.ppp, Unknown.ppp, quadsenv = Quadmat, n.sp=n.sp,
                        initweights = "CoinF", k=NULL, ppmform = ppmform,
                        classif = "hard")
```

We then calculate and store the performance measures: Weights, coefficients, predictions, accuracy, meanRSS, IMSE, and sumcor for each method using the perffunc function. We need to specify the fitted model object, the list of the true species intensities, the known marked point pattern defined earlier, the label hidden to be reclassified and the number of species. The pf argument helps to choose which performances we want to compute between accuracy, meanRSS, sumIMSE and sumcor. The default value NULL computes all performances. For IMSE and sumcor, we can choose to compute a log or a square root of the intensity with the argument fun. By default, fun = "Else", which does not modify the predicted intensity. We can also decide the method to do in the calculation of the correlation ("pearson", "kendall", "spearman"). For mixture methods, the LoopM argument is set up to FALSE by opposition to the loop methods for which this argument will be TRUE.

```
# for performance measures
knn.perf = Perffunc(simknn, sp_int.list, n.sp=3, Known.ppp.=Known.ppp,
                    Unknown_labels.=Unknown_labels, pf = c(NULL),
                    method=c("pearson"), LoopM=FALSE)

CF.perf = Perffunc(simCF, sp_int.list, n.sp=3, Known.ppp.=Known.ppp,
                    Unknown_labels.=Unknown_labels, pf = c(NULL),
                    method=c("pearson"), LoopM=FALSE)

knn2.perf = Perffunc(simknn2, sp_int.list, n.sp=3, Known.ppp.=Known.ppp,
                    Unknown_labels.=Unknown_labels, pf = c(NULL),
                    method=c("pearson"), LoopM=FALSE)

CF2.perf = Perffunc(simCF2, sp_int.list, n.sp=3, Known.ppp.=Known.ppp,
                    Unknown_labels.=Unknown_labels, pf = c(NULL),
                    method=c("pearson"), LoopM=FALSE)

# for intensity plots
knnpred = as.matrix(unlist(Predlist(simknn$pred.loc, n.sp)))
CFpred = as.matrix(unlist(Predlist(simCF$pred.loc, n.sp)))
```

Loop methods

We can do a similar job with the Loop methods. First, we run the Loop models where the argument Known.ppp, Unknown.ppp, n.sp and quadsenv are the ones also used for the Mixture methods. addpt allows to choose the looping method of the algorithm: -"LoopA" for all points, -"LoopT" for all points with membership probabilities above delta_max and we decrease at each iteration by delta_step the membership probabilities till we reach delta_min. Each of those delta needs to be specified. -"LoopE" for adding a similar

number of points for each species at start and increasing by one point after the first iteration. The number of initial points to add is determined by num.add.

```
# arguments
delta_max=0.5
delta_min=0.1
delta_step =0.1
num.add = 1

# models
simLoopT = ppmLoopEngine(Known.ppp, Unknown.ppp, n.sp, addpt = "LoopT",
                         quadsenv = Quadmat, ppmform= ppmform,
                         delta_max=delta_max, delta_min=delta_min,
                                      delta_step=delta_step, num.add = NULL)

simLoopE = ppmLoopEngine(Known.ppp, Unknown.ppp, n.sp, addpt = "LoopE",
                         quadsenv = Quadmat, ppmform= ppmform,
                         delta_max=NULL, delta_min=NULL,
                                      delta_step=NULL,  num.add = num.add)
```

We again calculate the performance measures. This time, we choose the argument LoopM=TRUE.

