Математички факултет

Универзитет у Београду

Статистички софтвер 3

Семинарски рад

мај 2019.
Београд

Аутор

Марија Костић 286/14

Ментор

Данијел Суботић

Садржај

Рад са сликама

Први задатак

За слике које се налазе у фолдеру $prvi\_zadatak$ потребно је:

(а) Одредити угао ротације геометријске фигуре у односу на x-осу.

(б) Изротирати фигуру, т.д. доња страница буде паралелна са x –осом.

(в) Сачувати новодобијену слику.

Фигуре су само правоугаоници произвољних димензија и боја.

Пакет "$imager$" садржи велики број функција помоћу којих можемо да радимо са сликама.

За почетак ћемо инсталирати и учитати тај пакет.

```
#install.packages("imager")
library(imager)

## Loading required package: magrittr

##
## Attaching package: 'imager'

## The following object is masked from 'package:magrittr':
##
##     add

## The following objects are masked from 'package:stats':
##
##     convolve, spectrum

## The following object is masked from 'package:graphics':
##
##     frame

## The following object is masked from 'package:base':
##
##     save.image

require(imager)
```

Сада ћемо учитати све слике из фолдера $prvi\_zadatak$ како би могли даље да радимо са њима.

```
setwd("C:/Users/Korisnik/Desktop/seminarski/prvi_zadatak")
files <- list.files(path=getwd(),all.files=T, full.names=F, no.. = T)
slike <- lapply(files, load.image)
```

Правимо функцију f помоћу које ћемо пронаћи фигуре на слици. Идеја је да пиксел по пиксел гледамо вредности боје које су веће од средње вредности и фиксирамо 4 тачке које су уствари темена правоугаоника.

Када посматрамо правоугли троугао са теменима $(x_1, y_1), (x_2, y_2)$ и (x_2, y_1) тангенс угла биће нам однос наспрамне и налегре катете $\frac{y}{x}$, где је $y = y_2 - y_1$ и $x = x_2 - x_1$.

Угао ротације ће бити $arctg\frac{y}{x}$, да би тај угао претворили у степене морамо да помножимо са $\frac{180}{\pi}$.

Када смо одредили координате темена, одмах можемо да нађемо центар правоугаоника.

Функција f враћа угао ротације и координате x и y центра правоугаоника.

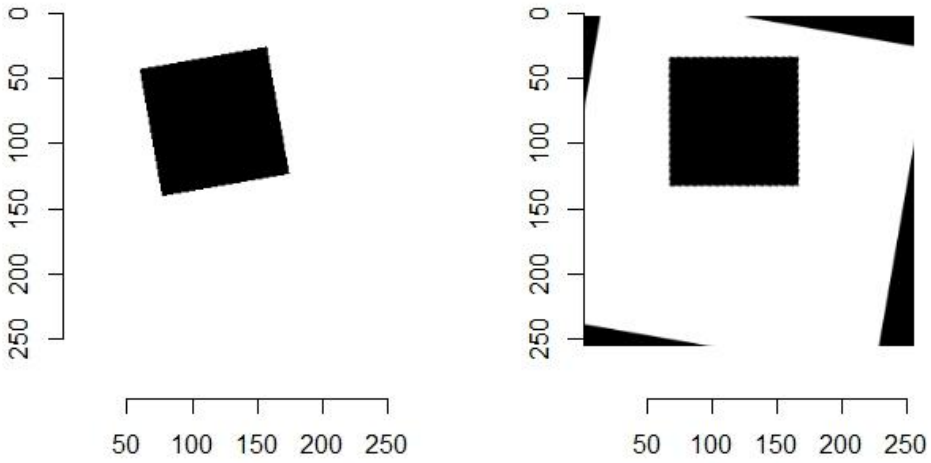
```
f <- function(slika,n)
{
  # n nam predstavlja kanal boje
  slika <- renorm(slika) # primenom funkcije "renorm" dobijamo normalizovani oblik s
like

  q <- slika[,,1,n]
  dimenzija <- dim(q)
  # q je matrica dimenzije 256x256
  srednja_vrednost <- mean(q) # srednja vrednost boje
  indikator <-(q>srednja_vrednost) # gledamo samo one vrednosti koje su vece od sred
nje vrednosti

  levo <-dimenzija # tacka skroz levo
  gore <-dimenzija # najvisa tacka
  desno <-c(0,0) # tacka skroz desno
  dole <- c(0,0) # najniza tacka

  for(i in 1:dimenzija[1])
  {
    for(j in 1:dimenzija[2])
    {
      if(indikator[i,j]!=indikator[1,1])
      {
        if(i<levo[1]) levo <- c(i,j)
        if(i>desno[1]) desno <- c(i,j)
        if(j<gore[2]) gore <- c(i,j)
        if(j>dole[2]) dole <- c(i,j)
      }
    }
  }
  # razlika od pozadine je tacka koja nam treba

  C <- (gore+dole+levo+desno)/4 #centar
  xy<- abs(dole-desno)
  ugao_rotacije <- atan2(xy[2],xy[1])*180/pi
  return(c(ugao_rotacije,C))

}
```

Сада ћемо направити фолдер у коме ћемо сачувати новодобијене изротиране слике.

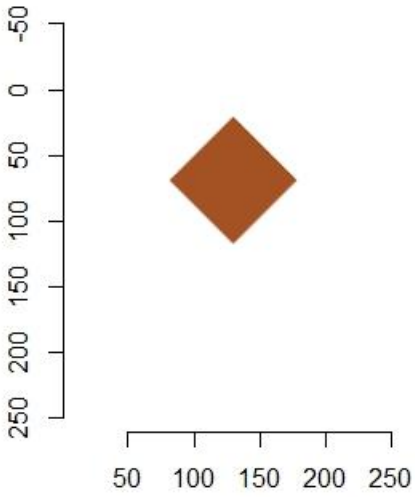
```
dir.create(file.path(getwd(),"slike"))
```

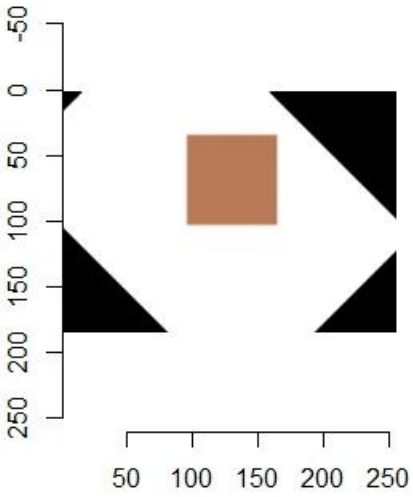
Функција "$imrotate$" омогућава ротацију слике у односу на дати угао у степенима, а функција "$round$" заокружује број на одређени број децималних места.

```
par(mfrow=c(1,2))

slika1 <- slike[[1]]
plot(slika1)
f(slika1,2)[1]

## [1] 9.841132

rotirana_slika1 <- imrotate(slika1,f(slika1,2)[1],round(f(slika1,2)[2]),round(f(slik
a1,2)[3]))
plot(rotirana_slika1)
```

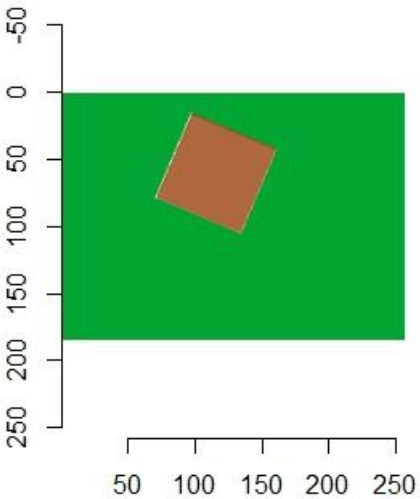


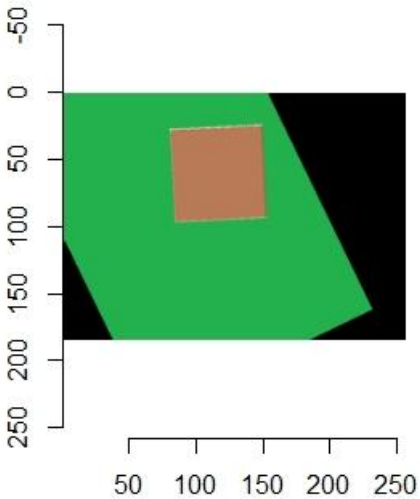

```
save.image(rotirana_slika1,paste("slike\\",strsplit(files[1],".",fixed = T)[[1]][1],
".png",sep=""))

slika2 <- slike[[2]]
plot(slika2)
f(slika2,2)[1]

## [1] 45

rotirana_slika2 <- imrotate(slika2,f(slika2,2)[1],round(f(slika2,2)[2]),round(f(slik
a2,2)[3]))
plot(rotirana_slika2)
```



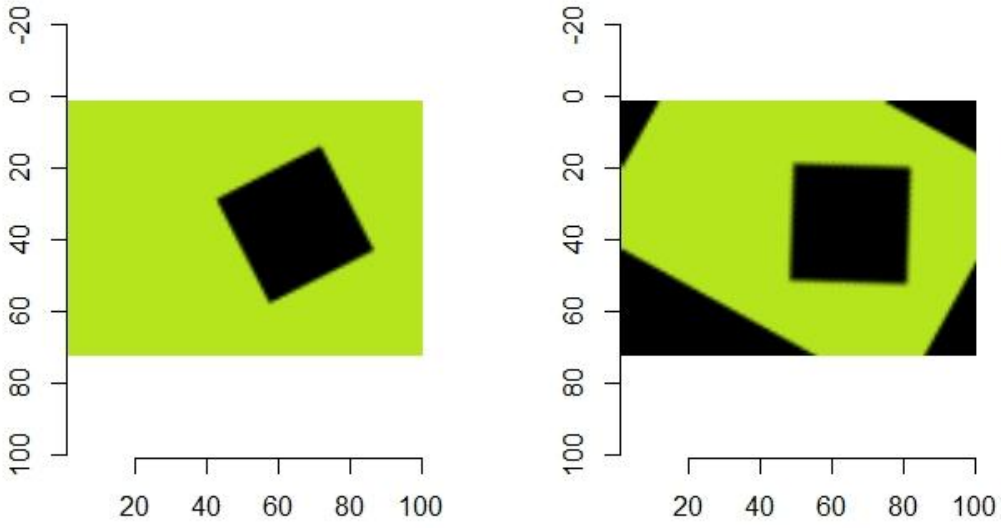



```
save.image(rotirana_slika2,paste("slike\\",strsplit(files[2],".",fixed = T)[[1]][1],
".png",sep=""))

slika3 <- slike[[3]]
plot(slika3)
f(slika3,2)[1]

## [1] 64.17901

rotirana_slika3 <- imrotate(slika3,f(slika3,2)[1],round(f(slika3,2)[2]),round(f(slik
a3,2)[3]))
plot(rotirana_slika3)
```

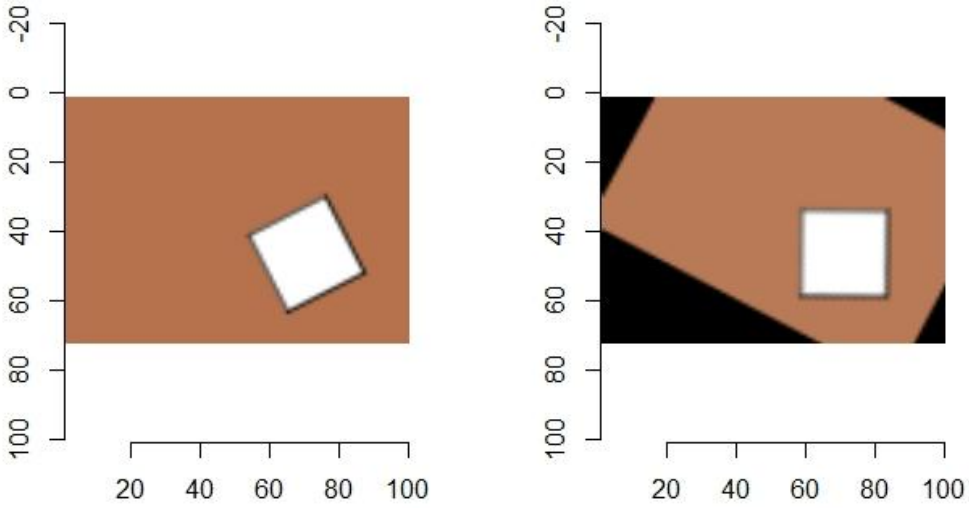




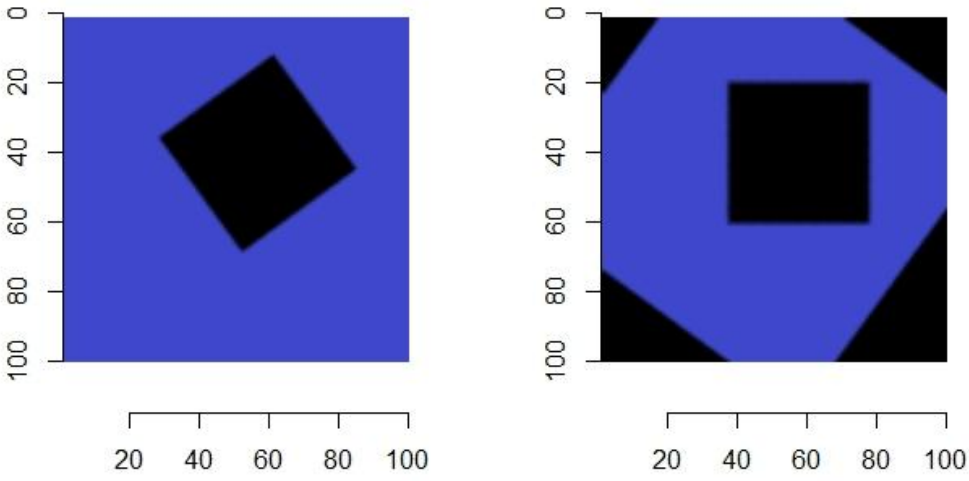
```
save.image(rotirana_slika3,paste("slike\\",strsplit(files[3],".",fixed = T)[[1]][1],
".png",sep=""))

slika4 <- slike[[4]]
plot(slika4)
f(slika4,2)[1]

## [1] 28.88658

rotirana_slika4 <- imrotate(slika4,f(slika4,2)[1],round(f(slika4,2)[2]),round(f(slik
a4,2)[3]))
plot(rotirana_slika4)
```



