Данијел Алексић

Статистички софтвер 3

Увод у статистичко и машинско учење

Универзитет у Београду
Математички факултет

Садржај

Предговор

Ова скрипта настала је[1] као резултат мог држања вежби из Статистичког софтвера 3 на Математичком факултету Универзитета у Београду.

Већи део скрипте настао је преобличавањем презентација колеге Данијела Суботића, који је неколико година пре мене држао овај курс и осмислио га у овом облику у ком данас постоји. На томе сам му бескрајно захвалан, као и на стварима које сам од њега научио на другим курсевима које сам код њега слушао.

Део курса преузео сам из књиге *An Introduction to Statistical Learning* Роберта Тибширанија и групе аутора, коју топло препоручујем.

Захвалан сам и Благоју Ивановићу, који ме је научио основама програмирања у програмском језику `R` и који ме је упутио на литературу за напредни ниво.

Хвала мом претходнику на овом курсу, Жикици Лукићу, на примерима из индустрије које ми је навео на основу свог радног искуства, а који су учинили да ови материјали буду блиски са стварним светом.

Хвала доц. др Бојани Милошевић, мом садашњем ментору, на томе што ме је научила сву теорију неопходну за разумевање алгоритама представљених у овим материјалима.

Молио бих сваког ко након мене буде држао овај курс да ми пише на мејл `danijel.aleksic98@gmail.com` како бих му послао изворни код скрипте да је додатно унапређује и, наравно, допише себе као аутора. Ако у тренутку читања ових редова и даље држим овај курс, молим да ми се за све грешке у скрипти обрати на `danijel_aleksic@matf.bg.ac.rs`.

У Београду,
Љета Господњег 2021,
Сочинитељ.

[1] Одбијам да прихватим да је *скрипта* реч у множини.

Вежбе 1

Упознавање са курсом. Регресија

Вежбе 2

Регресија - наставак. Класификација

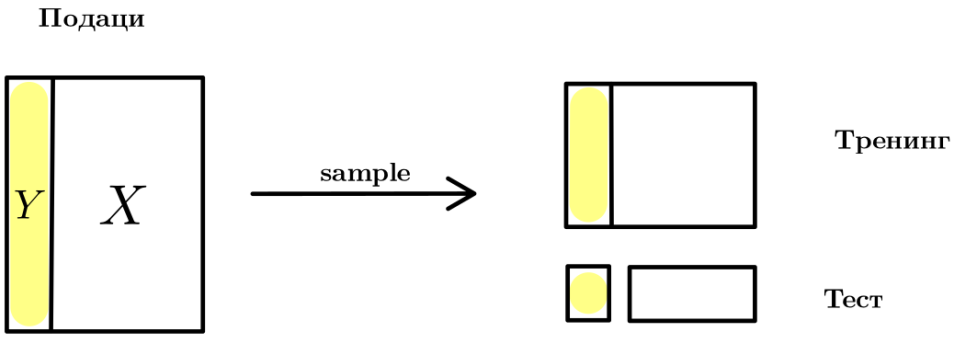
Вежбе 3

Класификациони алгоритми: kNN, LDA QDA

Вежбе 4

Реузорковање

Методе реузорковања су један од основних алата у модерној статистици. Рецимо, на пример, да имамо проблем линеарне регресије и да желимо да оценимо регресионе коефицијенте. Сходно том циљу, поделили смо наше податке на тренинг и на тест скуп, како бисмо на тренинг скупу добили саму оцену, а на тест скупу проверили грешку.



Уколико радимо са подацима великог обима, ово је углавном све што нам треба. Међутим, уколико обим нашег узорка није задовољавајући, или уколико из било ког разлога немамо приступ тест скупу (гуглати: DATATHON), мора се прибећи провери квалитета на самом тренинг скупу.

Такође, у таквим ситуацијама може се десити да је тренинг скуп нерепрезентативан, то јест да смо, случајно, поделили податке на тренинг и тест скуп на тај начин да су оцене коефицијената лоше. Стога се прибегава подели самог тренинг скупа на два дела, где ћемо на једном правити модел, а на другом, који ћемо звати *валидациони скуп*, проверавати квалитет модела. О чему причамо размотрићемо на примеру.

4.0.1 Пример - регресија

Посматрајмо `Auto` базу из пакета `ISLR2`. За почетак, ваља се упознати са базом. Она садржи 392 опсервације (аутомобила) за 9 променљивих:

- `mpg` - колико миља аутомобил може прећи са галоном горива (подаци су за САД);
- `cyllinders` - број клипова у мотору (између 4 и 8);
- `displacement` - кубикажа (у кубним инчима);

- `horsepower` - број коњских снага;
- `weight` - тежина возила;
- `acceleration` - време потребно да се убрза од 0 до 60 миља по сату (у секундама);
- `year` - година производње по модулу 100;
- `origin` - порекло возила: 1 за Америку, 2 за Европу и 3 за Јапан;
- `name` - име возила.

Рецимо да желимо да моделујемо број миља за галон на основу броја коњских снага. Стандардно, прва ствар која пада на памет јесте да правимо линеаран модел, али ћемо пре тога поделити податке на тренинг и тест скуп, тј. издвојићемо само тренинг скуп јер нам је он овде до интереса. Претварaћемо се да тест скуп немамо, већ да њега чува налогодавац истраживања, како би за себе проверио да ли наш модел стварно ваља.

```
library(ISLR2)

n <- length(Auto$mpg)
obim <- floor(0.7*n) # Delimo 70:30

train_idx <- sample(1:n, size = obim)
Auto_train <- Auto[train_idx, ]
```

Правимо модел.

```
model1 <- lm(mpg ~ horsepower, data = Auto_train)
```

Сада ћемо да извршимо предвиђање на основу овог модела.

```
predvidjanje1 <- predict(model1)
```

