ktaucenters package: Robust and efficient Clustering

Juan Domingo Gonzalez, Victor Yohai, Ruben Zamar

2019-07-23

Introduction

This package implements a clustering algorithm similar to kmeans, it has two main advantages:

- The estimator is resistant to outliers, that means that results of estimator are still correct when there are atipycal values in the sample.
- The estimator is efficient, roughly speaking, if there are not outliers in the sample (all data is good), results will be similar than those obtained by a classic algorithm (kmeans)

Clustering procedure is carried out by minimizing the overall robust scale so-called tau scale (see Yohai and Zamar, 1988, doi:10.1080/01621459.1988.10478611 and Gonzalez, Yohai and Zamar 2019 arxiv:1906.08198).

How to use the package ktaucenters

Example 1: behavior when data are clean

```
rm(list=ls())
library(ktaucenters)
#> Loading required package: MASS
#> Loading required package: dplyr
#>
#> Attaching package: 'dplyr'
#> The following object is masked from 'package:MASS':
#>
#>     select
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
#> Loading required package: dbscan
#> Loading required package: GSE
#> Loading required package: Rcpp
```

We generate synthetic data (three cluster well separated), and apply a classic algorithm (kmeans) and the robust ktaucenters. Results and code are shown below.

```
# Generate synthetic data (three cluster well separated)
set.seed(1)
Z <- rnorm(600);
mues <- rep(c(-4, 0, 4), 200)
X <-  matrix(Z + mues, ncol=2)

### Applying the ROBUST algortihm  ####
ktau_output <- ktaucenters(X, K=3,nstart=10)
### Applying the classic algortihm  ####
kmeans_output <- kmeans(X,centers=3,nstart=10)
```

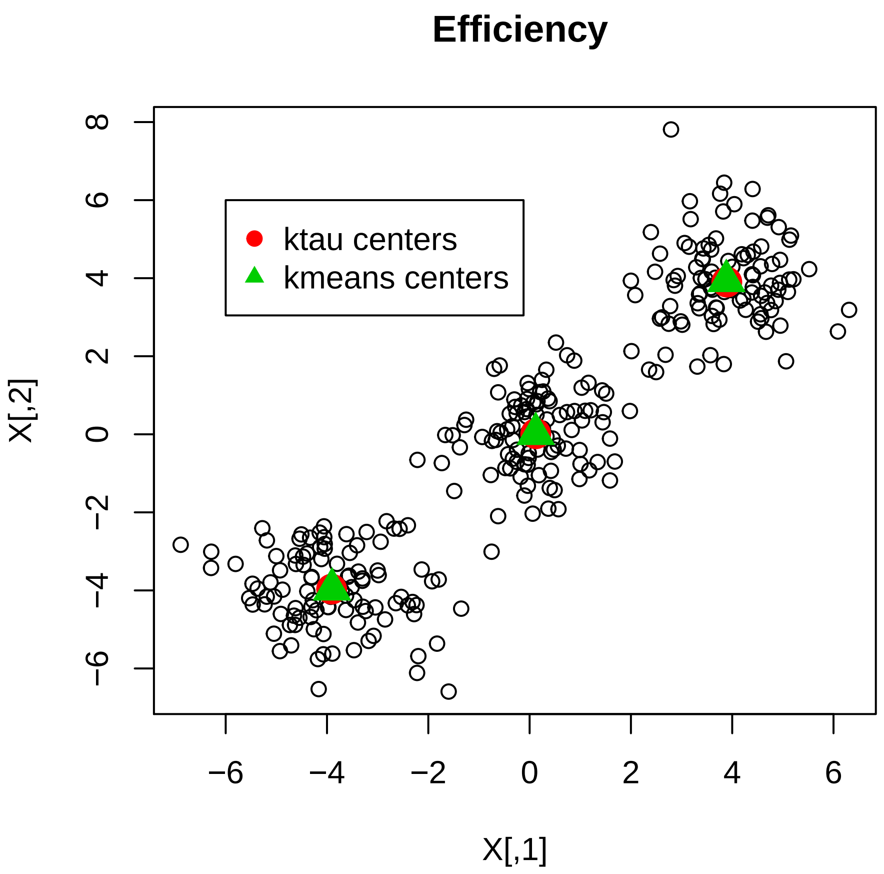


Figure 1: Clean data. Estimated centers by K-means and KTAU-centers algorithms.

```r
### plotting the center results
plot(X,main=" Efficiency")
points(ktau_output$centers,pch=19,col=2,cex=2)
points(kmeans_output$centers,pch=17,col=3,cex=2)
legend(-6,6,pch=c(19,17),col=c(2,3),cex=1,legend=c("ktau centers" ,"kmeans centers"))
```

Figure 1 shows that there are no differeces between kmeans and ktaucenters in clean data.