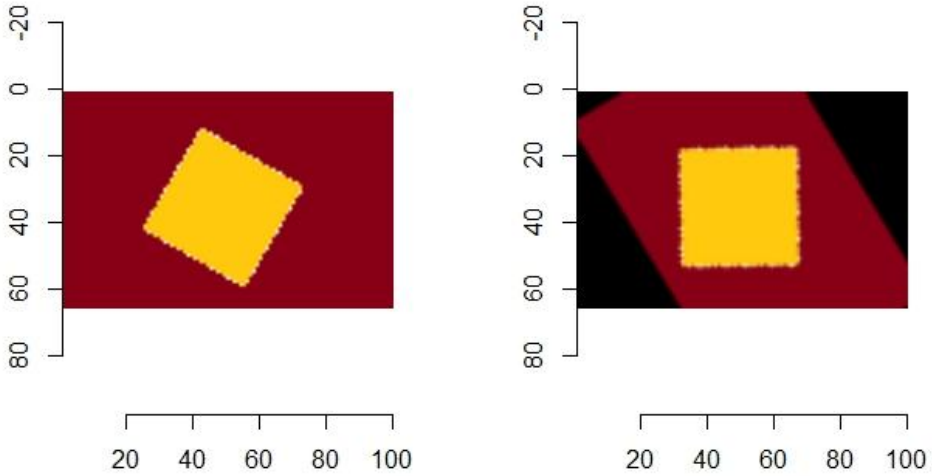

```
save.image(rotirana_slika4,paste("slike\\",strsplit(files[4],".",fixed = T)[[1]][1],
".png",sep=""))

slika5 <- slike[[5]]
plot(slika5)
f(slika5,2)[1]

## [1] 27.64598

rotirana_slika5 <- imrotate(slika5,f(slika5,2)[1],round(f(slika5,2)[2]),round(f(slik
a5,2)[3]))
plot(rotirana_slika5)
```




```
save.image(rotirana_slika5,paste("slike\\",strsplit(files[5],".",fixed = T)[[1]][1],
".png",sep=""))

slika6 <- slike[[6]]
plot(slika6)
f(slika6,2)[1]

## [1] 36.02737

rotirana_slika6 <- imrotate(slika6,f(slika6,2)[1],round(f(slika6,2)[2]),round(f(slik
a6,2)[3]))
plot(rotirana_slika6)
```



```
save.image(rotirana_slika6,paste("slike\\",strsplit(files[6],".",fixed = T)[[1]][1],
".png",sep=""))

slika7 <- slike[[7]]
plot(slika7)
f(slika7,2)[1]

## [1] 59.03624

rotirana_slika7 <- imrotate(slika7,f(slika7,2)[1],round(f(slika7,2)[2]),round(f(slik
a7,2)[3]))
plot(rotirana_slika7)
```




```
save.image(rotirana_slika7,paste("slike\\",strsplit(files[7],".",fixed = T)[[1]][1],
".png",sep=""))
```

Други задатак

За слике које се налазе у фолдеру $drugi\_zadatak$ потребно је:

(a) Написати функцију која проверава да ли је на слици тачно један црни или бели правоугаоник. Уколико слика садржи више од једног поравоугаоника или ако је неке друге боје – потребно је да функција врати грешку, тј. да обавести да на слици није један правоугаоник или није потребне боје.

(б) Уколико је на слици један правоугаоник црне или беле боје, функција треба да врати координате његових темена, у произвољном редоследу.

(в) Поновити део под (a) и део под (б) за круг, с тим што у делу под (б) уместо координата темена треба да се испишу координате центра круга и његов пречник у пикселима.

Фигуре могу да буду било које.

Учитаћемо слике из подфолдера $pravougaonici$ фолдера $drugi\_zadatak$ како би могли даље да радимо са њима.

```
setwd("C:/Users/Korisnik/Desktop/seminarski/drugi_zadatak/pravougaonici")
files <- list.files(path=getwd(),all.files=T, full.names=F, no.. = T)
pravougaonici <- lapply(files, load.image)
```

Правимо функције које ће да одређују координате темена правоугаоника са слика.

```
# Tacka dole levo
T1 <- function(slika,k)
{
  n1 = dim(slika)[1] # broj piksela na x-osi
  n2 = dim(slika)[2] # broj piksela na y-osi
  for(j in n2:1)
  {
    # Iduci odozdo ka gore, trazimo prvu tacku koja se razlikuje vise od 0.1% od poz
adine(k)
    if(mean(slika[,j]!=k) > 0.001)
    {
      for(i in 1:n1) # duz x-ose trazimo tacku koja se ralikuje
      {
        if(slika[i,j]!=k)
        {
          tacka <- c(i,j) #Prva tacka koja se razlikuje od pozadine je ona koja nam
treba
          return(tacka)
        }
      }
    }
  }
}

# Tacka dole desno
T2 <- function(slika,k)
{
  n1 = dim(slika)[1]
  n2 = dim(slika)[2]
  for(i in n1:1) # Idemo x-osom od kraja do pocetka
  {
    if(mean(slika[i,]!=k) > 0.001)
    {
      for(j in n2:1) # Odozdo trazimo prvu tacku koja se razlikuje od pozadine
      {
        if(slika[i,j]!=k)
        {
          tacka <- c(i,j)
          return(tacka)
        }
      }
    }
  }
}

# Tacka gore desno
T3 <- function(slika,k)
{
  n1 = dim(slika)[1]
  n2 = dim(slika)[2]
  for(j in 1:n2) # Idemo y-osom Odozgo na dole
```

```
  {
    if(mean(slika[,j]!=k) > 0.001)
    {
      for(i in n1:1) # Od desnog kraja x-ose trazimo prvu tacku koja se razlikuje od
pozadine
      {
        if(slika[i,j]!=k)
        {
          tacka <- c(i,j)
          return(tacka)
        }
      }
    }
  }
}

# Tacka gore levo
T4 <- function(slika,k)
{
  n1 = dim(slika)[1]
  n2 = dim(slika)[2]
  for(i in 1:n1) # Idemo sleva duz x-ose
  {
    if(mean(slika[i,]!=k) > 0.001)
    {
      for(j in 1:n2) # Odozgo trazimo prvu tacku koja se razlikuje od pozadine
      {
        if(slika[i,j]!=k)
        {
          tacka <- c(i,j)
          return(tacka)
        }
      }
    }
  }
}
```

Правимо фунцију *pravougaonici* која ће да проверава да ли је на слици тачно један црни или бели правоугаоник.

```
pravougaonik <- function(slika,i)
{
  slika1 <- rm.alpha(slika) # uklanjamo cetvrti spektar boje i dobijamo sliku u RGB
spektru
  slika <- grayscale(slika1) # pretvaramo sliku u crno belu
  slika <- slika[,,1,1]
  n1 <- dim(slika)[1]
  n2 <- dim(slika)[2]
  pozadina <- slika[1,1]

  A <- T1(slika,pozadina)
  B <- T2(slika,pozadina)
  C <- T3(slika,pozadina)
  D <- T4(slika,pozadina)


  koordinate <- cbind(A,B,C,D)
  koordinate <-as.matrix(koordinate)
  colnames(koordinate) <- c('A','B','C','D')
  rownames(koordinate) <- c('x','y')
```

```
  # Izdvajamo deo slike koji je ogranicen tackama A,B,C,D

  x1 <- min(A[1],B[1],C[1],D[1])
  x2 <- max(A[1],B[1],C[1],D[1])
  y1 <- min(A[2],B[2],C[2],D[2])
  y2 <- max(A[2],B[2],C[2],D[2])

  pravougaonik <- array(slika[x1:x2,y1:y2],c(x2-x1+1,y2-y1+1))
  pravougaonik <- renorm(as.cimg(pravougaonik))
  pravougaonik # izdvojili smo samo pravougaonik sa slike
 #plot(pravougaonik) # slika izdvojenog pravougaonika

  pravougaonik <- pravougaonik[,,1,1]
  n12 <- dim(pravougaonik)[1]
  n22 <- dim(pravougaonik)[2]

  # Kako bi proverili da li se na slici nalazi jedan ili vise pravougaonika,trazicem
o prelaze boja
  # Ako ima najvise dva prelaza, onda je na slici sigurno jedan pravougaonik
  # Ako ima vise od dva prelaza, onda na slici ima vise od jednog pravougaonika

  # Brojimo prelaze

  l <-1
  for(i in 1:n12)
  {
    l1 <- l
    l <- 1

    for(j in 2:n22)
    {
      if(pravougaonik[i,j]!=pravougaonik[i,j-1])
        l <- l+1
    }

    if(l1>l && l==1)
    {
      print("Na slici se nalazi vise od jednog pravougaonika.")
      return()
    }
  }


  # Ako je na slici jedan pravougaonik, treba da proverimo da li je crne, bele ili n
eke trece boje
  # Najlakse je naci centar pravougaonika i proveriti koje je boje

  # Koordinate centra pravougaonika racunamo pomocu formula
  s <- 2*(D[1]*(B[2]-C[2])+B[1]*(C[2]-D[2])+C[1]*(D[2]-B[2]))
  s1 <- (D[1]^2+D[2]^2)*(B[2]-C[2])+(B[1]^2+B[2]^2)*(C[2]-D[2])+(C[1]^2+C[2]^2)*(D[2
]-B[2])
  s2 <- (D[1]^2+D[2]^2)*(C[1]-B[1])+(B[1]^2+B[2]^2)*(D[1]-C[1])+(C[1]^2+C[2]^2)*(B[1
]-D[1])
  centar <- c(s1/s,s2/s)

  # Proveravamo koje je boje centar i ako je crne ili bele boje

  if(all(slika1[s1/s,s2/s,1,]==c(1,1,1)))
  {
    print("Na slici je jedan beli pravougaonik.Koordinate njegovih temena su:")
    return(koordinate)
```

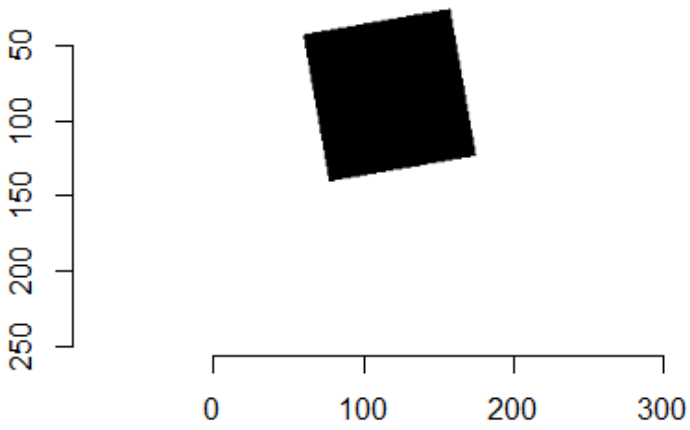
```
  }
  else if(all(slika1[s1/s,s2/s,1,]==c(0,0,0)))
  {
    print("Na slici je jedan crni pravougaonik.Koordinate njegovih temena su:")
    return(koordinate)
  }
  else print("Na slici nije pravougaonik potrebne boje.")

}
```

Применом претходне функције проверавамо за сваку слику из фолдера да ли се на њој налази тачно један црни или бели правоугаоник.

```
length(pravougaonici)
```

```
## [1] 9
```

```
# ima 9 slika
slika1 <- pravougaonici[[1]]
plot(slika1)
```

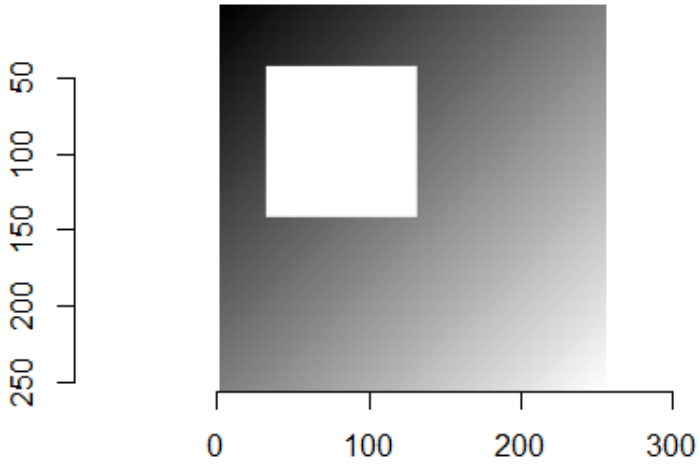



```
pravougaonik(slika1,1)
```

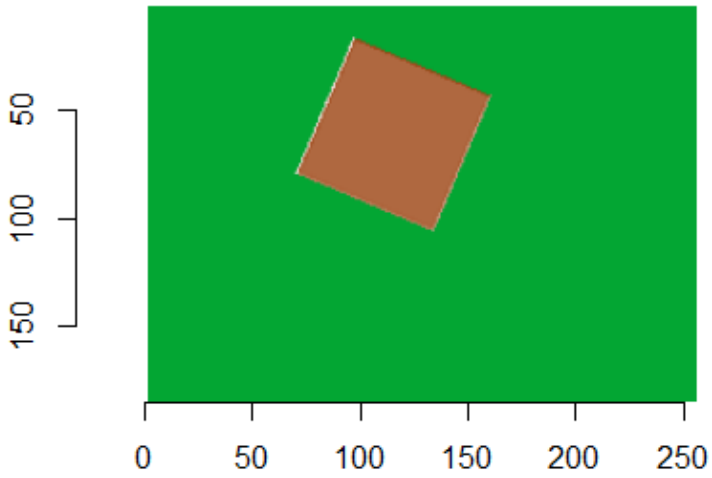
```
## [1] "Na slici je jedan crni pravougaonik.Koordinate njegovih temena su:"
```

```
##     A   B   C  D
## x  76 174 156 59
## y 139 122  24 42
```