```
LT.perf = Perffunc(simLoopT, sp_int.list, n.sp=3, Known.ppp.=Known.ppp,
                    Unknown_labels.=Unknown_labels, pf = c(NULL),
                    method=c("pearson"), LoopM=TRUE)

LE.perf = Perffunc(simLoopE, sp_int.list, n.sp=3, Known.ppp.=Known.ppp,
                    Unknown_labels.=Unknown_labels, pf = c(NULL),
                    method=c("pearson"), LoopM=TRUE)

# for intensity plots
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)
```

Methods comparison

We can then compare performance of the methods. If we have many simulations, we compare boxplots of the performance measures.

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

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

```
##              ACCvec2 meanRSSvec2  IMSEvec2 sumcorvec2
## knn        0.3210332   0.4091861 4.3750117   2.452780
## CoinF      0.3210332   0.4091806 4.3750117   2.452780
## knn-hard   0.3210332   0.4493174 1.2172676   2.825221
```

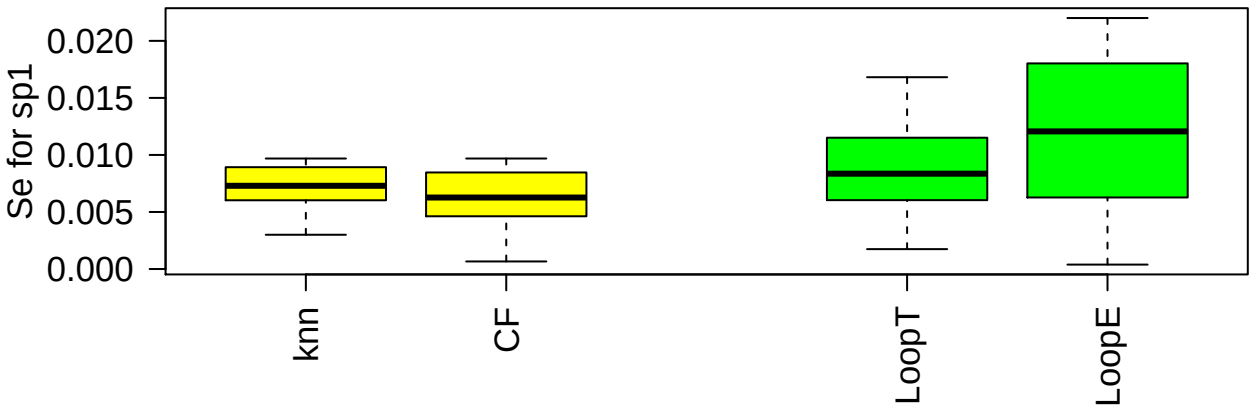
```
## CoinF-hard 0.3173432   0.3827175 0.8362754   2.918973
```

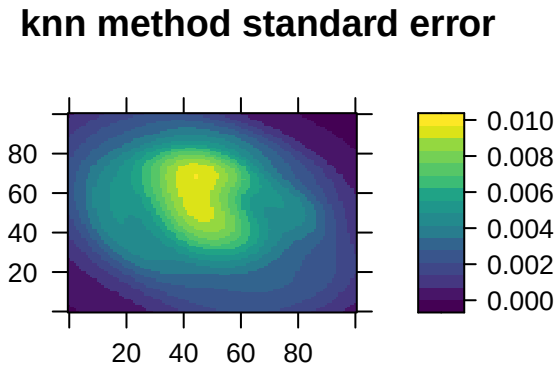
```
# Comparison between Mixture and Loop classifcation
ACCvec = c(knn.perf$accmat, CF.perf$accmat,LT.perf$accmat, LE.perf$accmat)
meanRSSvec = c(knn.perf$meanRSS, CF.perf$meanRSS, LT.perf$meanRSS, LE.perf$meanRSS)
IMSEvec = c(knn.perf$IMSE, CF.perf$IMSE, LT.perf$IMSE, LE.perf$IMSE)
sumcorvec = c(knn.perf$sumcor1, CF.perf$sumcor1, LT.perf$sumcor1, LE.perf$sumcor1)

