Control de calidad de clasificacion

Viacheslav Shalisko

12 de junio de 2017

```
verbose = 0

control_parcels_path <- "Control_zones.shp"

raster_path <- "../2016_classif/2016_clasificado.tif"

class_codes <- c(100,101,103,105,110,115,120,130,135,150,155,160,161,190,192,195,197,200)
class_legend <- c("Urbano",
                  "Urbano disperso",
                  "Infraestructura",
                  "SVA",
                  "Agr. riego",
                  "Agr. temporal",
                  "Pastizal",
                  "Matorral",
                  "BEsp",
                  "BTC",
                  "BTSC",
                  "BTemp disp.",
                  "BTemp dens.",
                  "Golf",
                  "AVU",
                  "VAS",
                  "BGal",
                  "Agua")
class_col <- c("gray20",
               "gray30",
               "gray40",
               "lightgray",
               "yellow",
               "lightyellow",
               "palegreen",
               "orange",
               "coral",
               "brown",
               "indianred3",
               "forestgreen",
               "darkgreen",
               "green",
               "seagreen3",
               "blue",
               "darkgreen",
               "navy")
```

Visualización de datos

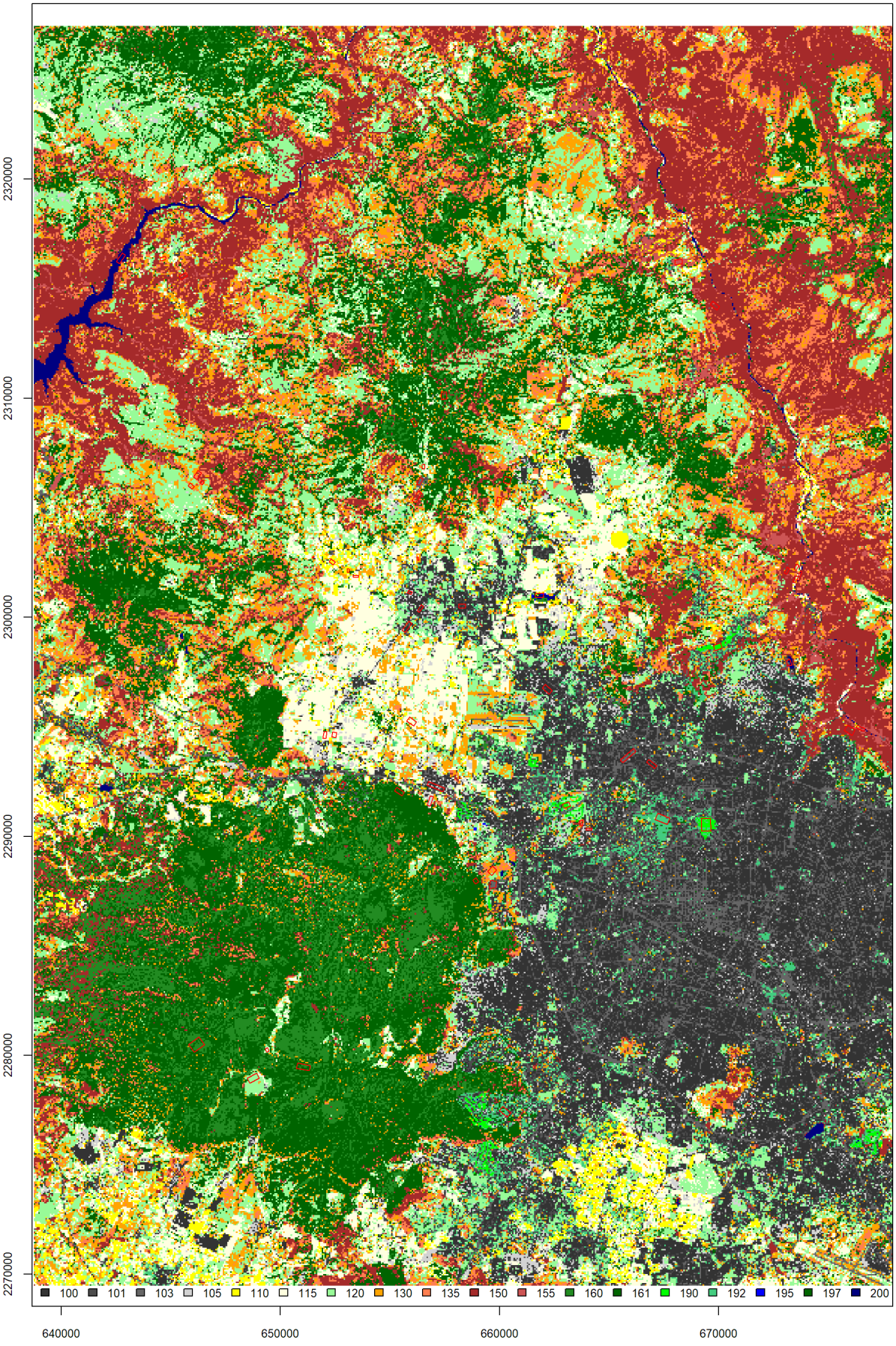
```
control_parcels <- readShapePoly(control_parcels_path)

classification_raster <- raster(raster_path)
dim(classification_raster)
```

```
## [1] 3840 2618    1
```

```
par(cex.axis = 0.8)
plot(classification_raster, axes = TRUE, legend = FALSE,
     col = class_col, breaks = c(class_codes - 1,201),
     main="Clasificación y parcelas de control")
plot(control_parcels, border = "red", add = TRUE)
legend("bottom", horiz = TRUE,
       legend = class_codes, fill = class_col,
       bty = "n", cex = 0.8)
```