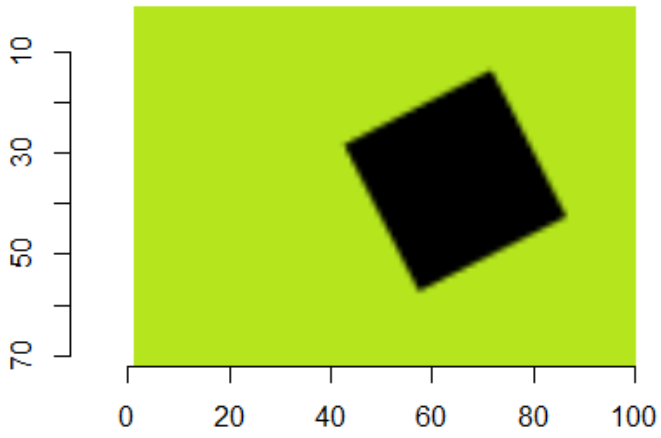
```
slika2 <- pravougaonici[[2]]
plot(slika2)
```



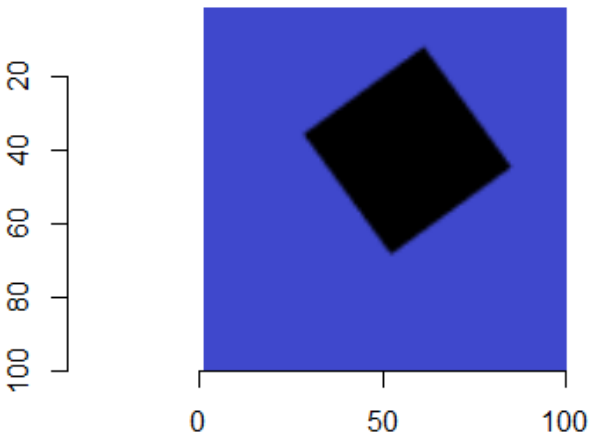

```
pravougaonik(slika2,2)

## [1] "Na slici je jedan beli pravougaonik.Koordinate njegovih temena su:"

##     A   B   C D
## x   1 256 256 1
## y 256 256   1 2

slika3 <- pravougaonici[[3]]
plot(slika3)
```



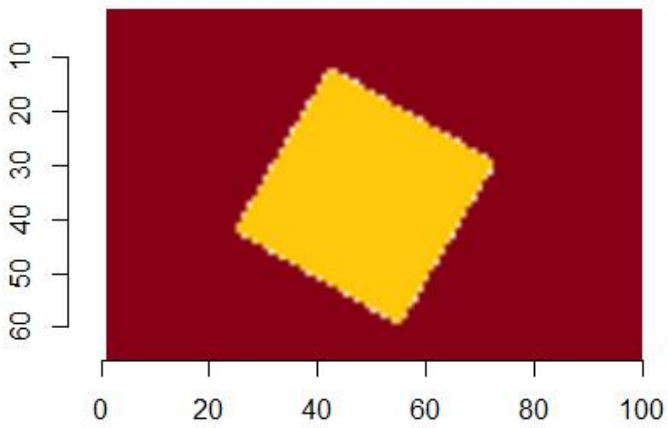
```
pravougaonik(slika3,3)

## [1] "Na slici nije pravougaonik potrebne boje."

slika4 <- pravougaonici[[4]]
plot(slika4)
```



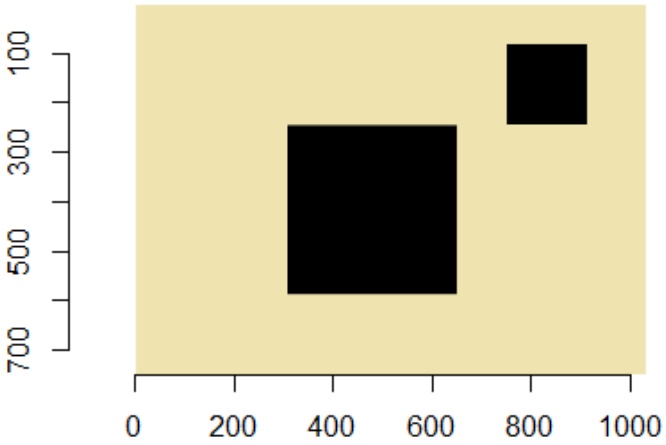

```
pravougaonik(slika4,4)

## [1] "Na slici je jedan crni pravougaonik.Koordinate njegovih temena su:"

##    A  B  C  D
## x 57 86 72 42
## y 58 43 13 28

slika5 <- pravougaonici[[5]]
plot(slika5)
```



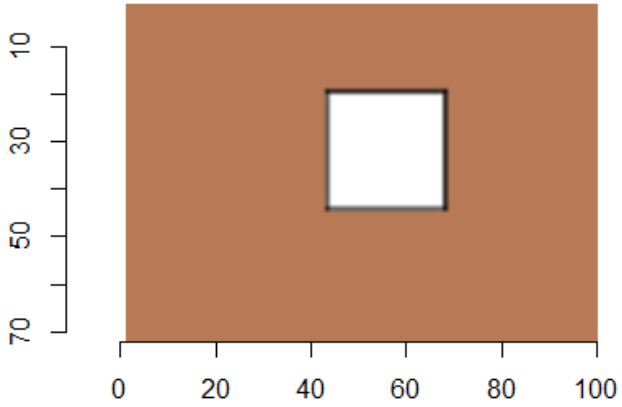
```
pravougaonik(slika5,5)

## [1] "Na slici je jedan crni pravougaonik.Koordinate njegovih temena su:"

##    A  B  C  D
## x 51 85 61 28
## y 68 44 11 35

slika6 <- pravougaonici[[6]]
plot(slika6)
```



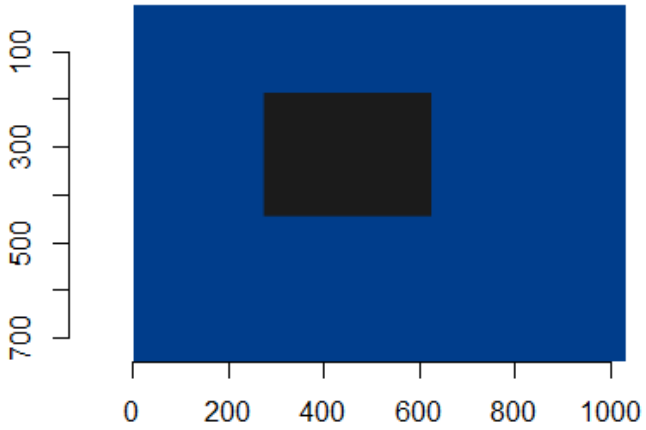

```
pravougaonik(slika6,6)

## [1] "Na slici nije pravougaonik potrebne boje."

slika7 <- pravougaonici[[7]]
plot(slika7)
```



```
pravougaonik(slika7,7)

## [1] "Na slici se nalazi vise od jednog pravougaonika."

## NULL

slika8 <- pravougaonici[[8]]
plot(slika8)
```




```
pravougaonik(slika8,8)

## [1] "Na slici je jedan beli pravougaonik.Koordinate njegovih temena su:"

##    A  B  C  D
## x 43 68 68 43
## y 44 44 19 19

slika9 <- pravougaonici[[9]]
plot(slika9)
```



```
pravougaonik(slika9,9)

## [1] "Na slici nije pravougaonik potrebne boje."
```

Сада ћемо учитати слике из подфолдера $krugovi$.

```
setwd("C:/Users/Korisnik/Desktop/seminarski/drugi_zadatak/krugovi")
files <- list.files(path=getwd(),all.files=T, full.names=F, no.. = T)
krugovi <- lapply(files, load.image)
```

На сличан начин правимо функцију $krug$.

```
krug <- function(slika)
{
  slika1 <- rm.alpha(slika) # uklanjamo cetvrti spektar boje i dobijamo sliku u RGB spektru
  slika <- grayscale(slika1) # pretvaramo sliku u crno belu
  slika <- slika[,,1,1]
  n1 <- dim(slika)[1]
  n2 <- dim(slika)[2]
  pozadina <- slika[1,1]

  A <- T1(slika,pozadina)
  B <- T2(slika,pozadina)
  C <- T3(slika,pozadina)
  D <- T4(slika,pozadina)

  x1 <- min(A[1],B[1],C[1],D[1])
  x2 <- max(A[1],B[1],C[1],D[1])
  y1 <- min(A[2],B[2],C[2],D[2])
  y2 <- max(A[2],B[2],C[2],D[2])

  krug <- array(slika[x1:x2,y1:y2],c(x2-x1+1,y2-y1+1))
  krug <- renorm(as.cimg(krug))
  krug
  # Na ovaj nacin smo izdvojili kvadrat, na polovini svake stranice nalazi se jedna od tacaka A,B,C,D
  #plot(krug) # slika izdvojenog kvadrata

  krug <- krug[,,1,1]
  n12 <- dim(krug)[1]
  n22 <- dim(krug)[2]

  l <- 0
  # Idemo duz x-ose, za fikciranu y koordinatu koja je bas sredina stranice kvadrata
  # Trazimo one tacke koje se razlikuju od (1,centar) u vise od 5% slucajeva

  for(i in 1:n12)
  {
    if(mean(krug[i,(y2-y1)/2]!=krug[1,(y2-y1)/2]) > 0.05)
      l <-l+1
  }
```

```
  # Sada idemo duz y-ose, za fikciranu x koordinatu koja je bas sredina stranice
kvadrata
  # Trazimo one tacke koje se razlikuju od (centar,n22) u vise od 5% slucajeva
  for(i in n22:1)
  {
    if(mean(krug[(x2-x1)/2,i]!=krug[(x2-x1)/2,n22]) > 0.05)
      l <-l+1
  }

  # Ako smo dobili da postoje tecke koje se razlikuju, ima smisla da gledamo da li
se na toj slici
  # nalazi vise krugova
  # Ako ima vise od 2 prelaza, imamo vise krugova

  k=1
  if (l>0)
  {
       for (i in 1:n12)
    {
      for (j in 2:n22)
      {
        if (krug[i,j]!=krug[i,j-1])
        k <-k+1
      }


       }
    if(k>3)
    {
      print("Na slici se nalazi vise od jednog kruga.")
      return()
    }
  }


  # Ako je na slici jedan krug, treba da proverimo da li je crne, bele ili neke
trece boje
  # Najlakse je naci centar kruga i proveriti koje je boje

  # Koordinate centra  racunamo pomocu formula
  s <- 2*(D[1]*(B[2]-C[2])+B[1]*(C[2]-D[2])+C[1]*(D[2]-B[2]))
  s1 <- (D[1]^2+D[2]^2)*(B[2]-C[2])+(B[1]^2+B[2]^2)*(C[2]-
D[2])+(C[1]^2+C[2]^2)*(D[2]-B[2])
  s2 <- (D[1]^2+D[2]^2)*(C[1]-B[1])+(B[1]^2+B[2]^2)*(D[1]-
C[1])+(C[1]^2+C[2]^2)*(B[1]-D[1])
  centar <- c(s1/s,s2/s)

  # Poluprecnik kruga racunamo pomocu formule
  R <- sqrt((B[1]-D[1])^2+(B[2]-D[2])^2)/2

  # Proveravamo koje je boje centar i ako je crne ili bele boje
```

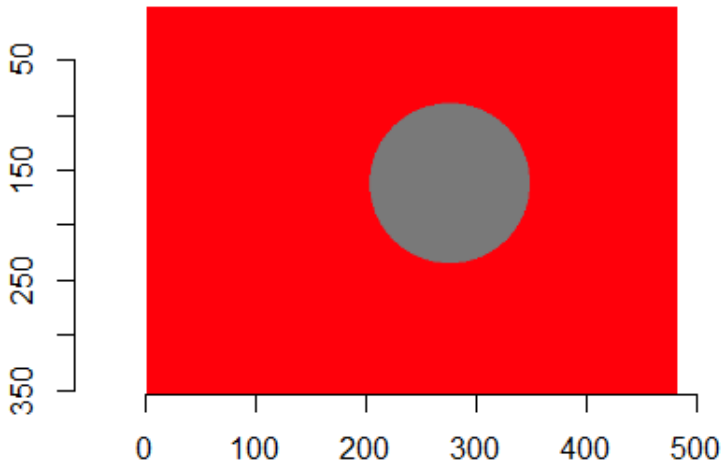
```
  rezultat <-list(koordinate_centra=centar,poluprecnik=R)

  if(all(slika1[s1/s,s2/s,1,]==c(1,1,1)))
  {
    print("Na slici je jedan beli krug.Koordinate centra i poluprecnik su redom:")
    return(rezultat)

  }
  else if(all(slika1[s1/s,s2/s,1,]==c(0,0,0)))
  {
    print("Na slici je jedan crni krug.Koordinate centra i poluprecnik su :")
    return(rezultat)
  }
  else print("Na slici nije krug potrebne boje.")

}
```

Применом претходне функције проверавамо за сваку слику из фолдера да ли се на њој налази тачно један црни или бели круг.

```
length(krugovi)
```

```
## [1] 4
```

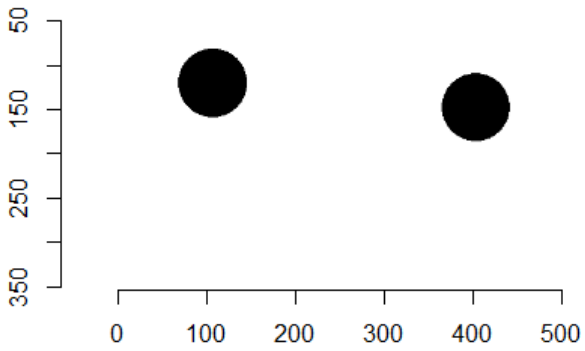
```
# Imamo 4 slike

slika1 <- krugovi[[1]]
plot(slika1)
```




```
krug(slika1)
```

```
## [1] "Na slici nije krug potrebne boje."
```

```
slika2 <- krugovi[[2]]
plot(slika2)
```