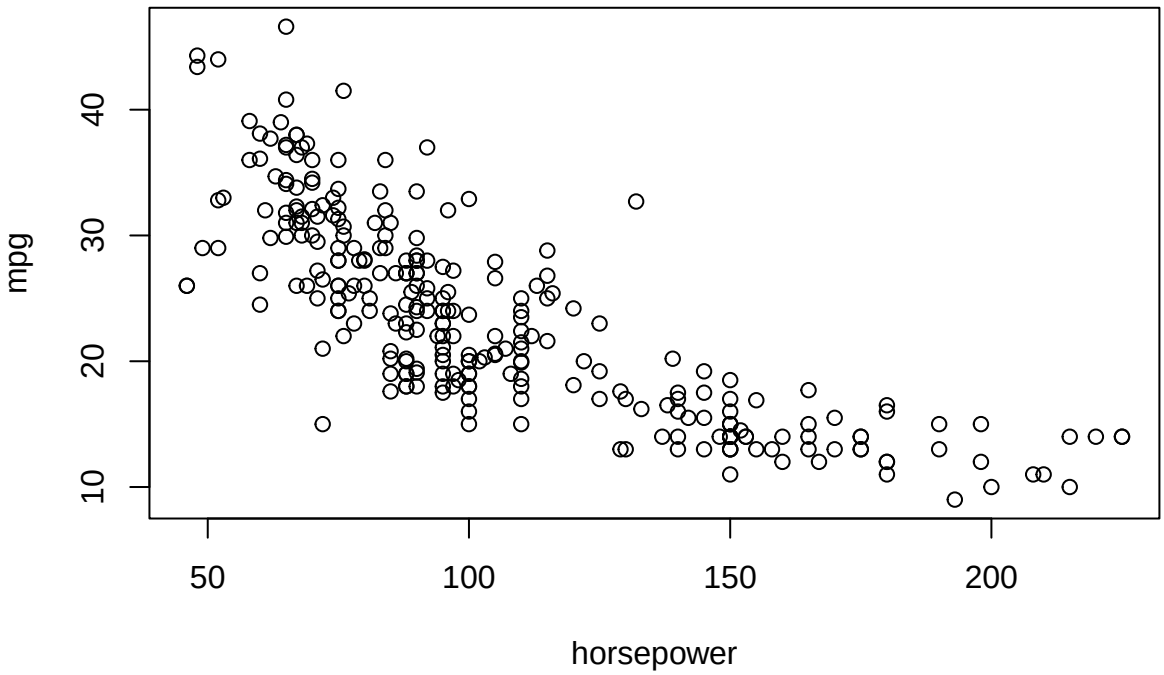
Сада ћемо да видимо колика нам је грешка. Користимо средњеквадратну грешку као показатељ.

```
mean((Auto_train$mpg - predvidjanje1)^2)
```

```
## [1] 22.88023
```

Хајде сада да исцртамо ову зависност.

```
plot(mpg ~ horsepower, data = Auto_train)
```



Овако са слике, није да је веза баш линеарна. Стога се намеће додавање члана $\texttt{mpg}^2$ у регресију.

```
model2 <- lm(mpg ~ horsepower + I(horsepower^2), data = Auto_train)
```

Вршимо предвиђање на основу другог модела и рачунамо грешку.

```
predvidjanje2 <- predict(model2)
mean((Auto_train$mpg - predvidjanje2)^2)
```

```
## [1] 18.74491
```

У овом случају добијамо значајно мању средњеквадратну грешку. Отуда закључујемо да је бољи модел онај код којег смо додали и квадратни члан.

У реду, рећи ће свако ко је догурао довде. У чему је фора? Ово већ знамо! Е па фора је у томе што радимо са тренинг скупом. Можда се у ових наших 70 посто са којима радимо заломило баш оних 70 посто који дају обрнут резултат. Можда додавање квадратног члана чак и квари прецизност предвиђања, само су се нама подаци „потрефили'' да нас наведу на погрешан закључак. Можда је грешка на нама недоступном тестном скупу другачија.

Е ту на сцену ступа реузорковање. Делимо доступне нам податке на „нове тренинг'' податке (зовимо их под-тренинг, да не дође до забуне) и на валидациони скуп. Модел правимо искључиво на под-тренинг подацима, а затим грешку проверавамо на валидационом. То понављамо довољан број пута, те гледамо мења ли се шта у оценама наших коефицијената од случаја до случаја. То се може урадити на много начина, а ми ћемо кренути од најнаивнијег до оних тежих.

Обратити пажњу: Може доћи до забуне у именима. Наиме, ми смо овде рекли да је неки имагинарни наручилац пројекта задржао део података за себе (тест), да би на њима проверио ваља ли оно што ми радимо, а нама послао остатак (тренинг). Ми смо то симулирали издвајањем `Auto_train`. Ми затим из тог тренинга издвајамо валидациони скуп, углавном више пута. Неко податке које смо добили зове просто подаци, а оно што ми издвајамо зове тренинг и тест, те је онда тест скуп само специјалан случај валидационог, када делимо само једном. То је само језички проблем, а ми ћемо се овде држати номенклатуре: [тренинг + валидациони] + тест.

Чисто да видимо шта се дешава са грешкама, одрадићемо мини демонстрацију. Поделићемо на пола тренинг скуп, на једној половини правити модел, на другој валидирати, те гледати какве су грешке на обе половине, све поновивши 10 пута.

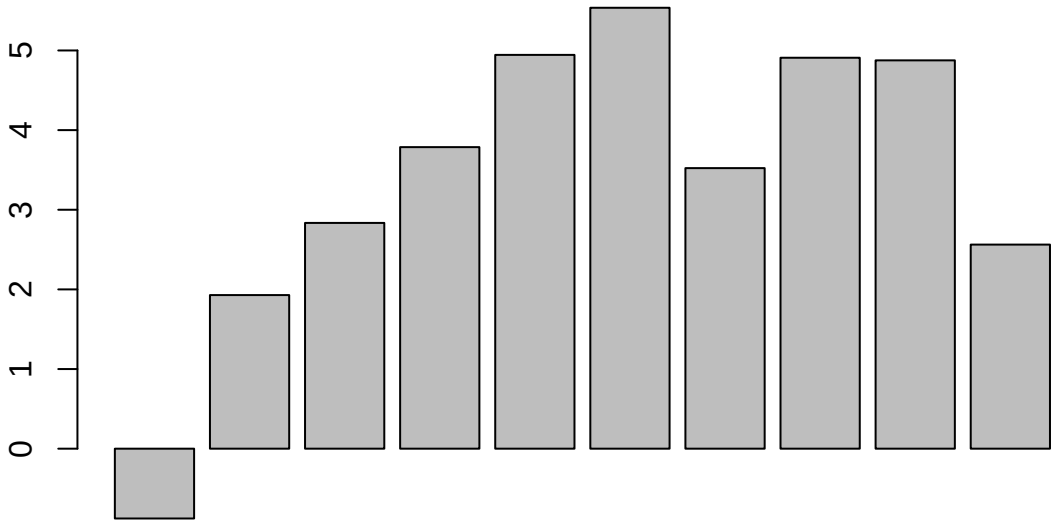
```
# Obim treninga i validacionog (pola)
n_tr <- length(Auto_train$mpg)

# Ovde cemo da upisemo greske
MSE1 <- c()
MSE2 <- c()

for (i in 1:10) {
  # Biramo indekse za pod-trening
  tr_idx_tmp <- sample(1:n_tr, size = floor(n_tr / 2))
  # Izdvajamo validacioni
  Auto_tr_tmp <- Auto_train[tr_idx_tmp, ]
  Auto_vld_tmp <- Auto_train[-tr_idx_tmp, ]

  # Pravimo oba modela na Auto_tr_tmp
  model1_tmp <- lm(mpg ~ horsepower, data = Auto_tr_tmp)
  model2_tmp <- lm(mpg ~ horsepower + I(horsepower^2), data = Auto_tr_tmp)

  # Vrsimo predvidjanje na validacionom skupu za oba modela
  pred1_tmp <- predict(model1_tmp, newdata = Auto_vld_tmp)
  pred2_tmp <- predict(model2_tmp, newdata = Auto_vld_tmp)

  # Sada pravimo vektore gresaka za modele
  MSE1 <- c(MSE1,
            mean((Auto_vld_tmp$mpg - pred1_tmp)^2))
  MSE2 <- c(MSE2,
            mean((Auto_vld_tmp$mpg - pred2_tmp)^2))
}
```