Clasificación y parcelas de control



```
## generar matriz de errores en blanco
var_num <- length(class_codes)
error_matrix <- matrix(rep(0,var_num * var_num), nrow = var_num, ncol = var_num)
rownames(error_matrix) <- class_codes
colnames(error_matrix) <- class_codes
```

Muestreo del raster clasificado

```
control_sampling <- extract(classification_raster, control_parcels)
```

Generaión de matriz de confusión

```
for (i in 1:length(control_sampling)) {
  primary_class <- control_parcels@data$Code_1[i]
  secondary_class <- control_parcels@data$Code_2[i]

  if (verbose) {
    cat("Parcela de control: ",as.character(control_parcels@data$GRID_ID[i]),"\n")
    cat("Clase primario = ",primary_class,"\n")
    cat("Clase secundario = ",secondary_class,"\n")
    cat("Pixeles observados = ",control_sampling[[i]],"\n")
    cat("Número de pixeles = ",length(control_sampling[[i]]),"\n\n")
  }

  for (j in 1:length(control_sampling[[i]])) {
    if (control_sampling[[i]][j] == primary_class) {
      # coincidencia de clase primario
      ind_1 <- as.character(primary_class)
      error_matrix[ind_1,ind_1] <- error_matrix[ind_1,ind_1] + 1
    } else {
      if (control_sampling[[i]][j] == secondary_class) {
          # coincidencia de clase secundario
          ind_1 <- as.character(secondary_class)
          error_matrix[ind_1,ind_1] <- error_matrix[ind_1,ind_1] + 1
      } else {
          # valor primario esperado (de control) - filas del matriz
          ind_1A <- as.character(primary_class)
          # valor secundario esperado (de control) - filas del matriz
          ind_1B <- as.character(secondary_class)
          # valor observado (en clasificación) - columnas del matriz
          ind_2 <- as.character(control_sampling[[i]][j])
          error_matrix[ind_1A,ind_2] <- error_matrix[ind_1A,ind_2] + 0.5
          error_matrix[ind_1B,ind_2] <- error_matrix[ind_1B,ind_2] + 0.5
      }
    }
  }
}

# representar matriz de confusión
cat("Matriz de confusión:\n")
```

```
## Matriz de confusión:
```

```
round(error_matrix, digits = 0)
```

```
##      100 101 103 105 110 115  120 130 135  150 155  160  161  190  192 195
## 100 1234  21   8  98   0   3    2   0   0    0   0    0    0    0    4   0
## 101   76 396  13  26   0  16    2  41   8    0   0   14   50    0    6   0
## 103  178  48 942  72   0   3   14  10   1    7   0   26    0    0    2   2
## 105  134  40 181 436   0   1   15  10   0    0   0   22    0    0    0   0
## 110    0   0   0   0 261  19   26  14   0   16  10    0    1    0    0   0
## 115    0   2   0   7   0 956   19  50   4    4   0    0    6    0    0   0
## 120    9   8   0  11   0  44 1610  73 101    8   0    0   17    0    3   0
## 130    2   0   0   2   0  34   18 871  89   50   0    4    4    0    0   0
## 135    1   0   1   2   0   4   32  83 603   52  23    4   30    0    0   0
## 150    2   2   9   0   8   2   22  52   9 1654  23    4   10    0    0   0
## 155    0   2   8   0   8   0    0   0   0    0 417    0    0    0    0   0
## 160    0   0   0   0   0   0    6   8   5   62   0 1159    0    0    0   0
## 161    6  40  38   0   0   0   32  32  14   70   0    4 2502    0    0   0
## 190    3  14   2   4   0   0    0   0   0    0   0    0    0 1417    3   0
## 192   12  48  46   0   0   0   10  10   0    1   0   10   40    0 1320   0
## 195    0   0   3   0   9  24   10   0   0   66  10   18   13    0    0  24
## 197    0   0   0   0   0   0    1  15   0    0   0    1   16    0    0   0
## 200    0   0   3   0   9   5    2   2   1   56   0   18   12    0    0   2
##     197 200
## 100   0   0
## 101   0   0
## 103   0   0
## 105   0   0
## 110   0   0
## 115   0   0
## 120   0   0
## 130   0   0
## 135   0   0
## 150   0   2
## 155   0   2
## 160   0   0
## 161   0   0
## 190   0   0
## 192   0   0
## 195   0   0
## 197   0   0
## 200   0 741
```