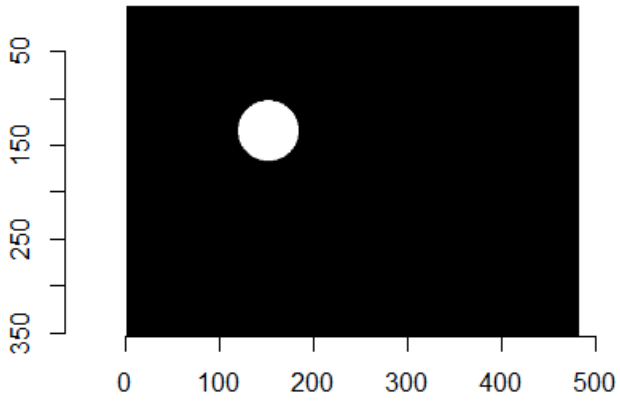



```
krug(slika2)
```

```
## [1] "Na slici se nalazi vise od jednog kruga."
```

```
## NULL
```

```
slika3 <- krugovi[[3]]
plot(slika3)
```

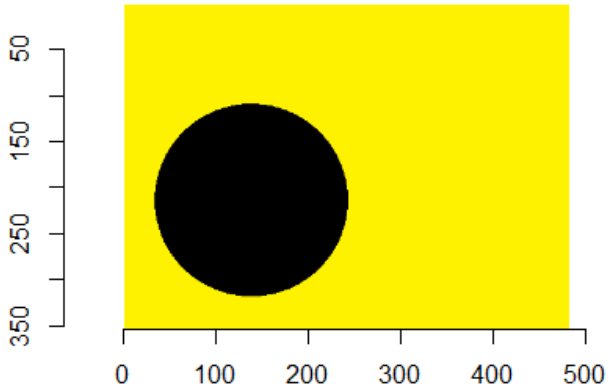



```
krug(slika3)
```

```
## [1] "Na slici je jedan beli krug.Koordinate centra i poluprecnik su redom:"
```

```
## $koordinate_centra
## [1] 151.0595 133.0766
##
## $poluprecnik
## [1] 32.31486
```

```
slika4 <- krugovi[[4]]
plot(slika4)
```




```
krug(slika4)
```

```
## [1] "Na slici je jedan crni krug.Koordinate centra i poluprecnik su :"
```

```
## $koordinate_centra
## [1] 137.5821 212.5967
##
## $poluprecnik
## [1] 104.9309
```

Трећи задатак

Написати функцију која за прослеђене две слике са ситним променама налази:

(а) На којим местима се те две слике разликују.

(б) Приказује обе слике и заокружује места где се разликују црвеним елипсоидима.

Пакет "$plotrix$"садржи велики број функција помоћу којих можемо да означавамо, бојимо итд. Нама ће требати функција "$draw.ellipse$" помоћу које чемо да закружујемо места где се слике разликују.

За почетак ћемо инсталирати и учитати тај пакет.

```
#install.packages("plotrix")
library(plotrix)
library(imager)
```

```
## Loading required package: magrittr
```

```
##
## Attaching package: 'imager'
```

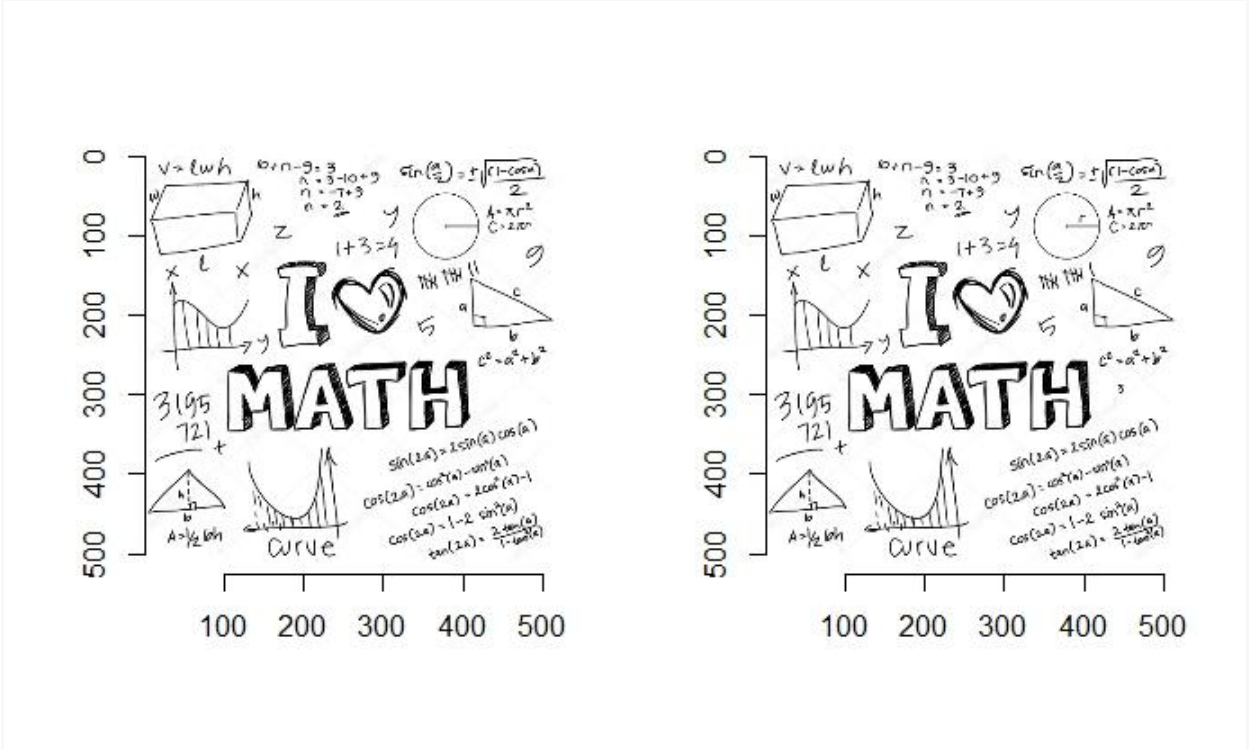
```
## The following object is masked from 'package:magrittr':
##
##     add

## The following objects are masked from 'package:stats':
##
##     convolve, spectrum

## The following object is masked from 'package:graphics':
##
##     frame

## The following object is masked from 'package:base':
##
##     save.image
```

Сада ћемо учитати све слике из фолдера $treci\_zadatak$ како би могли даље да радимо са њима.

```
setwd("C:/Users/Korisnik/Desktop/seminarski/treci_zadatak")
files <- list.files(path=getwd(),all.files=T, full.names=F, no.. = T)
slike <- lapply(files, load.image)
```

Нацртаћемо те две слике.

```
slika1 <- slike[[1]]
slika2 <- slike[[2]]
par(mfrow=c(1,2))
plot(slika1)
plot(slika2)
```



Морамо да проверимо да ли су слике истих димензија, иначе нећемо моћи да поредимо слике пиксел по пиксел.

```
d1 <- dim(slika1)
d2 <- dim(slika2)
```

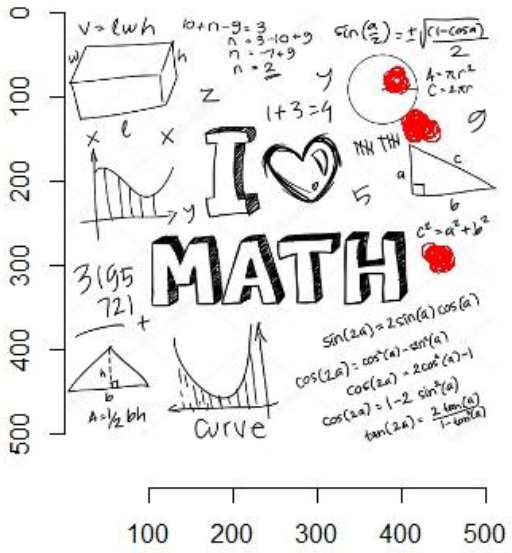
```
if ((d1[1]!=d2[1])||(d1[2]!=d2[2]))
  print("Slike nisu iste dimenzije.")
```

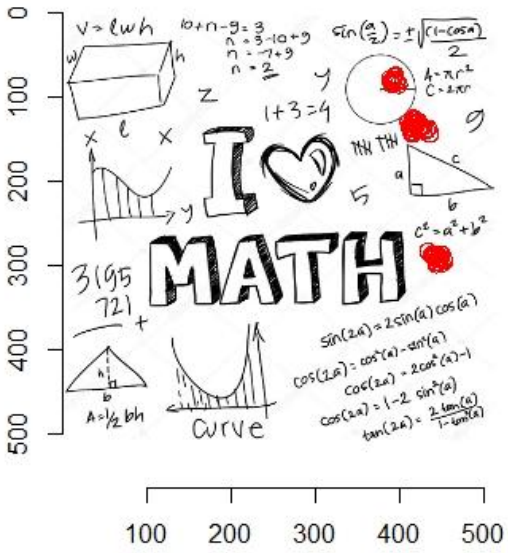
Правимо функцију "*razlike*" помоћу које ћемо пронаћи места где се слике разликују. Идеја је да пиксел по пиксел поредимо слике, и бележимо оне пикселе где постоје разлике, а затим те пикселе заокружимо.

```
razlike <- function(slika1,slika2)
{
  # Uzimamo dva prazna vektora u koja cemo smestati piksele na x odnsono y osi
  # gde se slike razlikuju

  X <- c()
  Y <- c()

  slika1 <- rm.alpha(slika1) # dobijamo da je slika samo u RGB spektru
  slika2 <- rm.alpha(slika2)
  # Proveravamo da li su iste, zato moramo da koristimo "renorm()"
  # Primenom funkcije "renorm()" trebalo bi da e dobije normalizovani oblik
  # svake slike i svi objekti koji se drugacije zapisuju u R-u, ali izgledaju
  # isto bi trebalo posle primene te funkcije da postanu (skoro) iste

  slika <- renorm(slika1)-renorm(slika2)
  n1 <- dim(slika)[1]
  n2 <- dim(slika)[2]

  # prolazimo kroz sve piksele
  for (i in 1:n1)
  {
    for (j in 1:n2)
    {
      if(!(all(slika[i,j,1,]==c(0,0,0))))  # ako nisu nule, onda ima razlike
      {
        X <- c(X,i) # dodajemo taj piksel gde postoji razlika
        Y <- c(Y,j)
      }

    }
  }

  plot(slika1)
  for(i in seq(1,length(X),length(X)/50))
    draw.ellipse(X[i],Y[i],10,10,border = "red")
 plot(slika2)
 for(i in seq(1,length(X),length(X)/50))
   draw.ellipse(X[i],Y[i],10,10,border = "red")

 razlika <-cbind(X,Y)
 return(razlika)
}
```

Примењујемо написану функције на учитане слике.

```
razlike(slika1,slika2)
```






```
##           X   Y
##   [1,] 389  78
##   [2,] 389  79
##   [3,] 389  81
##   [4,] 390  77
##   [5,] 390  78
##   [6,] 390  79
##   [7,] 390  80
##   [8,] 390  81
##   [9,] 390  82
##         .
##         .
##         .
## [418,] 454 293
## [419,] 454 295
## [420,] 454 296
## [421,] 454 297
## [422,] 454 298
```

Миноловац

Четврти задатак

За слике табле Миноловца, које се налазе у фолдеру $cetvrti\_zadatak\_obucavanje$ направити моделе $(QDA, LDA$ и Мултиномни Логистички $)$ чији је задатак да се одреди који се број налази у сваком од потпоља. Тестирати колико су добри тражени модели на сликама које су у фолдеру $cetvrti\_zadatak\_kontrolni$. Детаљно описати све искоришћене функције и предикторе, прокоментарисати резултате, одредити који је модел најбољи.

На почетку ћемо инсталирати и учитати пакете који ће нам бити потребни за рад.

```
library(imager)

## Loading required package: magrittr

##
## Attaching package: 'imager'

## The following object is masked from 'package:magrittr':
##
##     add

## The following objects are masked from 'package:stats':
##
##     convolve, spectrum

## The following object is masked from 'package:graphics':
##
##     frame

## The following object is masked from 'package:base':
##
##     save.image

# Odmah cemo i instalirati i ukljuciti pakete "MASS" i "nnet"
library(MASS) # U MASS-u su "lda()" i "qda()" koje ce nam trebati
library(nnet) # U "nnet" se nalazi "multinom()" koja ce nam trebati
```

Учитавамо све слике из фолдера за обучавање у листу.

```
setwd("C:/Users/Korisnik/Desktop/seminarski/cetvrti_zadatak_obucavanje")
files <- list.files(path=getwd(),all.files=T, full.names=F, no.. = T)
# Primenjujemo funkciju load.image da ucitamo sve slike iz foldera
slike <- lapply(files, load.image)
```

Правимо функције помоћу којих ћемо моћи да одредимо границе поља.

```
granice <- function(matrica, epsilon, prob, by.row=T)
{
  if(by.row)
  {
    Mean <- rowMeans #za granice po vrstama, koristimo f-ju rowMeans
  }
  else
```

```
  {
    Mean <- colMeans #za granice po kolonama colMeans
  }

  vec <- Mean(matrica < epsilon) >= prob
  return(which(vec!=F))
}

sredjivanje <- function(vektor)
{
  novi_vektor <- c()
  for(i in 1:(length(vektor)-1))
  {
    if(vektor[i+1]-vektor[i]>5)
    {
      novi_vektor <- c(novi_vektor, vektor[i])
    }
  }
  for(j in length(vektor):2)
  {
    if(vektor[j]-vektor[j-1]>5)
    {
      novi_vektor <- c(novi_vektor, vektor[j])
    }
  }
  return(sort(novi_vektor))
}
```