Example 2: behavior in the presence of outliers

We contaminate the previous data by replacing 60 observations to outliers located in a bounding box that contains the clean data. Then we apply kmeans and ktaucenters algorithms.

```r
# Generate 60 sintetic outliers (contamination level 20%)
X[sample(1:300,60), ] <- matrix(runif( 40, 2* min(X), 2 * max(X) ),
                                ncol = 2, nrow = 60)

### Applying the ROBUST algortihm  ####
ktau_output <- ktaucenters(X, K=3,nstart=10)
### Applying the classic algortihm  ####
```

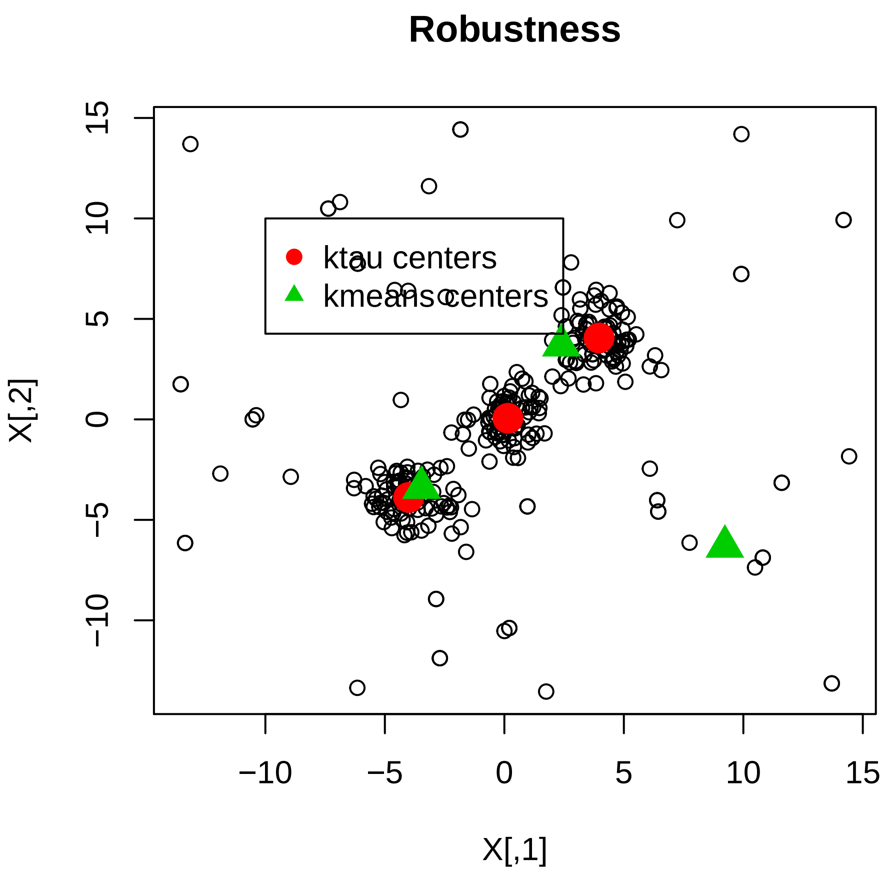


Figure 2: Contaminated data. Estimated centers by K-means and KTAU-centers algorithms.

```r
kmeans_output <- kmeans(X,centers=3,nstart=10)
```

```r
### plotting the estimated centers
plot(X,main=" Robustness ")
points(ktau_output$centers,pch=19,col=2,cex=2)
points(kmeans_output$centers,pch=17,col=3,cex=2)
legend(-10,10,pch=c(19,17),col=c(2,3),cex=1,legend=c("ktau centers" ,"kmeans centers"))
```