Дакле, 10 пута смо половили тренинг скуп и на једној половини правили модел, а на другој валидирали. Сада ћемо грешке претворити у разлике: посматраћемо разлику грешака за модел 2 и модел 1.

```
MSE_diff <- MSE1 - MSE2
```

Нацртајмо.

```
barplot(MSE_diff)
```



Видимо да се у једном случају, првом, добија негативна разлика, то јест недодавање квадратног члана даје бољи резултат! Да смо податке које смо добили од послодавца поделили баш на тај начин, извукли бисмо погрешан закључак! Тек након већег броја реузорковања, видимо да је у већини случајева додавање квадратног члана ипак бољи избор.

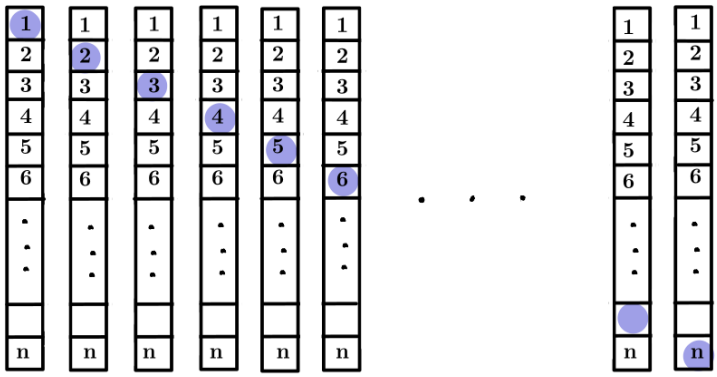
Напомена. Како у валидационом приступу правимо модел на мањем скупу, он ће да буде лошији (доказ: ЛСМ), те валидациони приступ генерално грешке приказује већим него што јесу.

4.1 Кросвалидација

Претходна прича о валидацији била је јасна и праволинијска, али ипак у неком смислу наивна. Види се да ту има простора за побољшање. Једно од таквих побољшања приче о валидацији јесте унакрсна валидација, позната још и као кросвалидација (јер се крст на неоколонијалном каже *cross*).

4.1.1 LOOCV

Први тип кросвалидације јесте валидација са избацивањем једног елемента (*Leave One Out Cross-Validation*). Малопре смо имали једноставан приступ: скуп опсервација смо делили на два једнака дела. Овај пут, ипак, урадићемо нешто сасвим другачије. **Издвојићемо само једну опсервацију**, на остатку направити модел, а на њој валидирати.



Нека смо издвојили опсервацију $(\mathbf{x}_1, y_1)$, на остатку направили модел и срачунали грешку $MSE_1 =$

$(y_1 - \hat{y}_1)^2$. Затим избацимо $(\mathbf{x}_2, y_2)$, направимо модел на остатку и срачунамо MSE_2. Настављамо даље све док не срачунамо и MSE_n. LOOCV оцена за грешку на целом скупу сада је дата са

$$\mathrm{CV}_{(n)} = \frac{1}{n} \sum_{i=1}^{n} MSE_i.$$

Овај приступ, за разлику од претходног (половљење), обично не прецењује грешку. Такође, тамо смо вршили много мање подела од могућег броја подела, док смо овде сваку поделу типа $1 + (n-1)$ узели у обзир. Сходно томе, овај приступ је за разлику од претходног слободан од случајности.

Делује да смо на добром путу. Међутим, **појављује се један велики проблем**. Модел се прави чак n пута! Ово је рачунарски веома захтеван процес. Срећом математичари су ухватили пречицу. Ако дефинишемо

$$h_i = \frac{1}{n} + \frac{(x_i - \overline{x})^2}{\sum_{i=1}^{n} (x_i - \overline{x})^2},$$

за **просту** линеарну регресију важи да је:

$$\mathrm{CV}_{(n)} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2,$$

где су y_i оригиналне фитоване вредности на целом тренинг скупу. Формула важи и за линеарну регресију која није проста, само се тада тежине h_i дефинишу мало другачије (требало би да буде на ЛСМ).

LOOCV може да буде коришћен не само са регресијом, већ и са разним типовима класификације. Нажалост, горња формула важи само у случају регресије, те у свим другим случајевима фитовање мора бити извршено n пута.

4.1.2 k-fold кросвалидација

Једна од алтернатива за LOOCV јесте k-fold кросвалидација (нема неко добро име на српском). Идеја је да се скуп опсервација подели на k дисјунктних скупова (*folds*), приближно исте величине. Затим се прва група третира као валидациони скуп, а модел се прави на осталих $k-1$ узетих заједно. Затим се на том првом рачуна грешка MSE_1. Слично се понавља са осталима и добија се низ грешака MSE_1, MSE_2, ..., MSE_k, од којих се прави оцена

$$\mathrm{CV}_{(k)} = \frac{1}{k} \sum_{i=1}^{k} MSE_i.$$

Овде се поставља ново питање: које k користити? Не постоји добар одговор. У суштини веће k даје боље резултате јер су „међумодели'' бољи, али може да буде веома захтевно за рачунар. Резултати за мало k су грубљи, али лакше израчунљиви. Свако треба да бира своје k спрам модела који користи, спрам обима узорка који има и спрам рачунара на којем ради.

4.1.2.1 Пристрасност vs дисперзија за k-fold

Када смо помињали однос LOOCV алгоритма и приступа пола-пола, поменули смо да LOOCV даје непристраснију оцену средњеквадратне грешке на тестном скупу неголи метод пола-пола, јер потоњи модел прави само на половини података. Дакле, спрам непристрасности, LOOCV је свакако пожељнији метод. Из истог разлога LOOCV је пожељнији у односу на k-fold.