Желимо сада да уведемо неки предиктор и нека прва идеја би била да то буде проценат у колико пиксела је највећа вредност баш плава (у одговарајућим матрицама).

```
prediktor <- function(slika, vrste, kolone)
{
  prosek_plave <- c()

  l <- 1
  for(j in 1:9)
  {
    for(i in 1:9)
    {
      #izdvojimo polje
      s <- array(slika[vrste[2*j-1]:vrste[2*j], kolone[2*i-1]:kolone[2*i], 1, 1:3],
                 c(vrste[2*j]-vrste[2*j-1]+1,kolone[2*i]-kolone[2*i-1]+1, 1, 3))
      nova <- cimg(s)
      crvena <- as.vector(nova[ , , 1, 1])
      zelena <- as.vector(nova[ , , 1, 2])
      plava <- as.vector(nova[ , , 1, 3])

      brojac_plave <- mean(plava/(crvena+zelena+plava),na.rm = TRUE)
      prosek_plave[l] <-brojac_plave

      l=l+1
    }
  }

  return(prosek_plave)
}
```

Други предиктор би била дисперзија поља у црно-брлом спектру.

```
prediktor2 <- function(slika, vrste, kolone)
{
  X <- c()
  l <- 1
  # r sirina odsecanja
  r <- ceiling((vrste[2]-vrste[1])/16)
  for(j in 1:9)
  {
    for(i in 1:9)
    {
      s <-array(slika[(vrste[2*j-1]+r):(vrste[2*j]-r), (kolone[2*i-1]+r):(kolone[2*i
]-r), 1, 1:3],
                c(vrste[2*j]-vrste[2*j-1]-2*r+1,kolone[2*i]-kolone[2*i-1]-2*r+1, 1, 3)
)
      X[l] <- var(as.vector(grayscale(cimg(s))[,,,1]))
      l <-l+1
    }
  }
  return(X)
}
```

Сада правимо вектор са вредностима за категоријску променљиву Y, што морамо ручно. Уз унапред обављен договор о додељивању бројева у зависности од тога шта се налази у пољу:

- Затворено поље : 0
- Број 1 : 10
- Број 2 : 20
- Број 3 : 30
- Број 4 : 40
- Број 5 : 50
- Број 6 : 60
- Број 7 : 70
- Број 8 : 80
- Мина : -100
- Празно поље : 100

```
Y1 <-c(0,0,0,0,0,0,0,0,0,
       0,0,0,20,10,10,-100,10,0,
       0,0,30,-100,10,10,10,10,0,
       20,30,-100,20,10,100,10,10,0,
       -100,20,10,10,100,100,10,-100,0,
       10,10,100,100,10,10,20,10,0,
       100,10,10,10,10,-100,20,10,0,
       100,10,-100,0,0,0,0,0,0,
       100,10,0,0,0,0,0,0,0)

Y2 <-c(0,0,0,10,100,10,10,10,100,
       0,20,0,20,10,10,-100,10,100,
       0,0,30,-100,10,10,10,10,100,
       20,30,-100,20,10,100,10,10,10,
       -100,20,10,10,100,100,10,-100,10,
       10,10,100,100,10,10,20,10,10,
       100,10,10,10,10,-100,20,10,100,
       100,10,-100,10,10,20,0,10,100,
```

```
       100,10,10,10,100,10,0,10,100)

Y3 <-c(100,100,100,100,100,100,10,0,0,
       10,10,100,100,100,10,20,0,0,
       0,10,100,100,100,20,0,0,0,
       0,10,100,100,100,20,0,30,10,
       0,10,10,10,100,10,10,10,100,
       0,0,0,10,100,100,100,100,100,
       0,0,0,10,100,10,10,10,100,
       0,0,0,10,10,10,0,20,10,
       0,0,0,0,0,0,0,0,0)

Y4 <-c(100,100,100,100,100,100,10,10,10,
       10,10,100,100,100,10,20,-100,20,
       -100,10,100,100,100,20,-100,40,-100,
       10,10,100,100,100,20,-100,30,10,
       100,10,10,10,100,10,10,10,100,
       100,10,-100,10,100,100,100,100,100,
       100,10,10,10,100,10,10,10,100,
       10,10,20,10,10,10,-100,20,10,
       10,-100,20,-100,10,10,10,20,-100)

Y5 <-c(100,100,100,100,100,100,100,10,0,
       10,10,100,100,100,100,100,10,0,
       0,10,100,100,10,20,30,30,0,
       20,20,100,100,10,-100,-100,-100,10,
       0,10,100,100,10,20,30,20,10,
       0,10,10,10,10,100,100,100,100,
       10,10,10,0,20,10,10,100,100,
       0,0,20,10,20,0,10,100,100,
       0,0,10,100,10,0,10,100,100)

Y6 <-c(100,100,100,100,100,100,100,100,100,
       100,100,100,100,100,10,10,10,100,
       100,100,100,100,100,20,0,30,10,
       10,10,100,100,100,20,0,0,0,
       0,10,100,10,20,30,30,30,0,
       10,10,100,10,0,0,10,10,0,
       100,10,10,20,20,20,10,10,10,
       10,20,0,10,100,100,100,10,0,
       0,20,10,10,100,100,100,10,0)

Y7 <-c(100,100,100,100,100,100,10,20,0,
       20,20,20,10,10,100,10,0,20,
       -100,-100,0,0,30,10,10,10,10,
       20,20,30,0,0,10,100,100,100,
       100,100,10,20,20,10,100,100,100,
       100,10,10,10,100,100,100,100,100,
       100,10,0,10,100,100,100,100,100,
       100,20,20,30,10,10,100,100,100,
       100,10,0,20,0,10,100,100,100)


Y8 <-c(100,10,10,10,100,100,100,100,100,
       100,10,0,10,100,100,10,20,20,
       10,20,10,10,10,20,30,0,0,
       0,10,100,100,10,0,0,40,30,
       10,10,10,10,20,20,20,20,0,
       100,100,10,0,20,10,100,10,10,
       100,100,10,20,0,10,100,100,100,
       100,100,100,10,10,10,10,10,10,
```

```
       100,100,100,100,100,100,10,0,0)


Y9 <-c(0,0,0,0,0,0,0,0,0,
        0,30,10,30,0,0,0,0,0,
        0,20,100,10,20,20,0,0,0,
        10,10,100,100,100,10,0,0,0,
        100,100,100,10,20,40,0,0,0,
        10,20,10,20,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,40,0,
        0,0,0,0,0,0,0,0,0)
```

Спојићемо их све у један вектор, који ће представљати зависну променљиву.

```
Y <- c(Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9)
```

Вектори у које ћемо сместити вредности предиктора за сваку таблу миноловца:

```
X1 <- c()
X2 <- c()
```

Сада рачунамо границе за сваку слику из скупа за обучавање, израчунавамо вредност предиктора и све смештамо у векторе.

```
n <- length(slike)
for(i in 1:n) {
  slika <- rm.alpha(slike[[i]]) #da slika bude samo u RGB spektru

  q11 <-slika [ , , 1, 1]
  q12 <-slika [ , , 1, 2]
  q13 <-slika [ , , 1, 3]
  M <- pmax(q11, q12, q13) #pmax dodeljuje maksimalnu vrednost po elementima matrica
/vektora

  m_vrsta <- granice(M,0.2, 0.3)
  m_kolona <-granice(M,0.2, 0.3,FALSE)

  v <-sredjivanje(m_vrsta)
  k <-sredjivanje(m_kolona)

  X1 <- c(X1, prediktor(slika, v, k)) #dodajemo vrednost prediktora i prediktora2 za
ovu sliku u vektor
  X2 <- c(X2, prediktor2(slika, v, k))
}
```

Учитавамо сада слике из скупа за тестирање модела.

```
setwd("C:/Users/Korisnik/Desktop/seminarski/cetvrti_zadatak_kontrolni")
files <- list.files(path=getwd(),all.files=T, full.names=F, no.. = T)
slike2 <- lapply(files, load.image)
```

Ово ће бити предиктори на основу чије вредности предвиђамо:

```
X12 = c()
X22 = c()
```

Рачунамо као и мало пре:

```
for(i in 1:length(slike2)) {
  slika <- rm.alpha(slike2[[i]]) #da slika bude samo u RGB spektru
```

```
  q11 <-slika [ , , 1, 1]
  q12 <-slika [ , , 1, 2]
  q13 <-slika [ , , 1, 3]
  M <- pmax(q11, q12, q13) #pmax dodeljuje maksimalnu vrednost po elementima matrica
/vektora

  m_vrsta2 <- granice(M,0.2, 0.3)
  m_kolona2<-granice(M,0.2, 0.3,FALSE)

  v2 <-sredjivanje(m_vrsta2)
  k2 <-sredjivanje(m_kolona2)

  X12 <- c(X12, prediktor(slika, v2, k2)) #dodajemo vrednost prediktora i prediktora
2 za ovu sliku u vektor
  X22 <- c(X22, prediktor2(slika, v2, k2))
}
```

Правимо векторе са вредностима за категориску променљиву Y за тестирање модела.

```
Y_kon1 <- c(10,10,20,-100,10,100,100,100,100,
            20,-100,30,10,10,100,100,100,100,
            20,-100,20,100,100,10,10,10,100,
            0,20,10,100,100,10,-100,20,10,
            -100,10,100,10,10,20,20,0,0,
            10,10,100,10,-100,10,10,0,0,
            100,100,100,10,10,10,10,20,0,
            100,100,100,100,10,10,20,0,0,
            100,100,100,100,10,0,20,0,0)

Y_kon2 <- c(0,0,10,0,0,-100,10,100,100,
            10,10,10,10,20,20,10,10,10,
            100,100,100,100,100,100,100,10,0,
            100,100,100,100,100,100,100,10,10,
            100,100,100,100,100,10,10,20,10,
            100,100,100,100,100,20,-100,30,-100,
            100,100,100,10,10,40,-100,40,10,
            100,100,100,20,-100,40,-100,20,100,
            100,100,100,20,-100,30,10,10,100)
```

Спојићемо их у један вектор који ће нам служити да проверимо модел.

```
Y_kontolni <- c(Y_kon1, Y_kon2)
```

Сада желимо мало озбиљније то да тестирамо и проверимо колико добро модел детектује елементе скупа за обучавање помоћу ова два предиктора и методама $"multinom", "lda", "qda"$.

Мултиномни логистички

```
model.multinom <- multinom(Y ~ X1+X2)

## # weights:  28 (18 variable)
## initial  value 1418.568499
## iter  10 value 1069.800670
## iter  20 value 1043.905473
## iter  30 value 1041.717592
## iter  40 value 1041.158294
## iter  50 value 1041.019741
```

```
## iter  60 value 1040.805160
## iter  70 value 1040.540560
## iter  80 value 1040.492945
## iter  90 value 1040.481922
## iter 100 value 1040.474825
## final  value 1040.474825
## stopped after 100 iterations
```

```
summary(model.multinom)
```

```
## Call:
## multinom(formula = Y ~ X1 + X2)
##
## Coefficients:
##     (Intercept)         X1          X2
## 0     3.3123048  -2.037840 -31.8982831
## 10    5.1742110  -6.944274  -8.4774008
## 20    6.5659421 -14.007287  -3.3584776
## 30    3.3528519  -9.166508   0.7866756
## 40   -0.3465304  -5.083470  15.5619070
## 100   7.9529353 -11.780710 -54.8295287
##
## Std. Errors:
##     (Intercept)       X1       X2
## 0      1.582881 3.128586 13.21599
## 10     1.562475 3.117945 12.57014
## 20     1.821511 3.841347 13.95349
## 30     2.340823 4.968836 17.92522
## 40     4.120644 8.767146 30.54566
## 100    1.563661 3.136798 12.96167
##
## Residual Deviance: 2080.95
## AIC: 2116.95
```

Предвиђамо вредности зависне променљиве за вредност предиктора слика из контролног скупа користећи модел направљен на основу скупа за обучавање.

```
model.mulpred <- predict(model.multinom, newdata = data.frame(X1=X12, X2=X22))
matrix(model.mulpred, ncol=9)
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
##  [1,] "10"  "10"  "10"  "100" "100" "10"  "100" "100" "100"
##  [2,] "10"  "10"  "10"  "100" "100" "10"  "100" "100" "100"
##  [3,] "10"  "10"  "100" "100" "100" "10"  "100" "100" "100"
##  [4,] "10"  "100" "10"  "10"  "100" "10"  "100" "100" "10"
##  [5,] "10"  "100" "10"  "10"  "10"  "10"  "100" "100" "10"
##  [6,] "100" "10"  "10"  "10"  "0"   "10"  "100" "10"  "10"
##  [7,] "100" "10"  "10"  "10"  "10"  "10"  "100" "10"  "10"
##  [8,] "100" "10"  "0"   "10"  "0"   "10"  "10"  "10"  "10"
##  [9,] "100" "100" "0"   "0"   "0"   "10"  "10"  "10"  "100"
## [10,] "10"  "0"   "10"  "100" "0"   "100" "100" "100" "100"
## [11,] "10"  "10"  "10"  "100" "0"   "100" "100" "100" "100"
## [12,] "10"  "10"  "100" "100" "10"  "100" "100" "100" "100"
## [13,] "10"  "100" "10"  "100" "0"   "100" "100" "10"  "10"
## [14,] "10"  "100" "10"  "10"  "0"   "100" "100" "10"  "10"
## [15,] "100" "10"  "10"  "10"  "10"  "100" "10"  "10"  "10"
## [16,] "100" "10"  "10"  "10"  "10"  "100" "10"  "10"  "10"
## [17,] "100" "10"  "0"   "0"   "100" "10"  "10"  "10"  "10"
## [18,] "100" "10"  "0"   "0"   "100" "0"   "10"  "10"  "100"
```

```
table(model.mulpred, Y_kontolni)
```

```
##              Y_kontolni
## model.mulpred -100  0 10 20 30 40 100
##          -100    0  0  0  0  0  0   0
##          0       0 16  0  0  0  0   0
##          10     13  0 46 18  3  3   0
##          20      0  0  0  0  0  0   0
##          30      0  0  0  0  0  0   0
##          40      0  0  0  0  0  0   0
##          100     0  0  0  0  0  0  63

mean(model.mulpred == Y_kontolni)

## [1] 0.7716049
```

Видимо да модел погађа око 77% вредности.