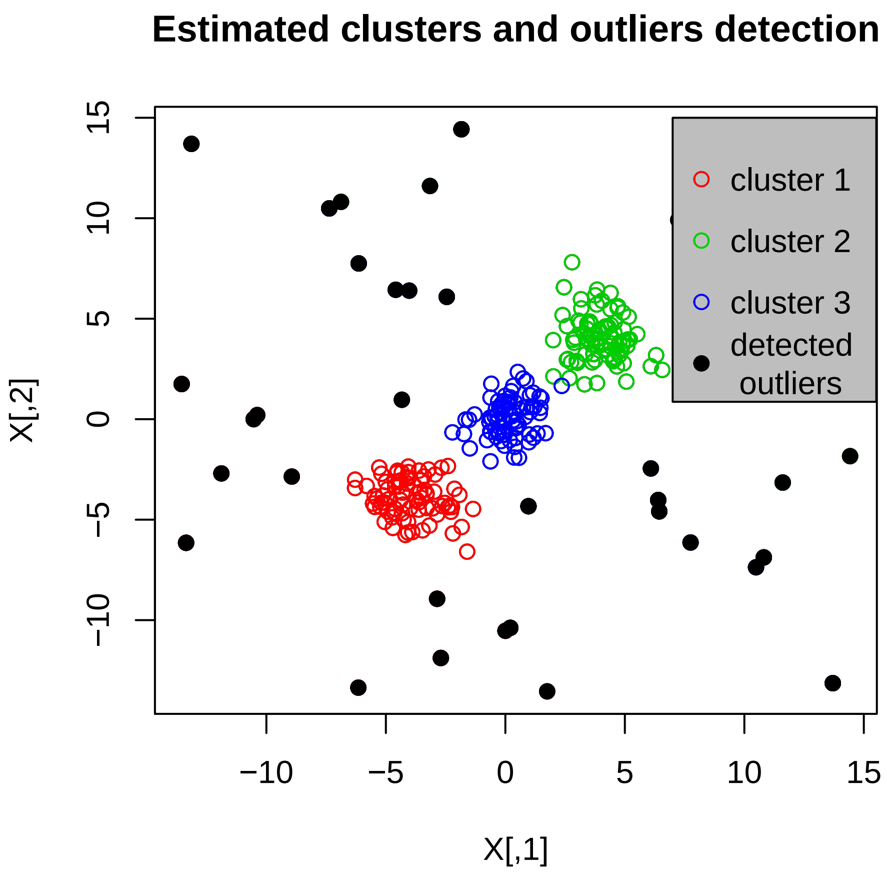
As it can be observed in Figure **??** kmeans center were very influenced by outliers, while ktaucenters results are still razonable.

Example 3: Showing clusters and outliers detection procedure

Continuation from Example 2, for outliers recognition purposes we can see the `ktau_output$outliers` that indicates the indices that may be considered as outliers, on the other hand, the labels of each cluster found by the algorithm are coded with integers between 1 and K (in this case K=3), the variable `ktau_output$clusters` contains that information.

```r
plot(X,main=" Estimated clusters and outliers detection ")
## plottig clusters
for (j in 1:3){
  points(X[ktau_output$cluster==j, ], col=j+1)
}

## plottig outliers
points(X[ktau_output$outliers, ], pch=19, col=1, cex=1)
legend(7,15,pch=c(1,1,1,19),col=c(2,3,4,1),cex=1,
       legend=c("cluster 1" ,"cluster 2","cluster 3","detected \n outliers"),bg = "gray")
```



improved ktaucenters

The algorithm ktaucenter works well under noisy data, but fails when clusters have different size, shape and orientation, an algorithm suitable for this sort of data is `improvektaucenters`. To show how this algorithm works we use the data set so-called M5data from package `tclust: tclust: Robust Trimmed Clustering`, tclust. M5 data were generated by three normal bivariate distributions with different scales, one of the components is very overlapped with another one. A 10% background noise is added uniformly

usage

First we load the data, then, run the `improvedktaucenters` function.

```r
## load non spherical datadata
library("tclust")
data("M5data")
```

```
X=M5data[,1:2]
true.clusters=M5data[,3]
### done ######

#run the function to estimate clusters
improved_output=improvedktaucenters(X,K=3,cutoff=0.95)
```

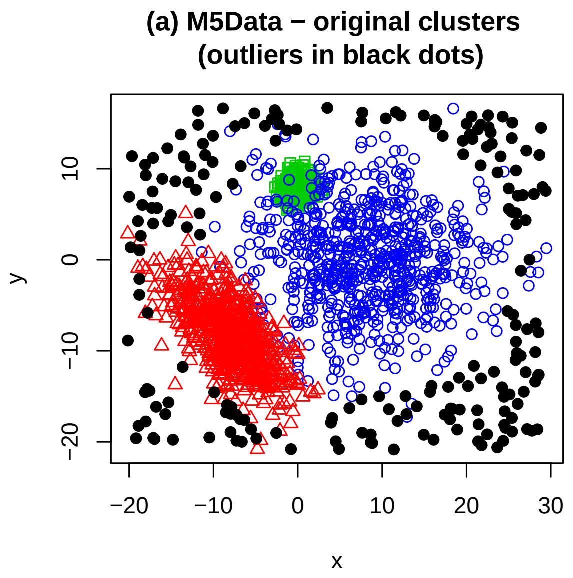
We keep the results in the variable `improved_output`, that is a list that contains the fields `outliers` and `cluster`, among others. For example, we can have access to the cluster labeled as 2 by typing