Међутим, није пристрасност оцене једина ствар која треба да нас брине. Наиме, испоставља се да LOOCV процедура има значајно већу дисперзију у односу на k-fold. Шта то значи и откуда то? Наиме, када вршимо LOOCV, ми упросечавамо излазе n модела који су прављени на скоро идентичним подацима. Стога су ти излази међусобно јако позитивно корелисани. Са друге стране, ако примењујемо k-fold, ми упросечавамо $k < n$ много мање корелисаних модела, јер су преклапања мања. Стога је оцена коју даје k-fold са мањом дисперзијом од LOOCV.

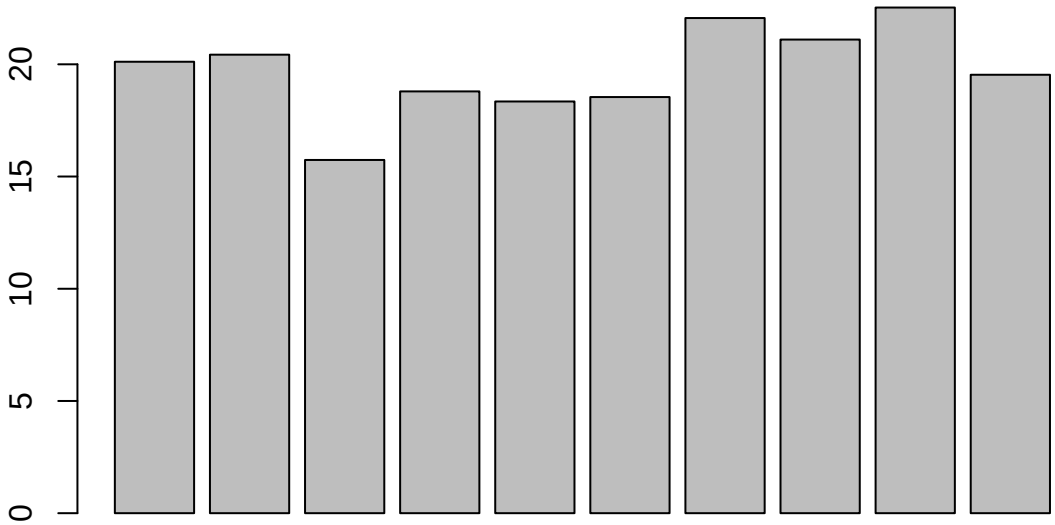
Да сумирамо, **LOOCV има мању пристрасност оцене, али већу дисперзију, док је за k-fold обрнуто.**

4.1.2.2 Наставак примера

Сада ћемо се вратити примеру са базом `Auto`, и применити кросвалидациони приступ. Користићемо модел са квадратним чланом.

За почетак, нацртајмо грешке (не разлике - већ баш грешке) за приступ пола/пола:

```
barplot(MSE2)
```



Види се значајна варијабилност у грешкама, за различите поделе. Одрадићемо и 10-fold, те упоредити варијабилност оцене за MSE.

У `R`-у постоји библиотека `caret` (*Classification and Regression Training*), која је једна од најпознатијих библиотека за машинско учење које постоје за `R`.

```
library(caret)
```

Сада ћемо да направимо линеарни модел у овом пакету и да извршимо кросвалидацију.

```
train_control_10f <- trainControl(method = "cv", number = 10)
```

Ово је објекат (листа), која заправо служи као упакован начин да се наредној функцији каже шта да ради.

```
model_10fold <- train(mpg ~ horsepower + I(horsepower^2), data = Auto_train,
                      trControl = train_control_10f, method = "lm")
```

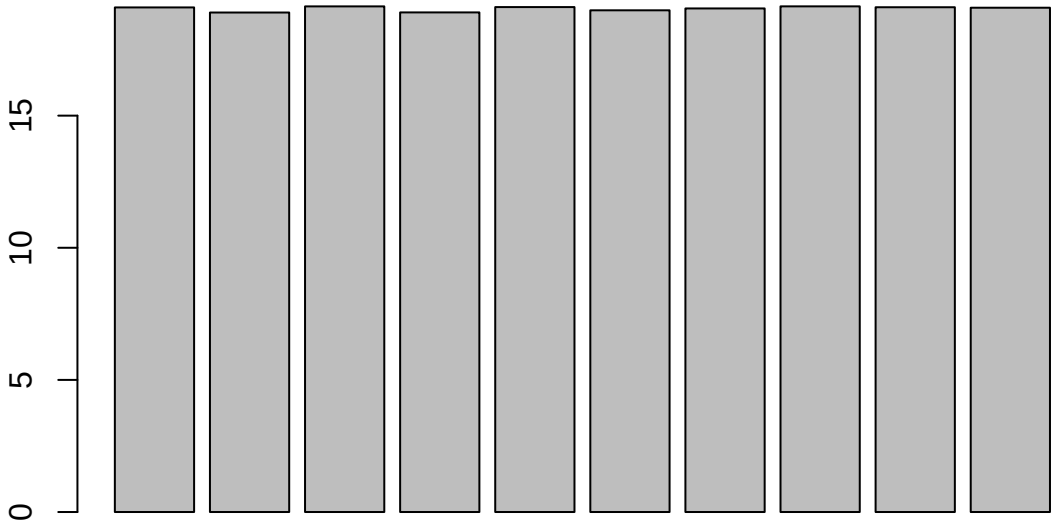
У повратној вредности `model$resample$RMSE` чувају се корени средњеквадратних грешака за сваки фолд. Одавде може да се добије кросвалидациона оцена као

```
sum((model_10fold$resample$RMSE)^2)/10
```

```
## [1] 19.09497
```

Сада ћемо поновити поступак десет пута, и видети како варирају оцене.

```
cv_10_err <- c()
for (i in 1:10) {
  model_i <- train(mpg ~ horsepower + I(horsepower^2), data = Auto_train,
                   trControl = train_control_10f, method = "lm")
  cv_i <- sum((model_i$resample$RMSE)^2) / 10
  cv_10_err <- c(cv_10_err,
                 cv_i)
}
barplot(cv_10_err)
```



Видимо много (!!!) мању варијабилност оцене MSE у односу на пола-пола приступ. Наравно, нема баш смисла радити ово исто за LOOCV, јер бисмо свих десет пута добили исту вредност, јер немамо случајан одабир фолдова.