ЛДА

```
model.lda <- lda(Y~X1+X2)
summary(model.lda)

##         Length Class  Mode
## prior    7     -none- numeric
## counts   7     -none- numeric
## means   14     -none- numeric
## scaling  4     -none- numeric
## lev      7     -none- character
## svd      2     -none- numeric
## N        1     -none- numeric
## call     2     -none- call
## terms    3     terms  call
## xlevels  0     -none- list

model.ldapred <- predict(model.lda, newdata = data.frame(X1=X12, X2=X22))
table(model.ldapred$class, Y_kontolni)

##       Y_kontolni
##        -100  0 10 20 30 40 100
##   -100    0  0  0  0  0  0   0
##   0       0 16  0  0  0  0   0
##   10     13  0 46 18  3  3   0
##   20      0  0  0  0  0  0   0
##   30      0  0  0  0  0  0   0
##   40      0  0  0  0  0  0   0
##   100     0  0  0  0  0  0  63

mean(model.ldapred$class == Y_kontolni)

## [1] 0.7716049
```

Видимо да и овај модел погађа око 77% вредности.

КДА

```
model.qda <- qda(Y~X1+X2)
summary(model.qda)

##         Length Class  Mode
## prior    7     -none- numeric
## counts   7     -none- numeric
## means   14     -none- numeric
```

```
## scaling 28     -none- numeric
## ldet     7     -none- numeric
## lev      7     -none- character
## N        1     -none- numeric
## call     2     -none- call
## terms    3     terms  call
## xlevels  0     -none- list
```

```
model.qdapred <- predict(model.qda, newdata = data.frame(X1=X12, X2=X22))
table(model.qdapred$class, Y_kontolni)
```

```
##       Y_kontolni
##        -100  0 10 20 30 40 100
##   -100    0  0  0  0  0  0   0
##   0       0 16  0  0  0  0   0
##   10     13  0 46 18  3  0   0
##   20      0  0  0  0  0  3   0
##   30      0  0  0  0  0  0   0
##   40      0  0  0  0  0  0   0
##   100     0  0  0  0  0  0  63
```

```
mean(model.qdapred$class == Y_kontolni)
```

```
## [1] 0.7716049
```

Видимо да и овај модел погађа око 77% вредности.

Пети задатак

Самостално изучити и испробати неку методу класификације коју нисмо детаљно обрађивали на часовима. Детаљно описати све искоришћене функције и предикторе, прокоментарисати резултате, одредити који је модел најбољи.

Стабла одлучивања

Основна идеја класификационих стабала

Подела простора атрибута којима су објекти описани у више разичитих и међусобно непреклопљених региона $R_1, R_2, \dots, R_n$.

Простор атрибута је p –димензионални простор кога чине могуће вредности p атрибута $(x_1, x_2, \dots, x_p)$ којима су дати објекти описани.

За нови објекат X, одређује се припадност једном од региона $R_1, R_2, \dots, R_n$ на основу вредности атрибута $(x_1, x_2, \dots, x_p)$ којима је X описан.

Класа новог објекта ће бити она класа која доминира у региону R_j у који је X сврстан.

Подела простора атрибута

Подела простора атрибута на регионе R_j је итеративни процес који се састоји од:

- избора атрибута x_i који ће бити основа за поделу
- избора вредности атрибута x_i која ће послужити као “гранична” вредност

Одређивање класе инстанци у регионима $R_1, R_2, \dots, R_k$

Користећи принцип већинске класе, сваком региону R_j придружити класу којој припада већина инстанци из скупа за обучавање која је сврстана у регион R_j.

Како и где извршити поделу?

Циљ је пронаћи регоне $R_1, R_2, \dots, R_n$ тако да се минимизује грешка при класификацији- $Classification\ Error\ Rate\ (CER)$.

CER представља пропорцију инстанци из скупа за обучавање у датом региону које не припадају доминантној класи тог регона

$$CER = 1 - \max_k \hat{p}_{ik}$$

$\hat{p}_{ik}$ је пропорција (тренинг) инстанци у региону i које припадају класи k.

Приступ који се примењује да би се идентификовали региони који минимизују грешку при класификацији заснива се на рекурзивној,бинарној подели простора атрибута.

Основне карактеристике овог приступа су:

- $Top - down$ приступ
  Креће од врха стабла, где све (тренинг) инстанце припадају једној (заједничкој) регији, а затим сукцесивно дели простор атрибута на регионе
- $Greedy$ приступ
  При сваком кораку, најбоља подела се одређује на основу стања у том кораку, односно, не узима се у обзир шта ће бити у наредним корацима, тј. која би то подела могла довести до бољих резултата у неком наредном кораку

Алгоритам разматра сваки атрибут x_j ,$j = 1, \dots, p$ и сваку тачку поделе s_j за тај атрибут, и бира ону комбинацију која ће поделити простор атрибута у два региона $\{X|x_j > s_j\}$ и $\{X|x_j < s_j\}$ тако да се минимизује грешка класификације.

Осим грешке при класификацији $(Classification\ Error\ Rate)$, као критеријуми за поделу простора атрибута, често се користе и:

- Gini index
  Дефинише се на следећи начин:
  $$G = \sum_{k=1}^{K} \hat{p}_{ik}(1 - \hat{p}_{ik})$$
  $\hat{p}_{ik}$ представља пропорцију тренинг инстанци у регији m које припадају класи k

  Често се описује као мера “чистоће” чвора, где су “чисти” чворови они у којима висок проценат инстанци припада истој класи.
  Мала вредност за Гини индекс указује на “чисте” чворове
- Cross-entropy
  Дефинише се на следећи начин:
  $$D = -\sum_{k=1}^{K} \hat{p}_{ik} \log \hat{p}_{ik}$$

Као и Гини индекс, cross-entropy представља меру “чистоће” чвора (што је вредност мања, то је чвор “чистији”)

Орезивање стабла

Велика класификациона стабла, тј. стабла са великим бројем терминалних чворова (листова), имају тенденцију $over - fitting$ –а (тј. превеликог уклапања са тренинг подацима).

Овај проблем се може решити ‘орезивањем’ стабла, односно одсецањем неких терминалних чворова.

Како ћемо знати на који начин и у којој мери треба да ‘орежемо’ стабло?

Препорука је применом крос валидације утврдити које подстабло даје најмању грешку при класификацији.

Предности и недостаци стабала одлучивања

Предности:

- Могу се графички представити и једноставно интерпретирати
- Могу се применити како на класификационе, тако и регресивне проблеме
- Могу се применити и у случају да атрибути имају недостајуће вредности

Недостаци:

- Дају слабије резултате (мање тачне предикције) него други приступи вођеног машинског учења

Сада ћемо испробати ову методу класификације.

У програмском језику R за стабла одлучивања се користи пакет "$tree$" .У наставку ћемо користити стабла одлучивања на скупу података у $ISLR$ пакету, па је за почетак потребно исталирати и учитати те пакете.

```
# install.packages("tree")
# install.packages("ISLR")

library(tree)
library(ISLR)
```

Прво ћемо користити стабла одлучивања за анализу података базе $Carseats$ из $ISLR$ пакета.

Фокусираћемо се на класификацију $Sales$ различитих дечијих ауто седишта. Овде је променљива $Sales$ континуирана. Као резултат тога, прво ћемо $Sales$ представити бинарно и креирати категоријску променљиву која одређује која су седишта аутомобила имала $High$ број продаје.

Детаље о бази можемо добити позивањем $?Carseats$.

За приказ уопштених података о бази користићемо функцију $summary$:

```
summary(Carseats)
```

```
##      Sales          CompPrice       Income        Advertising
##  Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000
##  1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000
##  Median : 7.490   Median :125   Median : 69.00   Median : 5.000
##  Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635
##  3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000
##  Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000
##    Population        Price        ShelveLoc        Age
##  Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00
##  1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75
##  Median :272.0   Median :117.0   Medium:219   Median :54.50
##  Mean   :264.8   Mean   :115.8                Mean   :53.32
##  3rd Qu.:398.5   3rd Qu.:131.0                3rd Qu.:66.00
##  Max.   :509.0   Max.   :191.0                Max.   :80.00
##    Education    Urban       US
##  Min.   :10.0   No :118   No :142
##  1st Qu.:12.0   Yes:282   Yes:258
##  Median :14.0
##  Mean   :13.9
##  3rd Qu.:16.0
##  Max.   :18.0
```

```
attach(Carseats)
```

```
hist(Sales)
```

Правимо бинарну променљиву која одређује да ли је продаја седишта била велика или не, тј. да ли је $Sales$ била $High$ (> 8),.

```
High <- ifelse(Sales<=8,"No","Yes")
```

Добијену променљиву спајамо са базом података.

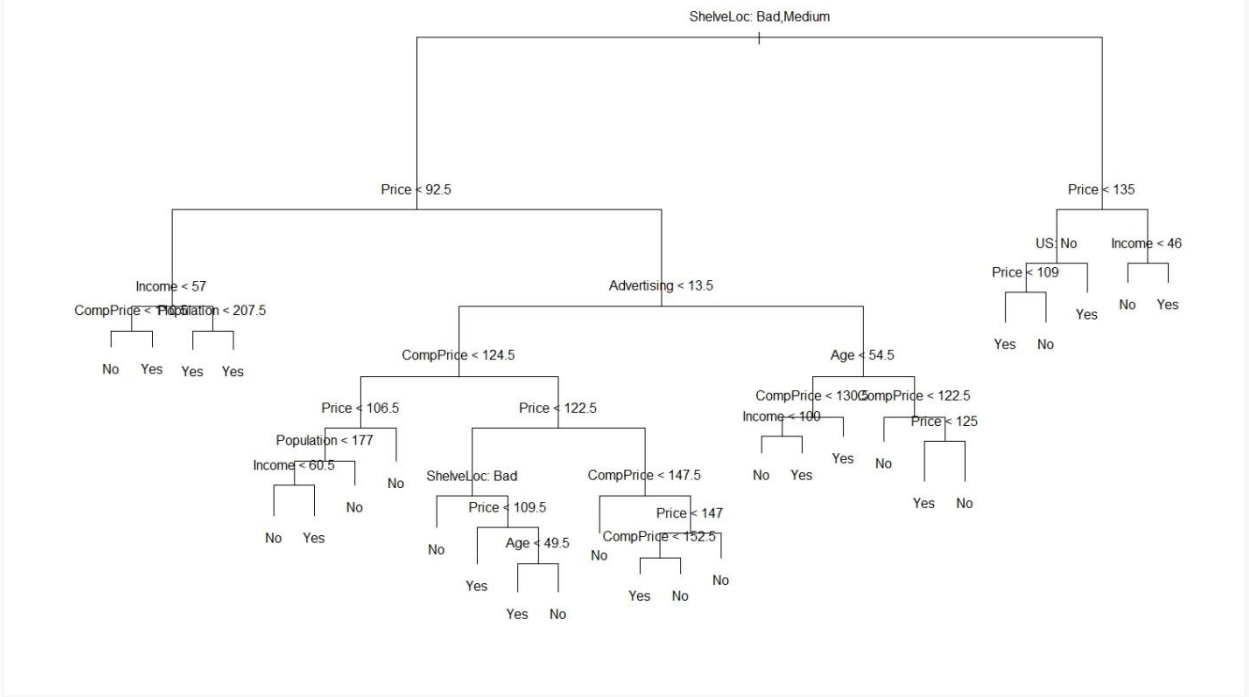
```
Carseats <- data.frame(Carseats, High)
```

Сада користимо функцију $tree()$ за класификациона стабла како би предвидели $High$ користећи све променљиве осим променљиве $Sales$. Затим ћемо погледати преглед стабла.

```
tree.carseats <- tree(High ~.-Sales, Carseats)
summary(tree.carseats)

##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

Сада можемо и нацртати стабло одлучивања за визуализацију. Овде морамо имати у виду да морамо додати текст стаблу користећи функцију $text()$.

```
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```




```
# Pogledajmo regije i slomove promenljive High na stablu
```

```
tree.carseats
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
```

```
##   1) root 400 541.500 No ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407 Yes ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
##          21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680 Yes ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702 No ( 0.90909 0.09091 ) *
##              85) ShelveLoc: Medium 40  52.930 Yes ( 0.37500 0.62500 )
##               170) Price < 109.5 16   7.481 Yes ( 0.06250 0.93750 ) *
##               171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
##                 342) Age < 49.5 13  16.050 Yes ( 0.30769 0.69231 ) *
##                 343) Age > 49.5 11   6.702 No ( 0.90909 0.09091 ) *
##            43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##              86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##              87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##               174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
##                 348) CompPrice < 152.5 7   5.742 Yes ( 0.14286 0.85714 )
*
##                 349) CompPrice > 152.5 5   5.004 No ( 0.80000 0.20000 ) *
##               175) Price > 147 7   0.000 No ( 1.00000 0.00000 ) *
##        11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
##          22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
##            44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
##              88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##              89) Income > 100 5   0.000 Yes ( 0.00000 1.00000 ) *
##            45) CompPrice > 130.5 11   0.000 Yes ( 0.00000 1.00000 ) *
##          23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##            46) CompPrice < 122.5 10   0.000 No ( 1.00000 0.00000 ) *
##            47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##              94) Price < 125 5   0.000 Yes ( 0.00000 1.00000 ) *
##              95) Price > 125 5   0.000 No ( 1.00000 0.00000 ) *
##     3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
##       6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
##        12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
##          24) Price < 109 8   0.000 Yes ( 0.00000 1.00000 ) *
##          25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##        13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
##       7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
##        14) Income < 46 6   0.000 No ( 1.00000 0.00000 ) *
##        15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *
```

Издвојићемо скуп за обучавање и контролни скуп:

```
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train ,]
High.test <- High[-train]
tree.carseats <- tree(High ~.-Sales, Carseats, subset=train)
```

Погледајмо предвиђање за подскуп који нисмо користили за учење и проверићемо колико је добра та оцена.

```
tree.pred <- predict(tree.carseats, Carseats.test, type = "class")
```

```
table(tree.pred, High.test)

##          High.test
## tree.pred  No Yes
##       No  104  33
##       Yes  13  50

mean(tree.pred == High.test)

## [1] 0.77
```

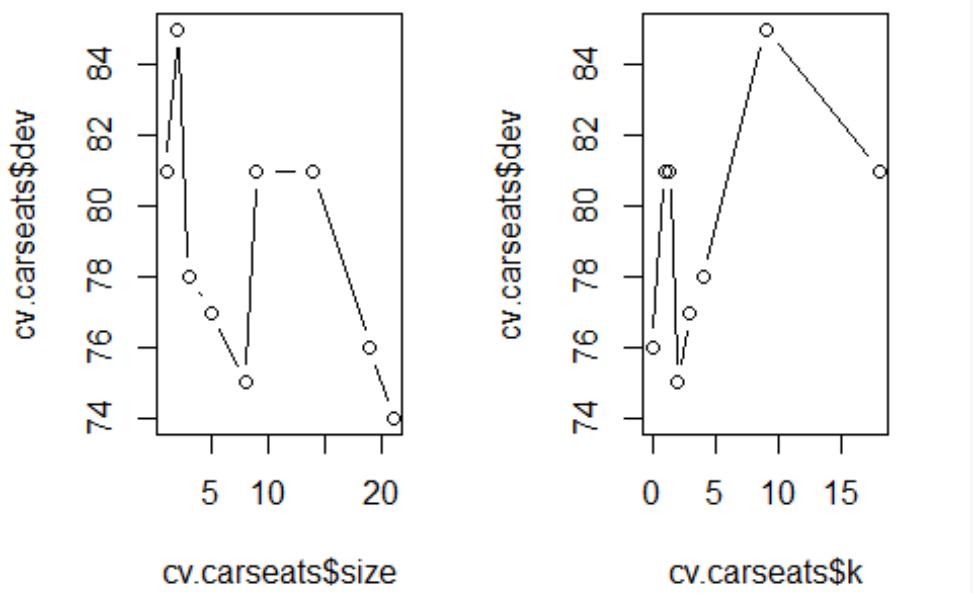
И након овог позива видимо да и није тако лоше,погађамо у 77% случајева.

