ktaucenters package: Robust and efficient Clustering

Juan Domingo Gonzalez, Ruben Zamar, Victor Yohai

2019-02-21

<! Cosas que faltan: 1. caption en figuras 2. referencias a escala tau, al paquete, tesis, y paper futuro. 3. reproducir datos de paper. 4. poner algun escenario de simulacion. !>

## Introduction

This package implements a kind of kmeans algorithm, 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 obtain 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].

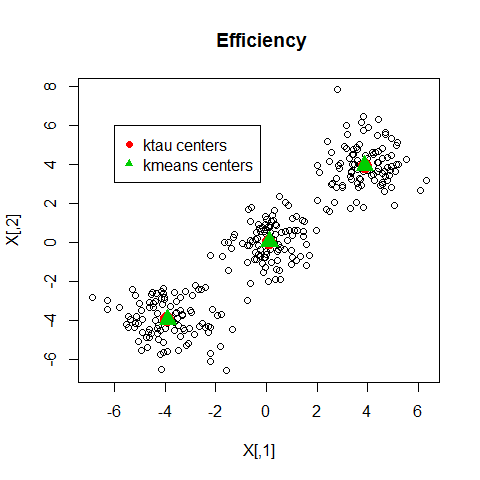
## How to use the package ktaucenters

### 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  
#> Loading required package: ggplot2

Como se vera en la figura

# Generate Sintetic 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)  
  
### 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"))



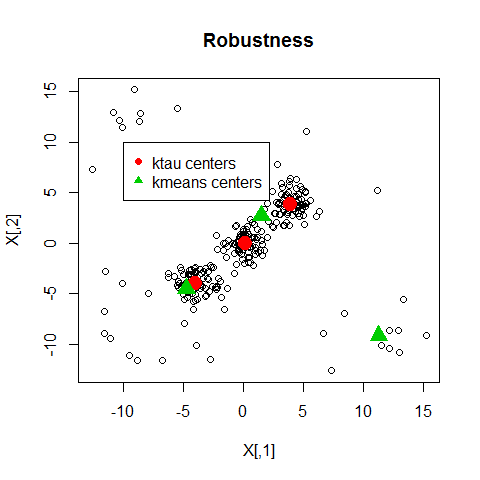
Your asdf asdf figure caption.

As was stated previously, if the data are clean, there are no differeces between kmeans and ktaucenters.

### Example 2: behavior in the presence of outliers

# 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 ####  
kmeans\_output <- kmeans(X,centers=3,nstart=10)

### 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"))

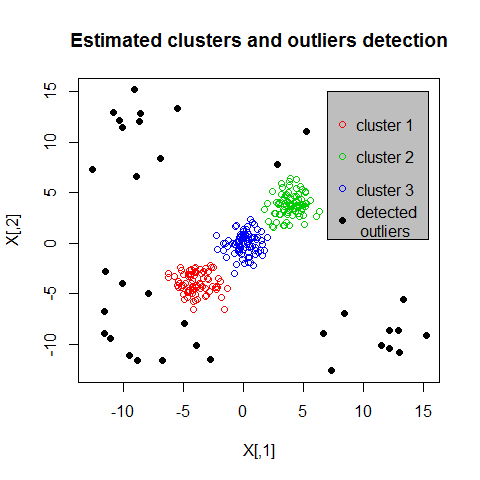


As it can be observed, kmeans center were hardly influencied by outliers, while ktaucenters results are still razonable.

### Example 3: Showing clusters and outliers detection procedure

For outliers recognition purposes we can see the ktau\_output$outliers that indicates the indices that may considered as outliers, on the other hand the labels of each cluster found by the algorithm is ktau\_output$clusters

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 work 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 ref()

“A bivariate data set obtained from three normal bivariate distributions with different scales and proportions 1:2:2. One of the components is very overlapped with another one. A 10% background noise is added uniformly distributed in a rectangle containing the three normal components and not very overlapped with the three mixture components.”

### usage

## 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)

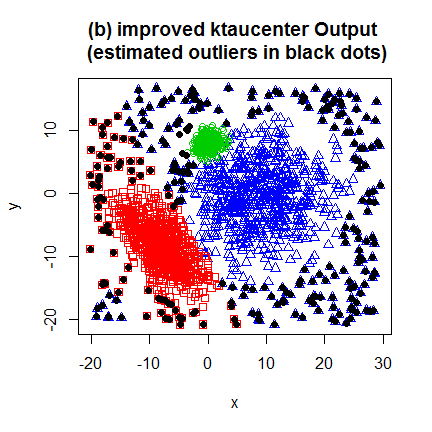
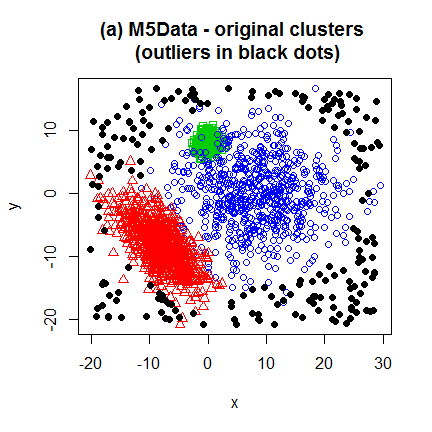
Variable improved\_output is a list that contains the fields outliers and cluster. We can have access to the cluster labeled as 2 by typing

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

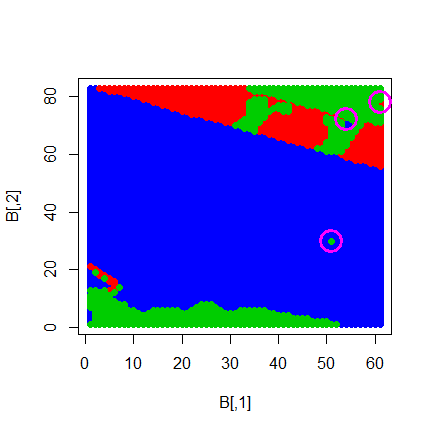
Package has a dataset called mars\_screw, containing the Intensity and Saturation pixels values of a picture from Mars taken from Rover Curiosity.