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

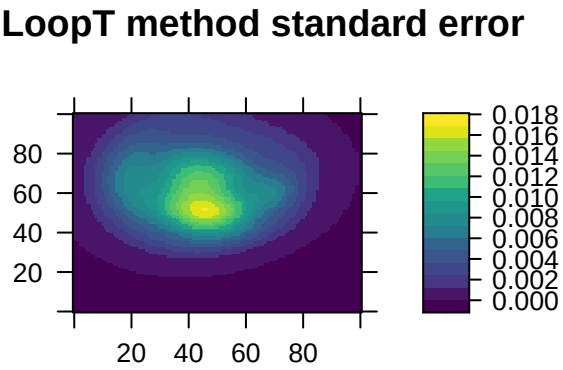
```
##          ACCvec meanRSSvec   IMSEvec sumcorvec
## knn   0.3210332  0.4091861 4.3750117  2.452780
## CoinF 0.3210332  0.4091806 4.3750117  2.452780
## LoopT 0.3505535  0.3122151 0.6602005  2.945083
## LoopE 0.3468635  0.3489501 6.3477740  2.566596
```

We can also calculate uncertainties. We choose to display and compare the prediction standard errors for species 1 only and compare the 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.CF1 <- predict(simCF$fit.final[[1]], locations=simCF$sp_aug.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$se, se.CF1$se)

# boxplots
yrangeSE = range(c(min(SEMmix.sp1, se.LT1$se, se.LE1$se),
                   max(SEMmix.sp1, 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$se, se.CF1$se, se.LT1$se, se.LE1$se,
        col = c("yellow","yellow",
                "green","green","green"),
        names = c("knn", "CF", "LoopT", "LoopE"),
        at = c(1,2, 4,5), 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$fit.final[[1]], se=TRUE)
seplot.LT <- predict(simLoopT$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 play with multiple simulations (and/or several percentages of hidden observation). To do so, we define the following parameters: the percentage of unknown points we want (we can have several values in a vector but the example use is 0.6), the number of simulations (here set up to 20). The other parameters have been defined before.

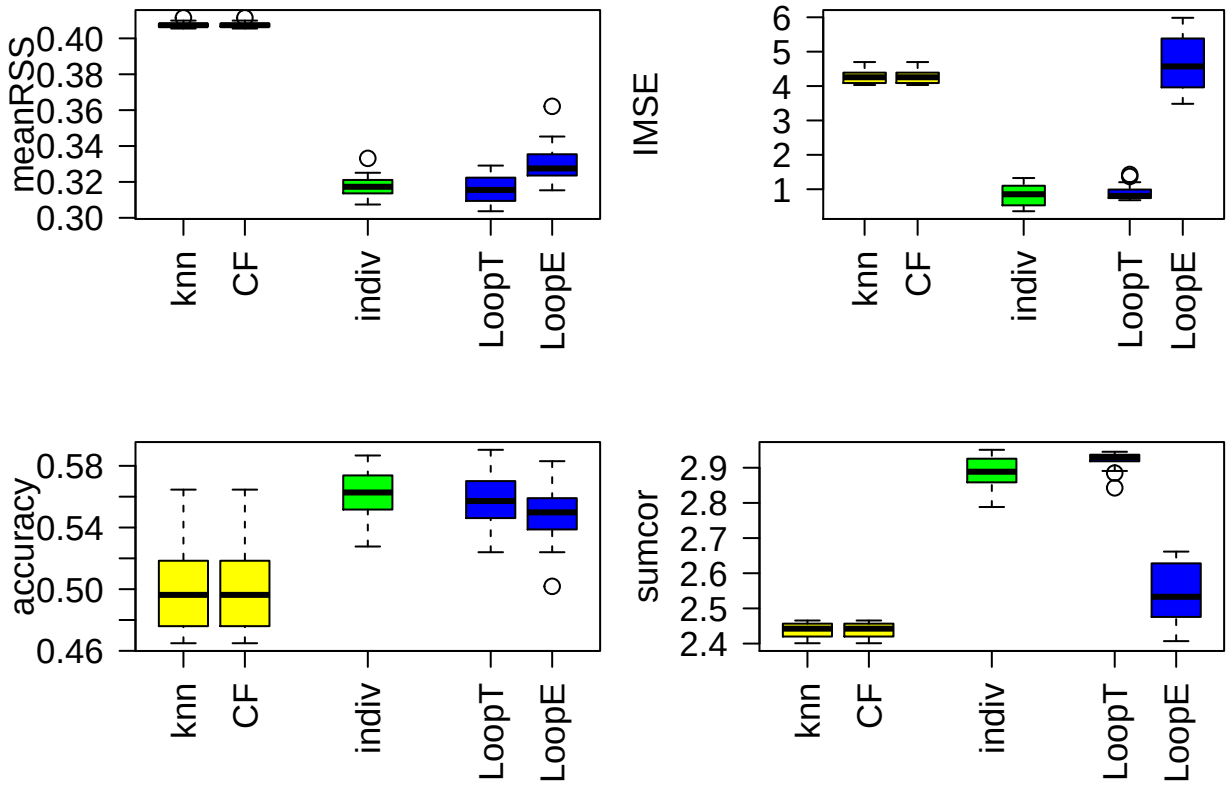
```
QuickTest = Testsims(hidepct=c(0.6), n.sims=20, sp_sim.list, n.sp=3, k=k,
                     ks=ks, nstart=nstart, quadsenv = Quadmat,
                     delta_max=delta_max, delta_min=delta_min,
                     delta_step=delta_step, num.add=num.add)
```