Међутим, потенцијално бисмо могли да побољшамо модел уз унакрсну валидацију

```
set.seed(3)
cv.carseats <- cv.tree(tree.carseats, FUN = prune.misclass)
# pogledajmo dobijeni rezultat
cv.carseats

## $size
## [1] 21 19 14  9  8  5  3  2  1
##
## $dev
## [1] 74 76 81 81 75 77 78 85 81
##
## $k
## [1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

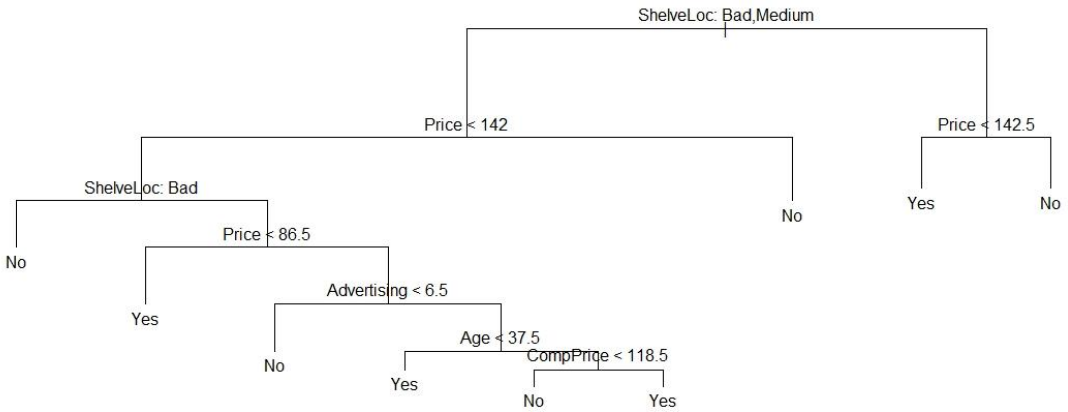
dev одговара грешци унакрсне валидације у овом случају. Видимо да дрво са 9 терминалних чворова резултира најнижом стопом унакрсне валидације. Нацртаћемо сада грешку као функцију од $size$ и k.

```
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```



Сада можемо да користимо функцију $prune.misclass()$ да обрежемо стабло до стабла са девет чворова.

```
prune.carseats <- prune.misclass(tree.carseats, best = 9)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```



На крају, проверавамо колико је добро наше подрезано дрво на контролном скупу.

```
# Pravimo predvidjanje na kontrolnom skupu
prune.pred <- predict(prune.carseats, Carseats.test, type = "class")
```

```
# Pogledajmo matricu konfuzije
table(prune.pred, High.test)

##           High.test
## prune.pred No Yes
##        No  97  25
##        Yes 20  58

# Proveravamo koliko je model dobar
mean(prune.pred == High.test)

## [1] 0.775
```

Добили смо да погађамо у 77,5% случајева. Оно не побољшава много наш модел али га зато доста поједностављује.

Шести задатак

Наћи одговарајуће предикторе за које применом метода невођеног учења можемо добити добро раздвајање симбола у Миноловац табли. Детаљно описати методу, предикторе и графички представити након примене метода редукције димензије ($t-SNE$ и PCA).

Седми задатак

Написати следеће функције:

(а) $prava\_matrica(matrica, dimenzija, broj\_mina)$ – за прослеђену већ попуњену до краја (нема ниједне преостале необележене мине) матрицу табеле Миноловца функција треба да врати одговор да ли је оваква матрица једна могућа табела Миноловца.

(б) $generator\_table(dimenzija, broj\_mina)$ – улазни аргумент димензије табле (не мора да буде квадратна) и број мина. Функција треба да врати матрицу – једну готову и потпуно попуњену таблу Миноловца.

(в) $sakrivanje\_polja(matrica, broj\_polja)$, која за произвољну матрицу – таблу Миноловца, затвори случајно одабраних $broj\_polja$ поља које нису мине и све мине.

(г) $MK\_simulacija(matrica, \dots)$ – за прослеђену матрицу табеле Миноловца функција треба да помоћу Монте Карло симулација одреди, које од преосталих поља има највећу вероватноћу да садржи мину и обратно, које поље има најмању вероватноћу за исто.

Објаснити и детаљно описати написане функције, као и све пропратне.

(а) $prava\_matrica(matrica, dimenzija, broj\_mina)$ – за прослеђену већ попуњену до краја (нема ниједне преостале необележене мине) матрицу табеле Миноловца функција треба да врати одговор да ли је оваква матрица једна могућа табела Миноловца.

Да би матрица била табла Миноловца мора да задовољава одређене услове:

1. Сва поља морају бити отворена
2. Број мина мора бити једнак укупном броју мина
3. Ако је поље празно, у околини не сме бити мина
4. Око поља са бројем, има тачно толико мина

Дакле, правимо функцију у којој ћемо да прверимо све наведене услове и уколико неки од услова није испуњен, функција ће да враћа $FALSE$.

```
prava_matrica <- function(matrica,dimenzija,broj_mina)
{
  M <- matrica
  M <- cbind(rep(1, dimenzija), M, rep(1, dimenzija))
  M <- rbind(rep(1, dimenzija+2), M, rep(1, dimenzija+2))

  # Broj mina mora biti jednak ukupnom broju mina
  if(sum(matrica==-100)!=broj_mina)
    return(FALSE)

  for(i in 2:(dimenzija+1))
  {
    for(j in 2:(dimenzija+1))
    {
      vektor <- c(M[i-1,j-1], M[i-1, j], M[i-1, j+1], M[i, j-1], M[i, j+1], M[i+1, j
-1], M[i+1, j], M[i+1, j+1])

      # Sva polja moraju biti otvorena
      if(M[i,j] == 0)
      {
        print("Matrica ima zatvorenih polja!")
        return(FALSE)
      }

      # Ako je prazno polje, u okolini ne sme biti mina
      if(M[i,j]==100)
      {
        if(sum(vektor==0)!=0)
        return(FALSE)
      }

      # Oko polja sa brojem ima tacno toliko mina
      for (k in 1:8)
      {
        if(M[i,j]==k*10)
        {
          if(sum(vektor==-100)!=k)
          return(FALSE)
        }
      }
    }

  }

}
```

(б) $generator\_table(dimenzija, broj\_mina)$ – улазни аргумент димензије табле (не мора да буде квадратна) и број мина. Функција треба да врати матрицу – једну готову и потпуно попуњену таблу Миноловца.

Правимо нула матрицу, димензије која је дата као улазни аргумент. На случајан начин изаберемо позиције на којој ћемо поставити мине(онолико колико је задато улазним аргументом).

Затим од те добијене матрице правимо нову матрицу, тако што додајемо по још једну врсту/колону са свим јединицама. Ово радимо да би све for петље прошле без проблема.

Пролазимо кроз сва поља табле Миноловца,и на основу околине поља, одређујемо да ли се у пољу налази број,при чему број мина у околини одређује који је број у питању, или је у питању празно поље.

На крају скидамо додате колоне/врсте.

```
generator_table <- function(n1,n2, broj_mina)
{
  # matrica ce predstavljati tablu minolovca
  matrica <- rep(0,n1*n2)

  # Slucajno izaberemo pozicije gde cemo postaviti mine
  pozicija_mina <- sort(sample(1:length(matrica), broj_mina))
  matrica[pozicija_mina] <- -100
  matrica <- matrix(matrica, nrow=n1)

  # Pravimo matricu koja ima u prvoj i poslednjoj vrsti(koloni)sve jedinice
  M <- matrica
  M <- cbind(rep(1, n1), M, rep(1, n1))
  M <- rbind(rep(1, n2+2), M, rep(1, n2+2))

  # Pravimo petlju koja prolazi kroz sva polja table minolovca
  for(i in 2:(n1+1))
  {
    for(j in 2:(n2+1))
    {
      # Okolina polja matrica[i,j]
      vektor <- c(M[i-1,j-1], M[i-1, j], M[i-1, j+1], M[i, j-1], M[i, j+1], M[i+1, j
-1], M[i+1, j], M[i+1, j+1])

      # Ako nije mina, mora biti ili broj ili prazno polje
      if (M[i,j]!=-100)
      {
        M[i,j] <- sum(vektor==-100)*10
        # Broj mina u okolini odredjuje koji je broj
        # Ako dobijemo da je ovo 0, nema mina u okolini, znaci prazno polje
        if (M[i,j]==0)
        M[i,j] <- 100
      }
    }
  }

  # Moramo da skinemo dodate kolone i vrste

  M <- M[-1, -1]
  M <- M[-(n1+1), -(n2+1)]
  return(M)
}
```

(в) $sakrivanje\_polja(matrica, broj\_polja)$, која за произвољну матрицу – таблу Миноловца, затвори случајно одабраних $broj\_polja$ поља које нису мине и све мине.

Прво ћемо затворити све мине, тј. уместо вредности -100 поставити вредности 0. Затим ћемо избацити позиције поља где су се налазиле мине и оставити само позиције на којима се налази или неки број или празно поље. На случајан начин изаберемо $broj\_polja$ тих поља и поставимо их на вредност 0 тј.затворимо их.

```
sakrivanje_polja <- function(matrica, broj_polja)
{
  M <- as.vector(matrica)
  dimenzija <- length(matrica[,1])

  # Prvo cemo zatvoriti sve mine
  M[M==-100] <- 0

  # Izbacujemo pozicije na kojima se nalaze mine(odnosno polje koje smo zatvoruli)
  # Ostavimo samo pozicije na kojima se nalazi broj ili prazno polje
  vektor <- seq(1:length(M))
  vektor <- vektor[-which(M == 0)]

  # Slucajno odaberemo polja gde nisu mine koja cemo da zatvorimo
  s <- sample(vektor,broj_polja)
  M[s] <- 0

  M <- matrix(M, nrow = dimenzija)
  return(M)
}
```

(г) $MK\_simulacija(matrica, \dots)$ – за прослеђену матрицу табеле Миноловца функција треба да помоћу Монте Карло симулација одреди, које од преосталих поља има највећу вероватноћу да садржи мину и обратно, које поље има најмању вероватноћу за исто.

Прво ћемо направити помоћну функцију $resi\_prosto$ која ће за прослеђену матрицу табле Миноловца, за сва поља, за која је могуће сигурно одлучити да ли је мина или не, да врати тачан одговор.