Напомена. У књизи *An Introduction to Statistical Learning* предлаже се коришћење функције `cv.glm` из пакета `boot`, и то у комбинацији са функцијом `glm` која прави уопштен линеаран модел (регресиони, класификациони). Припремајући ове материјале, уочио сам један велики проблем са овом функцијом. Наиме, функцији `cv.glm` треба као аргумент дати базу података (ОК), број фолдова (ОК) и већ фитован модел класе `glm` (???). Ова функција не само да користи један модел за оцену грешке на свим фолдовима, већ нема ни могућност случајног генерисања поделе на фолдове (као `train` из `caret`-а), већ увек дели на исти начин. Дакле, изузетно бескорисна функција за сваку проверу варијабилности. Не препоручујем је за било какву употребу.

Мало мање важна напомена. Нисмо хтели да поредимо који од пола-пола, LOOCV, 10-fold даје бољу оцену грешке на тест скупу, јер је наш модел исувише наиван за било какве озбиљне закључке.

4.1.3 Кросвалидација и класификациони проблеми

До сада смо кросвалидацију демонстрирали на проблему регресије где је зависна променљива квантитативно обележје. Међутим, кросвалидација је подједнако корисна и за проблеме класификације, где је зависна променљива катеоричког типа.

На пример, за LOOCV при класификацији имамо оцену:

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{ERR}_i,$$

где је $\mathrm{ERR}_i = I\{y_i \neq \hat{y}_i\}$. Овде, чак, $\mathrm{CV}_{(n)}$ има више смисла него код регресије: **он представља проценат погрешно класификованих опсервација**.

Кросвалидациона оцена за k-fold се дефинише аналогно.

4.2 Бутстреп

Код бутстреп приступа прескочићемо теоријски увод, те се одмах бацити на пример - тако ће нам бити јасније. Рецимо да желимо да уложимо фиксну своту новца у два асета на берзи, који, редом, имају приносе X и Y, где су X и Y случајне величине. Желимо да уложимо фракцију $\alpha \in (0,1)$ нашег новца у први асет, а $1-\alpha$ у други. Овакво улагање желимо да направимо тако да дисперзија нашег портфолија буде што мања. Другим речима, желимо да минимизујемо

$$D\left(\alpha X + (1-\alpha)Y\right).$$

Може се показати (није тема овог курса), да је оно α које минимизује горњу дисперзију дато са:

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}},$$

где је $\sigma_X^2 = DX$, $\sigma_Y^2 = DY$ и $\sigma_{XY} = \mathrm{Cov}(X,Y)$.

Нама (инвеститору) је циљ да се домогне овог α. Наизглед је све ту: куд ће боље већ готова формула. Међутим, проблем је тај што су величине σ_X^2, σ_Y^2 и σ_{XY} у начелу непознате, те мора да се користи оцена

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}.$$

Хајде да претпоставимо да је позната расподела за X и Y, као и (X,Y). Тада можемо да урадимо следеће:

1. Генеришемо узорак обима n за обе расподеле;
2. Из тих узорака оценимо све неопходне параметре;
3. Поновимо кораке 1. и 2. N пута (велики број);
4. За сваки од три параметра од интереса, назовимо га привремено p, добили смо низ оцена $p_1, p_2, \ldots, p_N$. За ове оцене цртамо хистограм, те узимамо просек (или медијану, моду итд) за оцену параметра.

Ове класичне Монте Карло симулације дају изванредне резултате у случајевима када знамо (макар приближно) расподелу обележја од интереса. **Оно што ми у пракси имамо јесте низ прошлих вредности за X и Y.** Стога ми морамо на основу доступних узорака, који су коначног обима, без могућности генерисања нових, да оценимо параметре.

Идеја бутстреп алгоритма је у томе да се **из доступних узорака ваде узорци.** И то не било какви, већ са понављањем, истог обима као почетни узорак, како бисмо добили што већи варијабилитет извучених узорака. Уколико имамо скуп података Z, применом бутстреп узорковања R пута добићемо R узорака са понављањем истог обима као и Z. Нека су то $Z_1^*, Z_2^*, \ldots, Z_R^*$. На основу њих правимо низ

оцена за α: $\hat{\alpha}_1^*, \ldots, \hat{\alpha}_R^*$. Оцену за α стандардно рачунамо као просек свих ових оцена, а оцену дисперзије као узорачку дисперзију:

$$\widehat{D\hat{\alpha}} = \frac{1}{R-1} \sum_{i=1}^{R} \left(\hat{\alpha}_i^* - \frac{1}{B} \sum_{j=1}^{R} \hat{\alpha}_j^* \right)^2 .$$

Добро, то би било то што се тиче теоријског дела. Да видимо како то сада да испрограмирамо у R-у.

Процес спровођења бутстрепа у R-у одвија се у два корака. Прво дефинишемо функцију која рачуна параметар (оцену) од интереса. Затим користимо функцију `boot` из истоименог пакета (истог оног у коме постоји лоша функција `cv.glm`, али ова је добра) која врши узроковање са понављањем и комбинује оцене.

У пакету `ISLR2` постоји база података `Portfolio` у којој су вештачки генерисани подаци обима 100 који представљају принос два асета које смо малопре поменули. За почетак правимо функцију која рачуна оцену параметра . Њој прослеђујемо базу, као и вектор индекса на основу којих се врши оцењивање.

```
alfa_fja <- function(data, index) {
  X <- data$X[index]
  Y <- data$Y[index]
  (var (Y) - cov (X , Y)) / (var (X) + var (Y) - 2 * cov (X , Y))
}
```

Функција `boot` као аргумент прима базу података (`data.frame`), функцију за рачунање оцене, као и аргумент `R` који говори о томе колико пута се врши узорковање.

```
library(boot)
```

```
boot(Portfolio, alfa_fja, R = 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Portfolio, statistic = alfa_fja, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.5758321 0.001006499  0.08978204
```

Повратне вредности су јасне по називу.

Вежбе 5

Увод у невођено учење: Кластеризација

Напомена. Неке од слика сам преузео из скрипте др Младена Николића и Анђелке Ковачевић, али сам те исте слике налазио по интернету, те им је тешко ући у траг, јер су их и они однекле преузели.

До сада смо се сретали само са примерима вођеног учења. То је значило да су наши подаци били означени (енг. *labeled*), тј. за дате податке знали смо вредност зависне променљиве Y. На основу познатог узорка правили смо модел, који смо после користили на новим подацима.

Код невођеног учења не постоји зависна променљива Y. Просто су нам дати подаци и ми сами треба да одлучимо шта са њима да радимо. Овде се, поред непостојања зависне променљиве, отварају и многи други проблеми:

- Како проверити тачност добијених резултата, када немамо са чиме да их упоредимо?
- Како извршити поделу скупа на валидацију и тест?
- У зависности од циља, постоји ли више (и колико тачних решења)?