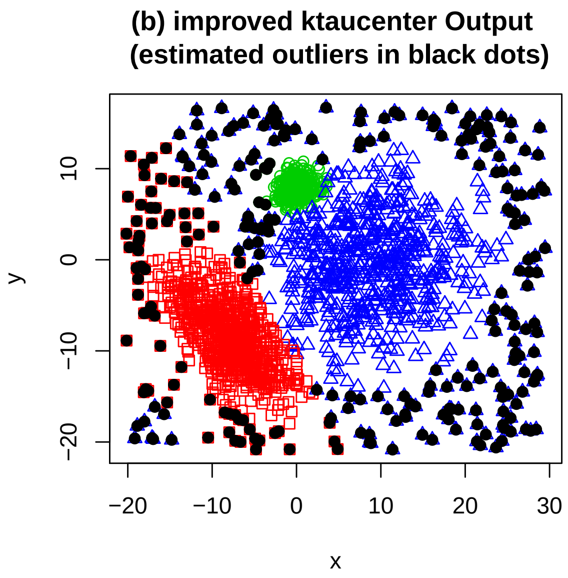
`X[improved_output$cluster==2, ]`.

If we want to know the values of outliers, type

`X[improved_output$outliers, ]`.

By using these commands, it is easy to estimate the original clusters by means of `improvedktaucenters` routine.





Real data application: finding a screw in Mars

Package has a dataset called `mars_screw`, containing the Intensity and Saturation pixels values of a picture from Mars taken from Rover Curiosity.

Data set can be loaded by typing `mars_screw$SI_matrix`, or `mars_screw$geographic_matrix`, thats variables means.

- SI_matrix: A matrix with 5063 rows and 128 columns. Elements 1 to 64 of each row indicate the Saturation values of pixels in a square cell 8 x 8 whereas elements 65 to 128 of each row indicate the cell's Intensity values.
- geographic_matrix: An integer matrix of dimension 5063 x 2, each row indicates each square cell's locations (x-axis y-axis) at the picture.
- screw_index: the index corresponding to the screw observation (screw_index=4180)

In order to find the screw, in a first stage, we run the clustering algorithm over the matrix A, with the aim to separate the differents materials presented in the picture

```r
A <- mars_screw$SI_matrix;
B <- mars_screw$geographic_matrix
screw_index <-mars_screw$screw_index
##
ret1=ktaucenters(X=A,K=3);
##
```

To find the screw candidate, we run a clustering procedure on geographic_matrix, for each cluster determined at the previous step. Outliers will be those that are the furthest observations regarding to the gruops they were assigned. The procedure to find them is shown below.

```r
screw_candidate_index=c()
for (j in 1:3){
  dj=ktaucenters(B[ret1$cluster==j, ],K=5,nstart=1,startWithROBINPD = FALSE)$di;
  jcandidate=dj==max(dj);
  jcandidate=which(ret1$cluster==j)[dj==max(dj)]
  screw_candidate_index=c(screw_candidate_index,jcandidate)
}
```

Figure **??**, shows locations of three candidates to be the screw.

```r
plot(B, type="n" )
for (j in 1:3){
#  col1=which(orderInd==j)
  col1=j
  points(B[ret1$cluster==j,],col=col1+1,pch=19)
}
points(B[screw_candidate_index,1],B[screw_candidate_index,2],col=6,pch=1,cex=3,lwd=3)
```

From package-data mars_screw to image

It is possible to see the image from package-data and screw location by running the following code. To see screw location we use the `mars_screw$screw_index` information.

```r
####### Reconstruction image from package-data #####
d=8;
intensityvar=matrix(0,max(B[,1]+1)*d,max(B[,2])*d)
svar=matrix(0,max(B[,1]+1)*d,max(B[,2])*d)

for (l in 1:dim(A)[1]){
  i=mars_screw$geographic_matrix[l,1]
  j=mars_screw$geographic_matrix[l,2]
  posi=(i-1)*d;
  posj=(j-1)*d;
  intensityvar[posi+(1:d), posj +(1:d)]= mars_screw$SI_matrix[l,(d^2+1):(2*d^2)]
  svar[posi+(1:d), posj +(1:d)]= mars_screw$SI_matrix[l,1:(d^2)]
  #A[l,]=c(auxs[,],auxi[,])
  #B[l,]=c(i,j)
}
svar[svar<0]=0
```

The following transformation from indices ij locations in B matrix to xy coordinates at the picture is useful to plot the screw location at the image.

```r
nrowIm=495 #dim(myjpg)[1] #(nrowIm pixels is equivalent to 1 en the  coordinate system)
ncolIm=664 #dim(myjpg)[2] #(ncolIm pixeles pixels is equivalent to 1 en the  coordinate system)
```