We can create boxplots to compare the performances measures of the different methods:

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

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

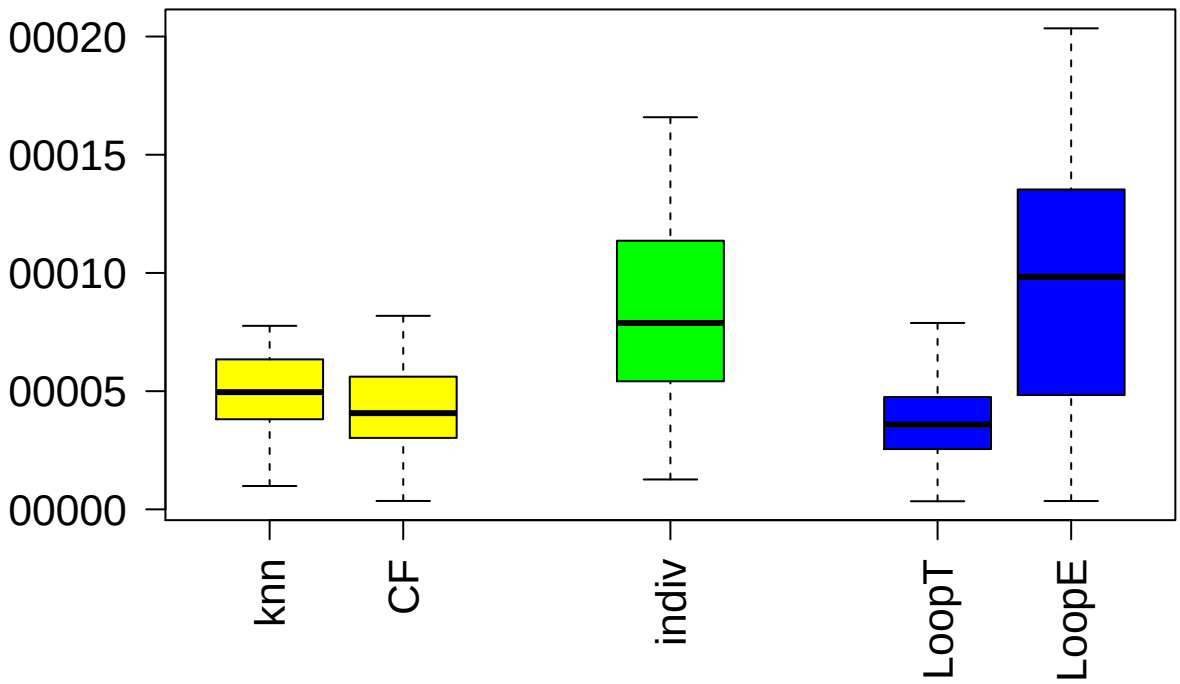
```
meanRSSmat = cbind(QuickTest$meanRSSknn,
                   QuickTest$meanRSSCF, QuickTest$meanRSSindiv,
                   QuickTest$meanRSSLoopT, QuickTest$meanRSSLoopE)

boxplot(meanRSSmat, col = c("yellow","yellow", "green", "blue", "blue"),
        names = c("knn","CF","indiv","LoopT","LoopE"),
        at = c(1,2, 4, 6,7), 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$IMSEknn2, QuickTest$IMSECF2, QuickTest$IMSEindiv2,
                         QuickTest$IMSELoopT2, QuickTest$IMSELoopE2, na.rm=T),
                     max(QuickTest$IMSEknn2, QuickTest$IMSECF2, QuickTest$IMSEindiv2,
                         QuickTest$IMSELoopT2, QuickTest$IMSELoopE2, na.rm=T)))

IMSEmat = cbind(QuickTest$IMSEknn2, QuickTest$IMSECF2, QuickTest$IMSEindiv2,
                         QuickTest$IMSELoopT2, QuickTest$IMSELoopE2)

boxplot(IMSEmat, col = c("yellow","yellow","green","blue","blue"),
        names = c("knn", "CF", "indiv", "LoopT","LoopE"),
        at = c(1,2, 4, 6,7), 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$accmatknn, QuickTest$accmatCF,
                           QuickTest$accmatindiv,
                           QuickTest$accmatLoopT, QuickTest$accmatLoopE, na.rm=T),
                       max(QuickTest$accmatknn, QuickTest$accmatCF,