Једна од основних грана невођеног учења јесте кластеризација (кластеровање).

5.1 Кластеризација: теорија

Кластеризација представља процес груписања података у групе (тзв. кластере) на тај начин да су подаци унутар кластера међусобно слични, а кластери међусобно што различитији.

ОПРЕЗ! Наш појам кластера није исти као из теорије узорака! Оно што се у машинском учењу зове кластер, у теорији узорака звало би се стратум.

Потребе за кластеризацијом су многе, а неке од њих укључују издвајање група на друштвеним мрежама зарад циљаног оглашавања, детекција малигног и бенигног ткива на рендгенским снимцима, раздвајање људи у групе на основу разних психолошких тестова итд. Такође, кластеризација се може користити у сврхе **претпроцесирања** података, тако што у првом кораку обраде података кластери бивају замењени својим представницима. Ово може бити корисно у смислу рачунске једноставности, али у зависности од ситуације може и смањити квалитет модела.

Такође, кластеризација је корисна и за **кросвалидацију**. Наиме, веома је корисно поделити податке на кластере, те затим сваки од кластера на k једнаких делова, те фолдове за k-fold правити тако

што се бира по један из сваког кластера. Тиме су фолдови учињени репрезентативнијим него што би потенцијално били случајним одабиром.

Треба напоменути и да кластеризација није јединствена, јер увек можемо разматрати грубље, односно финије поделе. Већина алгоритама за кластеризацију омогућава подешавање („штеловање'') финоће самог процеса.

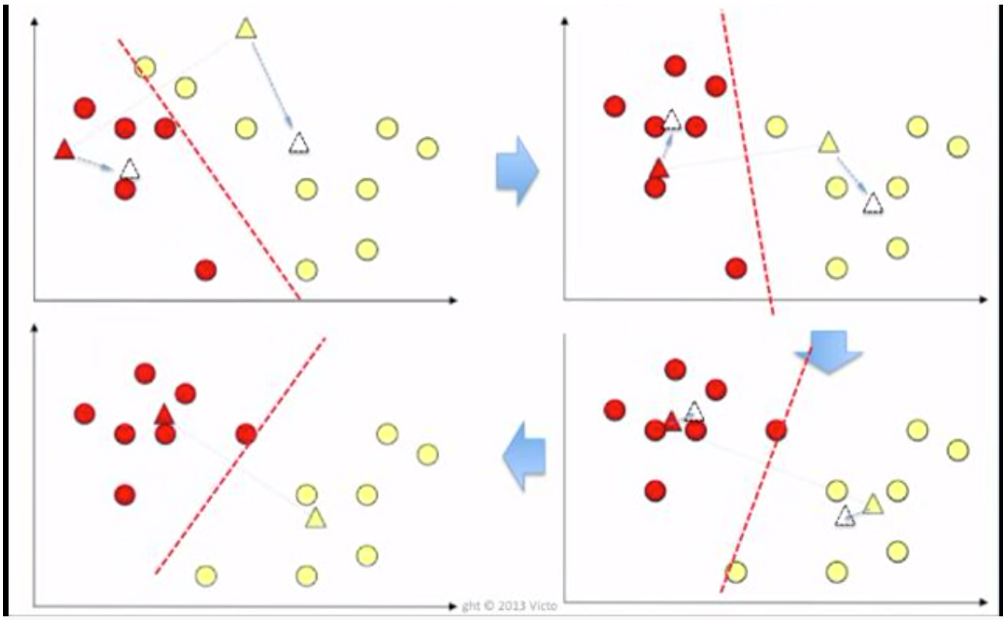
У вези са претходним разликујемо и различите врсте кластера: *Глобуларни* или *центрички* кластери јесу они који попуњавају унутрашњост лопте или евентуално елипсоида (у одговарајућем броју димензија, не мора три). Кластери су *добро раздвојени* уколико је свака тачка неког кластера ближа свим тачкама свог кластера неголи било којој тачки неког другог кластера. Кластери су *хијерархијски*, уколико у оквиру кластера можемо уочавати кластере.

5.1.1 *K*-means кластеризација

Један од најједноставнијих типова кластеризације јесте K-means кластеризација, или, на српском, кластеризација K-средина. Сада ћемо изложити како алгоритам ради.

За почетак, бира се број кластера, K. Очекује се да овај број буде процењен на основу претходних сазнања о подацима. Бира се K такозваних *центроида*, односно центара кластера, на случајан начин. Ако се зна нешто више о подацима, овај одабир свакако не мора бити случајан, и пожељније је да не буде. Затим се понављају следећи кораци:

1. За сваку опсервацију рачунамо којој је центроиди најближа. Када срачунамо, доделимо јој кластер који одговара тој центроиди.
2. Рачунамо центар новодобијеног кластера и у њега премештамо центроиду.



Претходна два корака понављамо све док процес не исконвергира. У вези са K-means кластеризацијом треба напоменути пар ствари:

- **Одабир метрике**. Особина „бити најближа центроиди'' коју има нека опсервација зависи од одабира метрике. За нумеричка обележја, јер таква морају да буду пошто рачунамо просеке, је то углавном еуклидска метрика, или, евентуално, нека од l^p метрика. За одабир еуклидске

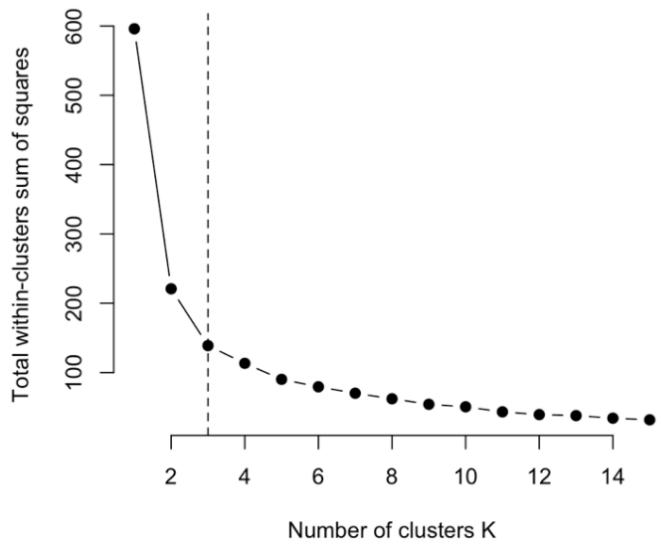
метрике d, може се показати да k-means кластеризација заправо минимизује функцију

$$\sum_{i=1}^{K} \sum_{x \in C_i} d(x, c_i)^2,$$

где је C_i кластер а c_i центроида. Минимизација наравно иде по центроидама.