Estimación del error de clasificación

```
error_stat <- function(m, legend) {
  n <- sum(m)
  cat("Número de elementos: ",n,"\n")
  d <- diag(m)

  rowsums <- apply(m, 1, sum)
  colsums <- apply(m, 2, sum)
  p <- rowsums / n
  q <- colsums / n
  cat("\nSumas normalizadas en filas (p) y en columnas (q):","\n")
  print(data.frame(legend,p,q))

  # respuesta por clase
  recall <- d / colsums
  # precición por clase
  precision <- d / rowsums
  # Métrica F1 (media harmónica de precición y respuesta)
  f1 <- 2 * precision * recall / (precision + recall)
  cat("\nPrecición y respuesta por clase:","\n")
  print(data.frame(legend,precision, recall, f1))

  # precición general
  accuracy <- sum(d) / n
  cat("\nPrecición general: ",accuracy,"\n")

  # métrica de precición kappa
  expaccuracy = sum(p * q)
  kappa = (accuracy - expaccuracy) / (1 - expaccuracy)
  cat("\nKappa de Cohen: ",kappa,"\n")

}

error_stat(error_matrix, class_legend)
```

```
## Número de elementos:  19572
##
## Sumas normalizadas en filas (p) y en columnas (q):
##              legend           p           q
## 100          Urbano 0.069972410 0.084712855
## 101 Urbano disperso 0.033108522 0.031780094
## 103 Infraestructura 0.066804619 0.064071122
## 105             SVA 0.042892908 0.033568363
## 110      Agr. riego 0.017780503 0.015072553
## 115   Agr. temporal 0.053545882 0.056764766
## 120        Pastizal 0.096259963 0.093245453
## 130        Matorral 0.054899857 0.064990803
## 135            BEsp 0.042739628 0.042662988
## 150             BTC 0.091789291 0.104537094
## 155            BTSC 0.022353362 0.024729205
## 160     BTemp disp. 0.063330268 0.065552831
## 161     BTemp dens. 0.139944819 0.138003270
## 190            Golf 0.073753321 0.072399346
## 192             AVU 0.076512365 0.068414061
## 195             VAS 0.009043532 0.001430615
## 197            BGal 0.001711629 0.000000000
## 200            Agua 0.043557122 0.038064582
##
## Precición y respuesta por clase:
##              legend precision    recall        f1
## 100          Urbano 0.9010588 0.7442702 0.8151941
## 101 Urbano disperso 0.6111111 0.6366559 0.6236220
## 103 Infraestructura 0.7204589 0.7511962 0.7355065
## 105             SVA 0.5193568 0.6636225 0.5826930
## 110      Agr. riego 0.7500000 0.8847458 0.8118196
## 115   Agr. temporal 0.9122137 0.8604860 0.8855952
## 120        Pastizal 0.8545648 0.8821918 0.8681585
## 130        Matorral 0.8106096 0.6847484 0.7423823
## 135            BEsp 0.7208607 0.7221557 0.7215076
## 150             BTC 0.9206791 0.8084066 0.8608979
## 155            BTSC 0.9531429 0.8615702 0.9050461
## 160     BTemp disp. 0.9350545 0.9033515 0.9189296
## 161     BTemp dens. 0.9134721 0.9263236 0.9198529
## 190            Golf 0.9816418 1.0000000 0.9907359
## 192             AVU 0.8814691 0.9858103 0.9307245
## 195             VAS 0.1355932 0.8571429 0.2341463
## 197            BGal 0.0000000       NaN       NaN
## 200            Agua 0.8692082 0.9946309 0.9276995
##
## Precición general:  0.8452381
##
## Kappa de Cohen:  0.8324667
```

Estimación del Kappa de Cohen

```
library(fmsb)
```

```
## Warning: package 'fmsb' was built under R version 3.3.2
```

```
Kappa.test(error_matrix, conf.level=0.95)
```

```
## $Result
##
##  Estimate Cohen's kappa statistics and test the null hypothesis
##  that the extent of agreement is same as random (kappa=0)
##
## data:  error_matrix
## Z = 405.41, p-value < 2.2e-16
## 95 percent confidence interval:
##  0.8269815 0.8379518
## sample estimates:
## [1] 0.8324667
##
##
## $Judgement
## [1] "Almost perfect agreement"
```

```
## obtener la estructura de la capa control_parcels
#str(control_parcels)
#as.character(control_parcels@data$GRID_ID)
#as.character(control_parcels@data$STDID1)
#as.character(control_parcels@data$STDID2)
#as.character(control_parcels@data$ANTRO)
```