In order to look the screw, in a first stage we do clustering on the matrix A

A <- mars\_screw$SI\_matrix;  
B <- mars\_screw$geographic\_matrix  
screw\_index <-mars\_screw$screw\_index  
##   
ret1=ktaucenters(X=A,K=3);  
##

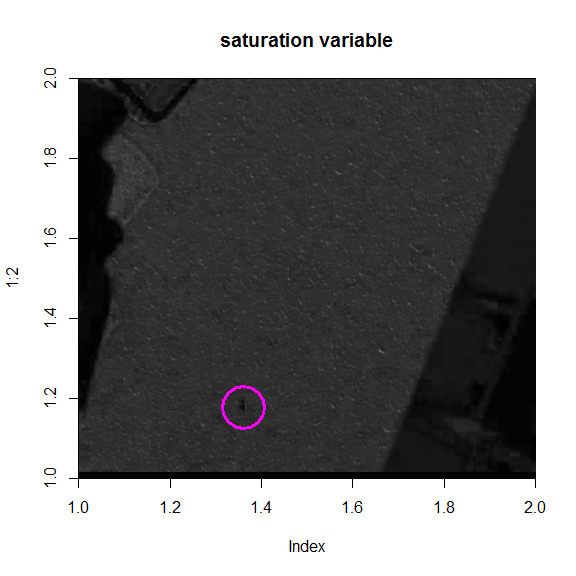
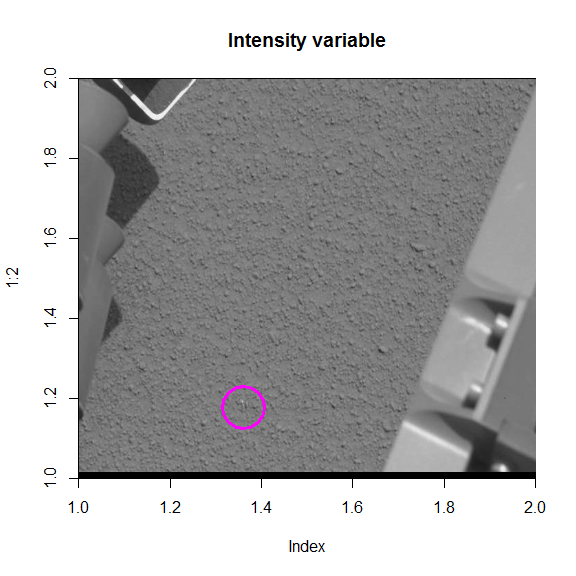
Then screw candidate will be those that are geographical outliers regarding to the gruops they were assigned.

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)  
}  
  
plot(B, type="n" )  
for (j in 1:3){  
 col1=which(orderInd==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)



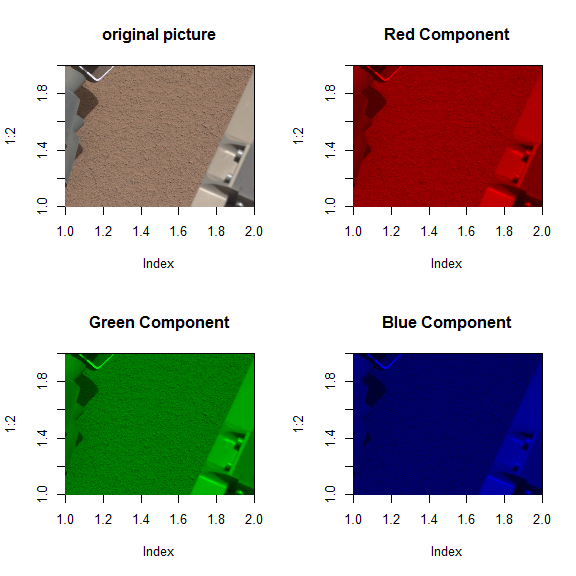
It is possible to see the image from package-data and screw location by running the following code. To see screw location we use the mars\_screw$screw\_index information.

####### 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 are transformation from indices ij locations in B matrix to xy coordinates at the picture   
  
nrowIm=495 #dim(myjpg)[1] #(nrowIm pixeles equivale a 1 en mi sist. de coordenadas)  
ncolIm=664 #dim(myjpg)[2] #(ncolIm pixeles equivale a 1 en mi sist. de coordenadas)  
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, ])  
  
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)



To see the original Image, and its RGB chanels use

### 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 three chanels are quite similar, that is consequence of that the RGB pixels values are 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   
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),]  
  
  
  
dim(A)  
#> [1] 5063 128  
library("ktaucenters")  
dim(A)  
#> [1] 5063 128  
dim(mars\_screw$SI\_matrix)  
#> [1] 5063 128  
#checking that elements A and mars\_screw$SI\_matrix are equal   
mean(abs(A-mars\_screw$SI\_matrix))  
#> [1] 0.1203398