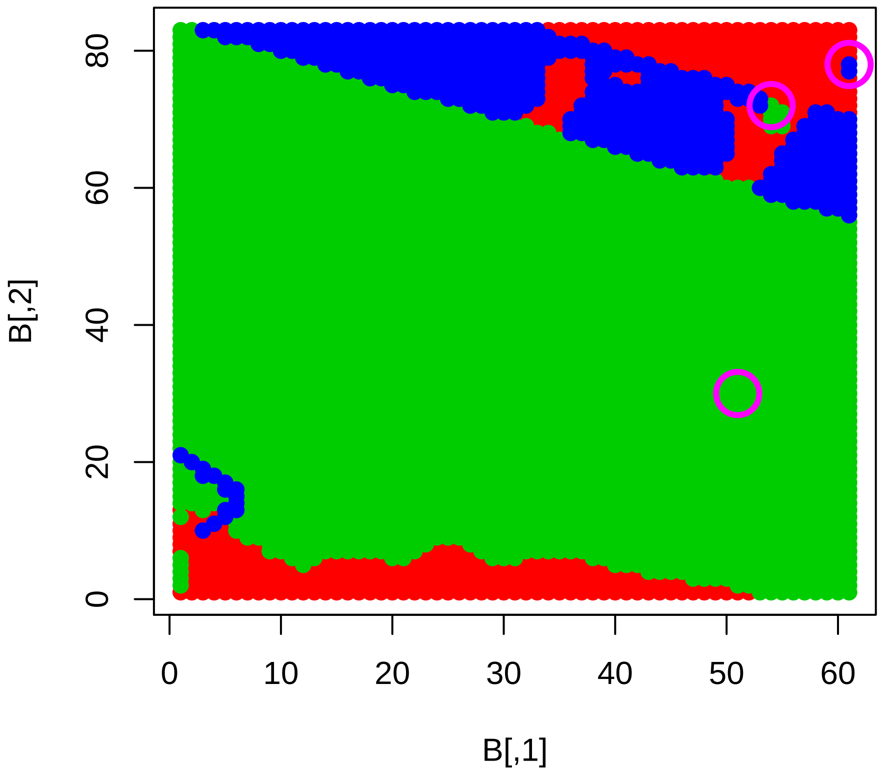


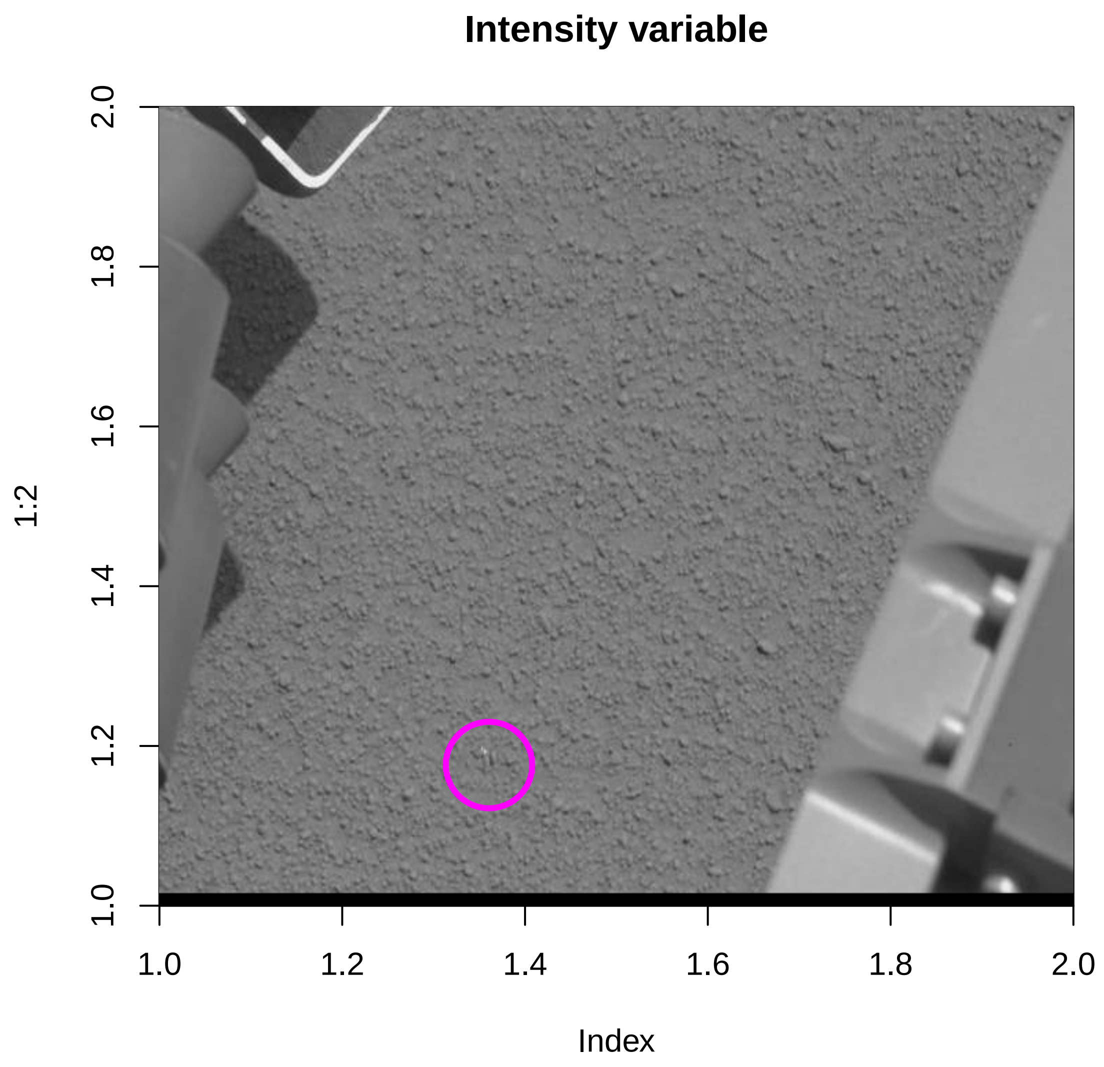
Figure 3: Plot of Geographic sub matrices, pink circles are the outliers candidates in the geographic matix space.