                           QuickTest$accmatindiv,
                           QuickTest$accmatLoopT, QuickTest$accmatLoopE, na.rm=T)))

accmatmat = cbind(QuickTest$accmatknn, QuickTest$accmatCF,
                   QuickTest$accmatindiv,
                   QuickTest$accmatLoopT, QuickTest$accmatLoopE)

boxplot(accmatmat, col = c("yellow","yellow", "green", "blue", "blue"),
        names = c("knn","CF","indiv","LoopT","LoopE"),
        at = c(1,2, 4, 6,7), ylab = "accuracy", ylim=yrangeaccmat,
        las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)

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

yrangesumcor = range(c(min(QuickTest$sumcorknn2, QuickTest$sumcorCF2,
                           QuickTest$sumcorindiv2,
                           QuickTest$sumcorLoopT2, QuickTest$sumcorLoopE2, na.rm=T),
                       max(QuickTest$sumcorknn2, QuickTest$sumcorCF2,
                           QuickTest$sumcorindiv2,
                           QuickTest$sumcorLoopT2, QuickTest$sumcorLoopE2, na.rm=T)))

sumcormat = cbind(QuickTest$sumcorknn2, QuickTest$sumcorCF2,
```

```
                  QuickTest$sumcorindiv2,
                  QuickTest$sumcorLoopT2, QuickTest$sumcorLoopE2)

boxplot(sumcormat, col = c("yellow","yellow", "green", "blue", "blue"),
        names = c("knn","CF","indiv","LoopT","LoopE"),
        at = c(1,2, 4, 6,7),ylab = "sumcor", ylim=yrangesumcor,
        las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)
```