```
resi_prosto <- function(M,broj_mina)
{
  dimenzija <- length(M[1,])
  # Pravimo matricu m koja ima u prvoj i poslednjoj vrsti(koloni)sve jedinice,
  # da ne bismo imali problem sa for petljom i inedksiranjem
  m <- M
  m <- cbind(rep(1, dimenzija), m, rep(1, dimenzija))
  m <- rbind(rep(1, dimenzija+2), m, rep(1, dimenzija+2))

  # Prolazimo kroz sva polja table minolovca
  for(i in 2:(dimenzija+1))
  {
    for(j in 2:(dimenzija+1))
    {
      for(k in 1:8)
      {
```

```
        # Ako je broj u polju m[i,j] (u okolini mora biti tacno toliko mina)
        if(m[i, j]==k*10)
        {
          # Okolina polja m[i,j]
          vektor <- c(m[i-1,j-1], m[i-1, j], m[i-1, j+1], m[i, j-1], m[i, j+1], m[i+
1, j-1], m[i+1, j], m[i+1, j+1])
          # Ako je tacno k mina u okolini onda zatvorena polja ne kriju mine, pa moz
emo da ih otvorimo
          # Ovde to manifestujemo povecanjem za 1

          if(sum(vektor==-100)==k)
          {
            if(m[i-1, j-1]<9 & m[i-1, j-1]!=-100)
              m[i-1, j-1] <- m[i-1, j-1]+1
            if(m[i-1, j]<9 & m[i-1,j]!=-100)
              m[i-1, j] <- m[i-1, j]+1
            if(m[i-1, j+1]<9 & m[i-1,j+1]!=-100)
              m[i-1, j+1] <- m[i-1, j+1]+1
            if(m[i, j-1]<9 & m[i,j-1]!=-100)
              m[i, j-1] <- m[i, j-1]+1
            if(m[i, j+1]<9 & m[i,j+1]!=-100)
              m[i, j+1] <- m[i, j+1]+1
            if(m[i+1, j-1]<9 & m[i+1,j-1]!=-100)
              m[i+1, j-1] <- m[i+1, j-1]+1
            if(m[i+1, j]<9 & m[i+1,j]!=-100)
              m[i+1, j] <- m[i+1, j]+1
            if(m[i+1, j+1]<9 & m[i+1,j+1]!=-100)
              m[i+1, j+1] <- m[i+1, j+1]+1
          }
        }
      }

      # Ako je polje prazno, onda u okolini nema mina, mozemo da otvorimo sva polja
u okolini
      if(m[i,j]==100)
      {
        if(m[i-1, j-1]==0)
          m[i-1, j-1] <- m[i-1, j-1]+1
        if(m[i-1, j]==0)
          m[i-1, j] <- m[i-1, j]+1
        if(m[i-1, j+1]==0)
          m[i-1, j+1] <- m[i-1,j+1]+1
        if(m[i, j-1]==0)
          m[i, j-1] <- m[i,j-1]+1
        if(m[i, j+1]==0)
          m[i, j+1] <- m[i,j+1]+1
        if(m[i+1, j-1]==0)
          m[i+1, j-1] <- m[i+1,j-1]+1
        if(m[i+1, j]==0)
          m[i+1, j] <- m[i+1,j]+1
        if(m[i+1, j+1]==0)
          m[i+1, j+1] <- m[i+1,j+1]+1
      }
    }
  }

  for(i in 2:(dimenzija+1))
  {
    for(j in 2:(dimenzija+1))
    {
      for(k in 1:8)
```

```
      {
        if(m[i,j]==k*10)
        {
          vektor <- c(m[i-1,j-1], m[i-1, j], m[i-1, j+1], m[i, j-1], m[i, j+1], m[i+
1, j-1], m[i+1, j], m[i+1, j+1])
          # Ako je ukupno polja sa minama i zatvorenih polja k, onda sva zatvorena k
riju mine
          if(sum(vektor==-100)+sum(vektor==0)==k)
          {
            if(m[i-1, j-1]==0)
              m[i-1, j-1] <- -100
            if(m[i-1, j]==0)
              m[i-1, j] <- -100
            if(m[i-1, j+1]==0)
              m[i-1, j+1] <- -100
            if(m[i, j-1]==0)
              m[i, j-1] <- -100
            if(m[i, j+1]==0)
              m[i, j+1] <- -100
            if(m[i+1, j-1]==0)
              m[i+1, j-1] <- -100
            if(m[i+1, j]==0)
              m[i+1, j] <- -100
            if(m[i+1, j+1]==0)
              m[i+1, j+1] <- -100
          }
        }
      }
    }
  }

  # Ostalo je jos da odredimo koji su brojevi u poljima koja smo otvorili
  for (i in 2:(dimenzija+1))
  {
    for (j in 2:(dimenzija+1))
    {
      if (m[i,j]<10 & m[i,j]>0)
      {
        vektor <- c(m[i-1,j-1], m[i-1, j], m[i-1, j+1], m[i, j-1], m[i, j+1], m[i+1,
j-1], m[i+1, j], m[i+1, j+1])
        # Broj u polju je broj mina u njegovoj okolini
        m[i,j] <- sum(vektor==-100)*10
      }
    }
  }

  # Skidamo dodate vrste i kolone
  m <- m[-1, -1]
  m <- m[-(dimenzija+1), -(dimenzija+1) ]

  # Ako je broj mina bas jednak trazenom, nema vise mina
  if(sum(m==-100)==broj_mina)
  {
    return(m)
  }

  # A ako je broj mina manji od navedenog, i ako je manji bas za broj zatvorenih pol
ja, onda sigurno znamo
  # da se iza tih zatvorenih polja kriju mine
  if(sum(m==-100)<broj_mina)
  {
```

```
    if(sum(m==0)+sum(m==-100)==broj_mina)
    {
      for(i in 1:dimenzija)
      {
        for(j in 1:dimenzija)
        {
          if(m[i,j]==0)
            m[i,j] <- -100
        }
      }
    }
  }

  return(m)
}
```

Сада можемо да правимо функцију $MK\_simulacija(matrica, \dots)$.

Идеја је уз помоћ функције $resi\_prosto$ прво отворимо сва поља за која са сигурношћу знамо шта крију. Затим пролазимо кроз таблу Миноловца да покупимо позиције затворених поља, која чувамо у посебној листи,и одређујемо колико има затворених поља.

Радимо 1000 симулација и бројимо појављивање мина иза сваког од затворених поља, али тако да нам је таква матрица права табла Миноловца.На крају одређујемо иза ког поља се најчешће појављивала мина, а из ког најређе и то враћамо као резултат.

```
MK_simulacija <- function(M,broj_mina)
{
  dimenzija <- length(M[1,])
  # Prvo otvorimo sva polja za koja sa sigurnoscu znamo sta kriju
  M <- resi_prosto(M,broj_mina)

  # Lista u kojoj cemo cuvati pozicije(u matrici minolovca) zatvorenih polja
  zatvorena <- list()

  k <- 1
  # Prolazimo kroz tabelu minolovca da pokupimo pozicije zatvorenih polja
  for (i in 1:dimenzija)
  {
    for (j in 1:dimenzija)
    {
      if (M[i,j]==0)
      {
        zatvorena[[k]] <- c(i,j)
        k <- k + 1
      }
    }
  }

  k <-length(zatvorena) # Ovoliko imamo  zatvorenih polja
  z_polja <- rep(0,k) # Ovde cemo beleziti pojavljivanja mina iza svakog od zatvoren
ih polja iz liste

  # Radimo 1000 simulacija, brojimo pojavljivanja mina iza svakog od zatvorenih polj
a, ali tako da
  # nam je takva tabla prava tabla minolovca

  for (t in 1:1000)
```

```
  {
    m <- M
    n <- broj_mina - sum(as.vector(M)==-100) # n je broj zatvorenih polja koja kriju
mine
    s = sort(sample(1:k,n)) # Biramo n mesta slucajno, od ukupnog broja zatvorenih p
olja

    # Iz liste zatvorenih polja nadjemo pozicije mesta koja smo odabrali, i postavim
o da budu mine
    for (i in 1:n)
      m[zatvorena[[s[i]]][1],zatvorena[[s[i]]][2]] <- -100

    # Opet dodajemo vrste i kolone da ne bismo imali problema
    m <- cbind(rep(1, dimenzija), m, rep(1, dimenzija))
    m <- rbind(rep(1, dimenzija+2), m, rep(1, dimenzija+2))

    # Sve mine smo rasporedili, ostala  zatvorena polja su brojevi ili prazna,
    # sto vidimo iz toga koliko je mina u okolini
    for (i in 2:(dimenzija+1))
    {
      for (j in 2:(dimenzija+1))
      {
        if (m[i,j]==0)
        {
          vektor <- c(m[i-1,j-1], m[i-1, j], m[i-1, j+1], m[i, j-1], m[i, j+1], m[i+
1, j-1], m[i+1, j], m[i+1, j+1])
          m[i,j] <- sum(vektor==-100)*10
          # Uklapa se i prazno, onda je sum(vektor==-100)==0, nema mina u okolini
        }
      }
    }

    # Skinemo dodate kolone i vrste
    m <- m[-1, -1]
    m <- m[-(dimenzija+1), -(dimenzija+1)]

    # Ako smo dobili pravu tablu minolovca, belezimo u vektoru z_polja iza kojih zat
vorenih polja su nam mine
    # tj. povecavamo broj pojavljivanja polja na odgovarajucim pozicijama za 1
    if (prava_matrica(m,dimenzija,broj_mina))
    {
      for (i in 1:n)
        z_polja[s[i]] <- z_polja[s[i]]+1
    }
  }

  # Iza njega je najcesce bila mina
  v1 <- zatvorena[which.max(z_polja)]
  # Iza njega je najmanje puta bila mina
  v2 <- zatvorena[which.min(z_polja)]

  return (c(v1,v2))
}
```

Остало

Осми задатак

Написати функцију која симулира и непосредно игра игру *„Детерминанта“* за произвољне димензије табли на основу Монте Карло методе.

Игра *„Детерминанта“* има следећа правила. Дата је квадратна матрица димензија $(n\ x\ n)$, која је у старту празна и дати су бројеви $\{1,2,\dots,n^2\}$. Два играча, један за другим, попуњавају дату матрицу једним од бројева из задатог скупа (без враћања). На крају побеђује играч један ако је детерминанта добијене матрице позитивна, односно играч два ако је негативна.

```
igra_determinanta <- function(N,matrica,moguci_brojevi,ind_prvog)
{
  # Od date matrice pravimo vektor
  vektorizacija <- as.vector(matrica)

  # M je velika matrica u koju cemo da smestamo nase verovatnoce
  M <- matrix(rep(0,81),ncol=9,nrow=9)

  # Moguca mesta su mesta koja jos nisu popunjena u matrici
  moguca_mesta <- which(vektorizacija==0)

  for(i in moguci_brojevi)
  {
    for(j in moguca_mesta)
    {
      # Kako smo fiksirali jedno mesto i jedan broj, moramo ih izbaciti sa raspolaga
nja
      moguca_m <- moguca_mesta[moguca_mesta!=j]
      moguci_b <- moguci_brojevi[moguci_brojevi!=i]

      for(k in 1:N)
      {
        # Dobijemo neku slucajnu permutaciju mogucih mesta i samim tim i brojeva.
        if(length(moguca_m) > 1)
        {
          x <- sample(moguca_m, length(moguca_m), replace=FALSE)
        }
        else
        {
          x <- moguca_m
        }
        y <- moguci_b # pravljenje samo jednog 'sample'-a dvostruko ubzava kod

        # Ubacimo sve u nasu vektorizovanu matricu.
        for(l in 1:length(moguci_b))
        {
          vektorizacija[x[l]]=y[l]
        }
        vektorizacija[j] <- i
        matr <- vektorizacija
        dim(matr) <- c(3,3)
```

```
        D <- det(matr)
        if(D>0 & ind_prvog==1)
        {
          M[i, j] = M[i, j]+1
        }
        if(D<0 & ind_prvog==0)
        {
          M[i, j] = M[i, j]+1
        }
      }
    }
  }

  M <- M/N
  return(M)
  # vraca verovatnocu kada je detminanta pozitivna, za broj i stavljen na j-oto mest
o (na mestu (i,j) u  matrici)
}

# Napravimo praznu matricu
matrica <- matrix(rep(0,9), ncol=3)

# m - matrica koju vraca funkcija
# Odabir gde da se postavi prva cifra, mada svejedno je:
m <- igra_determinanta(100, matrica, c(1,2,3,5,6,7,8,9), T)
m

##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,] 0.49 0.50 0.62 0.47 0.51 0.50 0.50 0.51 0.48
##  [2,] 0.49 0.43 0.54 0.49 0.53 0.39 0.51 0.45 0.46
##  [3,] 0.48 0.56 0.58 0.49 0.42 0.55 0.53 0.43 0.42
##  [4,] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
##  [5,] 0.50 0.44 0.49 0.42 0.54 0.57 0.50 0.55 0.44
##  [6,] 0.42 0.57 0.45 0.53 0.46 0.61 0.51 0.51 0.41
##  [7,] 0.50 0.48 0.49 0.43 0.51 0.47 0.52 0.50 0.49
##  [8,] 0.59 0.45 0.65 0.46 0.58 0.54 0.54 0.57 0.53
##  [9,] 0.52 0.44 0.50 0.60 0.43 0.55 0.57 0.47 0.52

simulacija_igre <- function(N, dimenzija)
{
  matrica <- matrix(0, dimenzija[1], dimenzija[2])
  n <- prod(dimenzija)
  for(i in 1:n)
  {
    message("Pocinje ", i, ". potez.")

    m <- igra_determinanta(N = i * 3000, matrica = matrica,moguci_brojevi = 1:n, ind
_prvog = i %% 2)


    verovatnoca <- m[which.max(m)]
    broj_pozicija <- which(verovatnoca == m, arr.ind = TRUE)
    broj <- broj_pozicija[1, 1]
    pozicija <- broj_pozicija[1, 2]
    if(verovatnoca == 0)
    {
      if(i %% 2 == 0)
      {
        igrac_koji_pobedjuje = 'prvi'
      }
      else
```

```
      {
        igrac_koji_pobedjuje = 'drugi'
      }
      message('Nema smisla nastavljati, pobedio je: ', igrac_koji_pobedjuje, ' igrac
!')
      return()
    }
    # upisemo trazeni broj
    matrica[pozicija] <- broj
    message("Posle ", i, "-tog poteza:")
    print(matrica)
  }
  det(matrica)
}

simulacija_igre(1000, c(3,3))

## Pocinje 1. potez.

## Posle 1-tog poteza:

##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    9    0    0

## Pocinje 2. potez.

## Posle 2-tog poteza:

##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    0    0
## [3,]    9    0    0

## Pocinje 3. potez.

## Posle 3-tog poteza:

##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    0    0
## [3,]    9    0    9

## Pocinje 4. potez.

## Posle 4-tog poteza:

##      [,1] [,2] [,3]
## [1,]    1    1    0
## [2,]    0    0    0
## [3,]    9    0    9

## Pocinje 5. potez.

## Posle 5-tog poteza:

##      [,1] [,2] [,3]
## [1,]    1    1    0
## [2,]    0    0    9
## [3,]    9    0    9

## Pocinje 6. potez.
```

```
## Posle 6-tog poteza:

##      [,1] [,2] [,3]
## [1,]    1    1    9
## [2,]    0    0    9
## [3,]    9    0    9

## Pocinje 7. potez.

## Posle 7-tog poteza:

##      [,1] [,2] [,3]
## [1,]    1    1    9
## [2,]    0    1    9
## [3,]    9    0    9

## Pocinje 8. potez.

## Posle 8-tog poteza:

##      [,1] [,2] [,3]
## [1,]    1    1    9
## [2,]    1    1    9
## [3,]    9    0    9

## Pocinje 9. potez.

## Nema smisla nastavljati, pobedio je: drugi igrac!
```