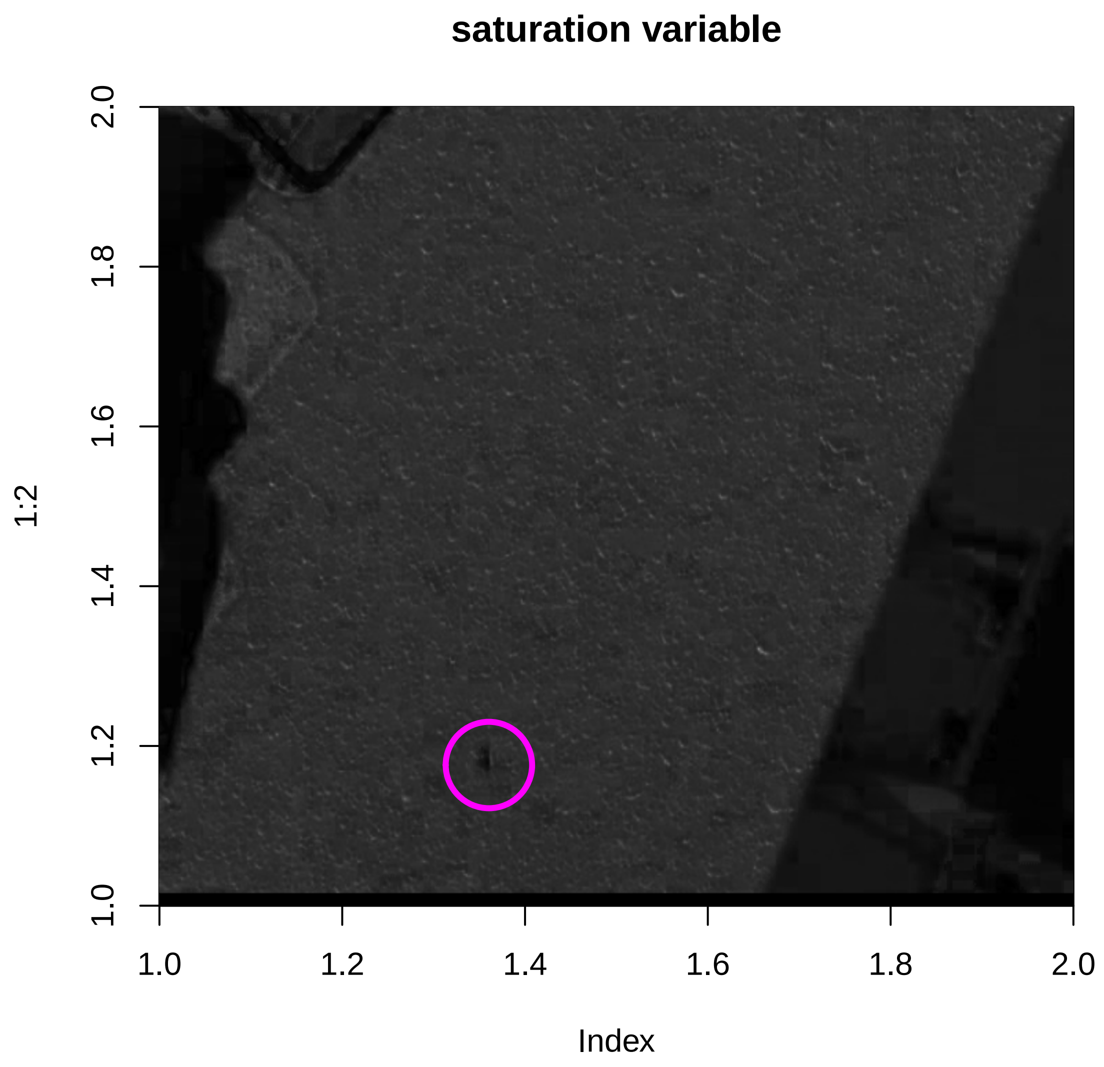
```
index2ejx=function(indexj){(1/(ncolIm-1))*(indexj-1) +1 }
index2ejy=function(indexi){((2-1)/(1-nrowIm))*(indexi-nrowIm) +1 }
mapIndex2coord=function(filaij){ c(index2ejx(filaij[2]),index2ejy(filaij[1]))}
#transforming pixel position ij to equivalent xy values at the image
xyscrew=mapIndex2coord(d*mars_screw$geographic_matrix[screw_index, ])
```