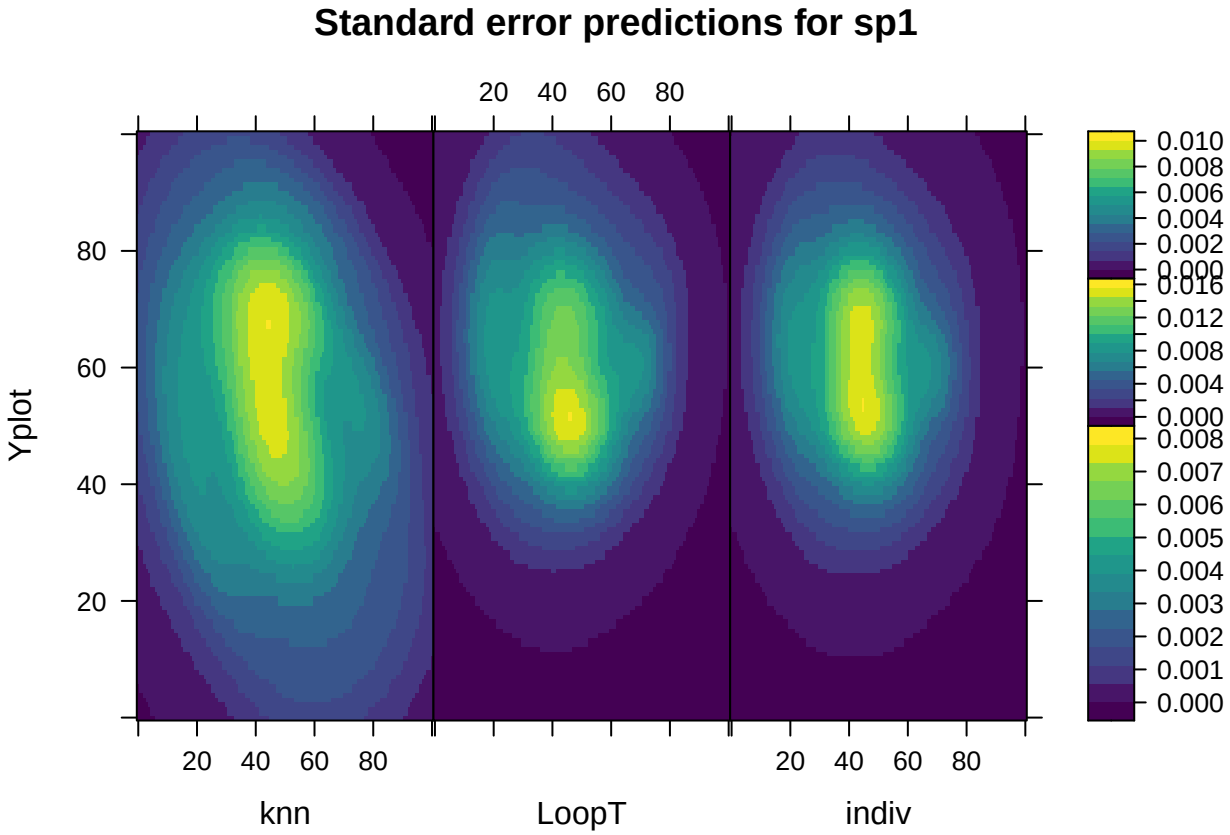


Finally, we can calculate the prediction standard errors from the multiple simulations test:

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

boxplot(unlist(QuickTest$se.knn1), unlist(QuickTest$se.CF1), unlist(QuickTest$se.indiv1),
        unlist(QuickTest$se.LT1), unlist(QuickTest$se.LE1),
        col = c("yellow","yellow", "green", "blue", "blue"),
        names = c("knn","CF","indiv","LoopT","LoopE"),
        at = c(1,2, 4, 6,7), main = "Se for sp1", ylim=yrangeSE,
        las = 2, cex=1.3, cex.lab=1.3, cex.axis=1.3)
```




```
# 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
Yplot
20
40
60
80
knn
LoopT
indiv
0.010
0.008
0.006
0.004
0.002
0.000
0.016
0.012
0.008
0.004
0.000
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000