We plot the variables intensity and saturation as a raster Image trhough the `rasterImage` function from `graphics` package, (a pixel with high/low
saturation value is drawn in white/black respectively)
The location of screw is show in pink circle. To transform the `ij` data from matrix B to its equivalent `xy` mapIndex2coord was used.

```
plot(1:2, type="n",yaxs="i",xaxs="i",main="Intensity variable")
rasterImage(intensityvar, 1, 1, 2, 2)
points(xyscrew[1],xyscrew[2],col=6,pch=1,cex=6,lwd=3)

plot(1:2, type="n",yaxs="i",xaxs="i",main="saturation variable")
rasterImage(svar, 1, 1, 2, 2)
points(xyscrew[1],xyscrew[2],col=6,pch=1,cex=6,lwd=3)
```


Intensity variable
2.0
1.8
1.6
1.4
1.2
1.0
1:2
1.0
1.2
1.4
1.6
1.8
2.0
Index



From image to mars_screw matrix data

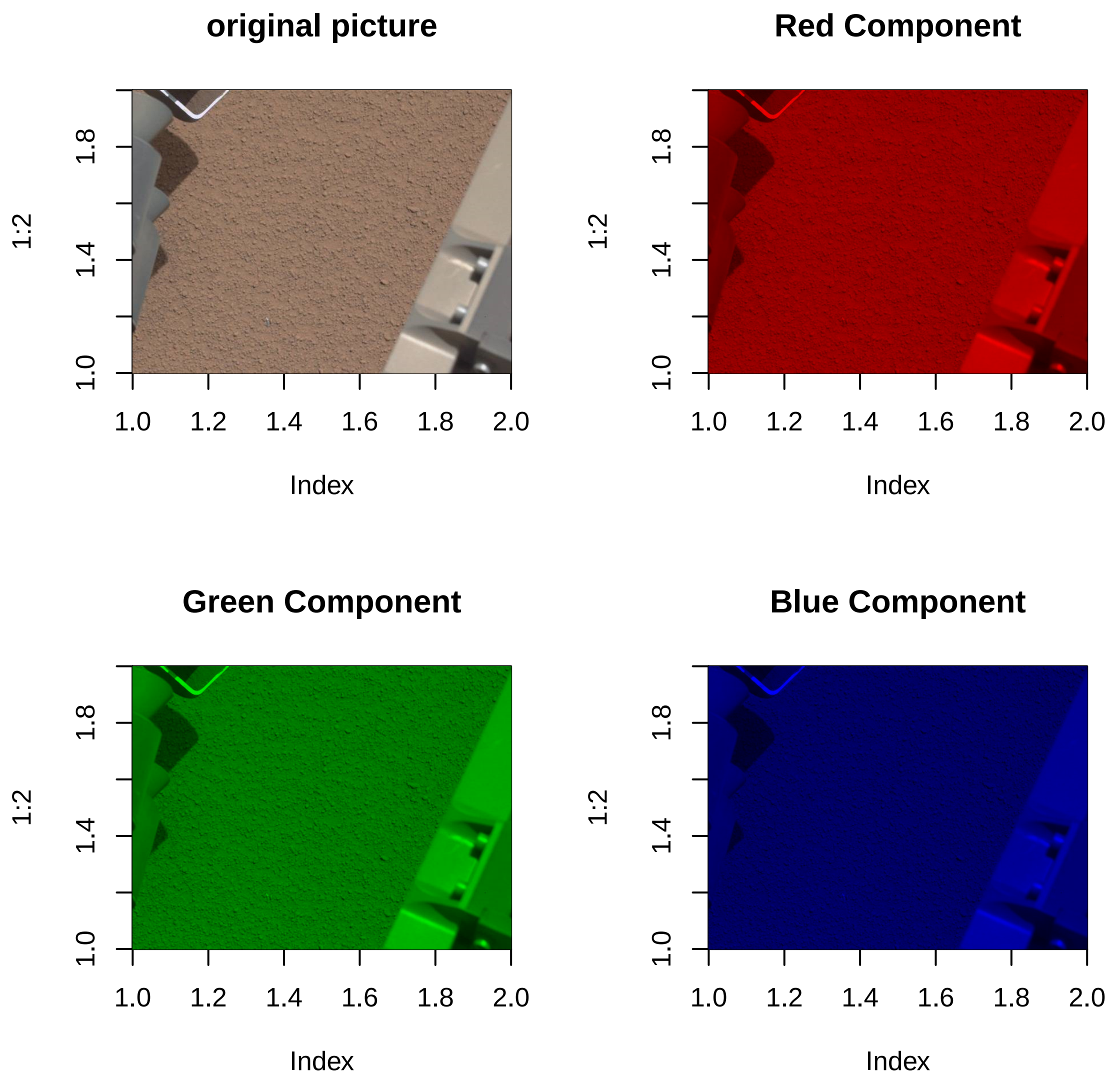
First of all, to see the original Image, and its RGB (Red, Gren, Blue) chanels use

```r
### Rebuilding data from image
library("jpeg")
imageFile="screw2.jpeg"
par(mfrow=c(2,2))
myjpg=readJPEG(imageFile)
plot(1:2, type="n",yaxs="i",xaxs="i",main="original picture")
rasterImage(myjpg, 1, 1, 2, 2)

plot(1:2, type="n",yaxs="i",xaxs="i",main="Red Component")
myjpgR=myjpg;
myjpgR[,,2]=myjpgR[,,3]=0
rasterImage(myjpgR, 1, 1, 2, 2)
```

```
plot(1:2, type="n",yaxs="i",xaxs="i",main="Green Component")
myjpgG=myjpg;
myjpgG[,,1]=myjpgG[,,3]=0
rasterImage(myjpgG, 1, 1, 2, 2)

plot(1:2, type="n",yaxs="i",xaxs="i",main="Blue Component")
myjpgB=myjpg;
myjpgB[,,1]=myjpgB[,,2]=0
rasterImage(myjpgB, 1, 1, 2, 2)
```



The images corresponding to the three chanels are quite similar, this is because the RGB pixels values are usually highly correlated. On the other hand, if we want to rebuild the data-set from the original picture, the following code can be used.

```
# To reconstruct the data from source image.
XXX=cbind(myjpg[,,1],myjpg[,,2],myjpg[,,3])
# define functions Intensity and saturation
```

```r
Icolor=function(COLOR){((COLOR[1]+COLOR[2]+COLOR[3]))/3}
Scolor=function(COLOR){
  Iaux=Icolor(COLOR)
  ret=0
  if(!(Iaux==0)){ret=1 - min(COLOR[1],COLOR[2],COLOR[3])/Iaux}
  if(Iaux==0){ret=0}
  ret
}

I=apply(XXX,1,Icolor)
S=apply(XXX,1,Scolor)

myjpgHSI=myjpg
myjpgHSI[,,2]=S
myjpgHSI[,,3]=I
scomp=myjpgHSI[,,2];
icomp=myjpgHSI[,,3];
d=8
p=d^2
nrowIm=dim(myjpg)[1] #(nrowIm pixeles equivale a 1 en mi sist. de coordenadas)
ncolIm=dim(myjpg)[2] #(ncolIm pixeles equivale a 1 en mi sist. de coordenadas)
NporM=ncolIm*nrowIm

# transforming each d-square cell into an array of size 2*dxd
A=matrix(0,ncol=2*p,nrow=NporM/p)
B=matrix(0,ncol=2,nrow=NporM/p)
l=1;
for (i in 1:(nrowIm/d)){
  for (j in 1:(ncolIm/d)){
    posi=(i-1)*d;
    posj=(j-1)*d;
    auxs=scomp[posi+(1:d), posj +(1:d)];
    auxi=icomp[posi+(1:d), posj +(1:d)];
    A[l,]=c(auxs[,],auxi[,])
    B[l,]=c(i,j)
    l=l+1;
  }
}

A=A[1:(l-1),]
```

This is a check whether A and mars_screw$SI_matrix are equal

```r
mean(abs(A-mars_screw$SI_matrix))==0
#> [1] FALSE
```