Како је у овом случају алгоритам заснован на минимизовању еуклдских растојања, он тежи проналаску кластера у облику лопте.

- **Кластеровање, макар и за фиксно K, не мора да буде јединствено.** На пример, довољно је да имамо податке униформно распоређене унутар круга и да их треба поделити на два кластера. Такође, горња сума може имати локалне минимуме, који су већи од глобалног, и алгоритам у њима може да „заглави". Због тога се у пракси кластеризација понавља више пута, са различитим почетним одабиром центроида, те се гледа шта буде у највећем броју случајева.
- **Одабир броја K.** Веома често није јасно колико кластера се очекује, те се не зна које K проследити алгоритму. Једно од правила јесте и *„правило лакта"*. Оно подразумева да се за различите вредности броја K изврши кластеризација, те се у сваком од тих случајева срачуна унутаркластерна сума квадрата растојања, а затим се нацрта зависност збира тих сума у односу на K, као на слици испод.



Бирамо оно K које је „на лакту", јер након њега видимо заравњење у суми квадрата, што значи да је додавање нових кластера скоро па никако не смањује.

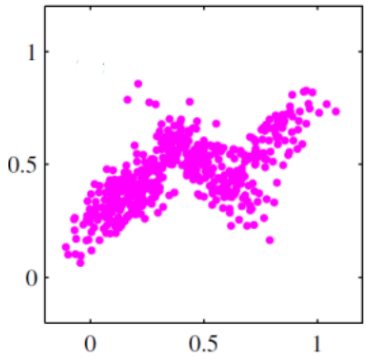
5.1.2 Мешавина нормалних расподела и EM алгоритам

Предност алгоритма K-means свакако јесте његова једноставност. Међутим, појављују се и многи проблеми:

1. Свака опсервација била је додељена тачно једном кластеру. Међутим, постоје опсервације које су „на граници", и чије додељивање кластеру јесте толико несигурно, да се може сматрати и случајним погађањем. Желели бисмо да некако за такве опсервације **сачувамо неодређеност.**
2. Користи еуклидско растојање. Чак иако одаберемо неку од других l^p норми, облик кластера никад неће бити, рецимо, елипса. Шта ако кластер није кружан, већ елиптичан?

3. Након извршене кластеризације, исти је однос према подацима близу и далеко од центроиде кластера. А за ове ближе смо сигурнији.

4. Шта ако се кластери преклапају, као на слици испод?



Због тога ћемо изложити мало софистицираниј верзију K-means алгоритма, која је позната под називом Мешавина нормалних расподела, у комбинацији са EM алгоритмом. Користићемо скраћеницу GMM, од енглеског *Gaussian Mixture Model.* EM алгоритам би требало да је, као концепт, познат са Математичке статстике, те му овде нећемо доказивати теоријска својства, већ ћемо их само наводити по потреби.

Идеја GMM алгоритма јесте да се опсервацијама, уместо кластера коме припадају, додели **вероватноћа** припадања сваком од кластера. За почетак, треба дефинисати број кластера, C, што и овде радимо „од ока'' или на основу неких сазнања о подацима „са стране''. Рецимо да имамо n опсервација и d предиктора. Ако погледамо на слику изнад, делује нам да су подаци дати у виду три дводимензионе нормалне расподеле. Ако се подсетимо, густина вишедимензионе нормалне расподеле зависи од вектора просека μ и од коваријационе матрице Σ, а дата је формулом:

$$f_{\mu,\Sigma}(\mathbf{x}) = \frac{|\Sigma|^{-1/2}}{(2\pi)^{d/2}} \exp\left(-(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right).$$

Сада је природно задати густину свих наших података као неку **конвексну** комбинацију густина појединачних кластера:

$$f(\mathbf{x}) = \sum_{c=1}^{C} \pi_c f_{\mu_c,\Sigma_c}(\mathbf{x}).$$

Идеја је да се крене од произвољних параметара π_c, μ_c и Σ_c, те да се они итеративно „побољшавају''. Једну илустрацију густине f можемо видети на слици испод.

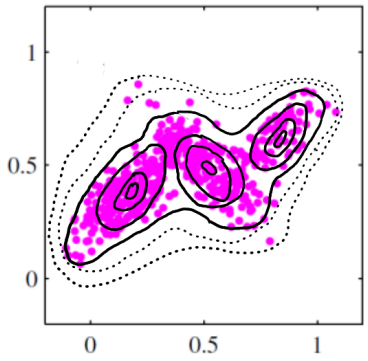
Густина f може се схватити и на следећи начин. Можемо уочити да заправо постоји једна скривена, **латентна** случајна величина Z, која заправо представља расподелу вероватноћа π_j по кластерима:

$$Z : \begin{pmatrix} c_1 & c_2 & \cdots & c_C \\ \pi_1 & \pi_2 & \cdots & \pi_C \end{pmatrix}.$$

Сада се може записати да је

$$f(\mathbf{x} \mid Z = c) = f_{\mu_c,\Sigma_c}(\mathbf{x}).$$

Ово је згодан начин за представљање, уколико бисмо желели да генеришемо опсервацију са густином f, јер је $f(\cdot) = f(\cdot \mid Z = c)\pi_c$, а све на десној страни знамо да генеришемо.



Океј, вратимо се GMM алгоритму. Крећемо од произвољног (или не, ако знамо нешто) одабира почетних елемената μ_c, Σ_c, π_c. Затим спроводимо следеће кораке:

- **E корак.** За сваку опсервацију $\mathbf{x}^i$ рачунамо вероватноћу да припада c-том кластеру као:

$$r_{ic} = \frac{\pi_c f_{\mu_c, \Sigma_c}(\mathbf{x}^i)}{\sum_{c'=1}^{C} \pi_{c'} f_{\mu_{c'}, \Sigma_{c'}}(\mathbf{x}^i)}.$$

- **M корак.** На основу срачунате вероватноће припадања, ажурирамо информације о кластерима. Прво ћемо да рачунамо n_c које представља суму свих вероватноћа припадања за c-ти кластер:

$$n_c = \sum_{i=1}^{n} r_{ic}.$$

  Ажурирану вредност за π_c сада добијамо као количник n_c и n:

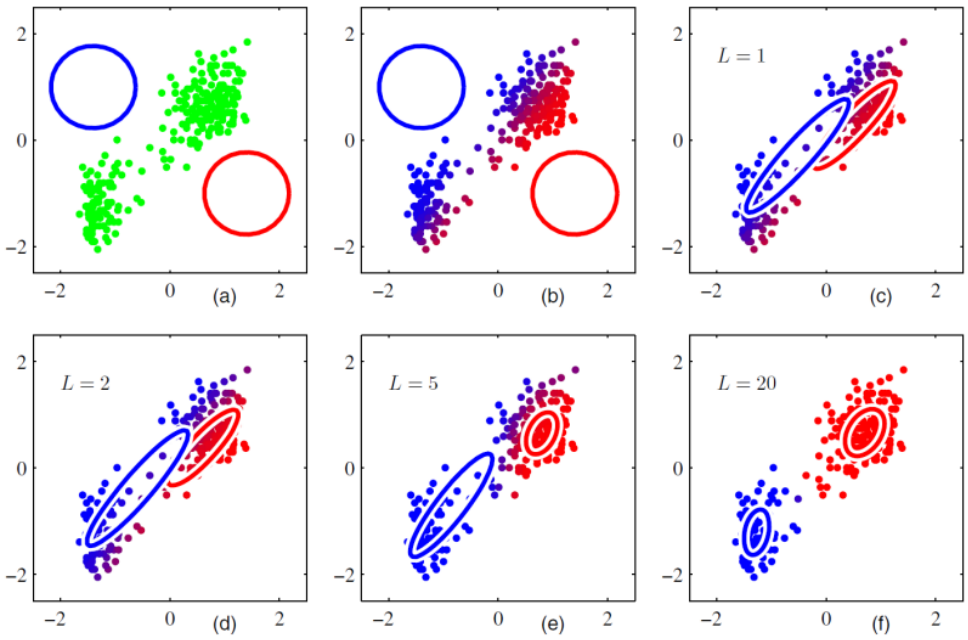
$$\pi_c = \frac{n_c}{n}.$$

  Она се ажурира на овај начин јер у неком смислу представља „тежину" c-тог кластера у заједничкој суми. Средњу вредност и коваријациону матрицу сада, природно, рачунамо као тежинске средине:

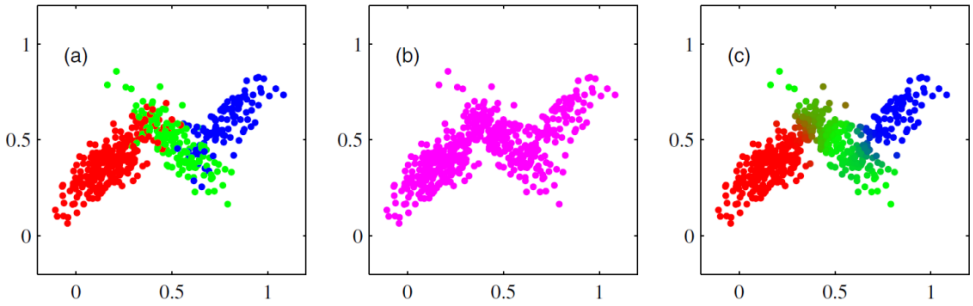
$$\mu_c = \sum_{i=1}^{n} \frac{r_{ic}}{n_c} \mathbf{x}^i, \quad \Sigma_c = \sum_{i=1}^{n} \frac{r_{ic}}{n_c} (\mathbf{x}^i - \mu_c)(\mathbf{x}^i - \mu_c)^T.$$

Напомена. Може се доказати (доказ у [23]) да свака итерација повећава функцију веродостојности, те овај алгоритам сигурно конвергира. Међутим, он може исконвергирати локалном, а не глобалном максимуму. Стога се и овде препоручује „пуштање'' алгоритма више пута, са различитим почетним параметрима.

Илустрација GMM+EM за два кластера и 20 итерација дата је на слици испод:



На наредној слици, скроз лево приказани су подаци, вештачки генерисани, онаквима какви јесу. На средњој слици видимо их обједињене. На десној слици, интензитет боје сваке тачке сразмеран је броју r_{ic}. Видимо да је на преклапању кластера боја најблеђа, и долази до преклапања боја, што осликава нашу несигурност око одабира кластера за те тачке.



Уколико ипак желимо да опсервацију доделимо неком кластеру, а не да је оставимо са вероватноћом, просто ћемо одабрати онај кластер чија је вероватноћа припадности највећа. Уколико добијемо **нову опсервацију** поступамо идентично.

За одабир броја кластера и овде важи „лакат правило''.

Уочимо још и да је k-means алгоритам заправо специјалан случај алгоритма GMM и то за

- $\Sigma_c = \sigma^2 I$ за све c;
- $\pi_c = 1/C$, за све c;
- $P(Z = c \mid \mathbf{x}^i)$ је 1 ако је $\mathbf{x}^i$ најближи просеку μ_c, а нула иначе.

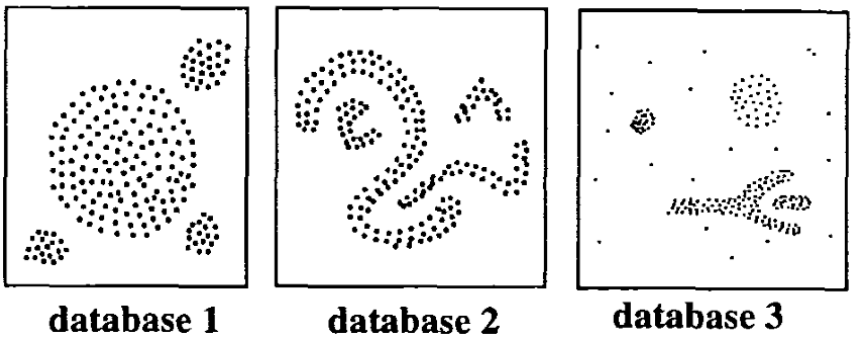
5.1.3 DBSCAN кластеризација

Напомена. Део слика преузет је из [3].

Следећи тип кластеризације са којим ћемо се упознати јесте алгоритам DBSCAN. Име му је акроним и скраћеница је од енглеског *Density-Based Spatial Clustering and Application with Noise*. Пре него што дамо детаљан опис алгоритма, мотивисаћемо.

Код алгоритма K-means успевали смо да издвајамо кластере правилног(симетричног) облика. За еуклидску метрику то су били кружни кластери, док би за неки други одабир метрике то био неки други правилан облик. Коришћењем GMM+EM успели смо да кластере које смо у стању да уочимо проширимо на елиптичне кластере.

Међутим, размотримо наредну слику.



У сва три случаја јасно су, људском оку, уочљиви кластери. Међутим, јасно је да свака од поменуте две методе неће бити у стању да их препозна на адекватан начин, јер сами клсатери нису одговарајућег облика (осим на првој слици).

Идеја DBSCAN алгоритма јесте да кластере замишљамо као области велике густине, а да их раздвајамо областима мање густине. Два основна параметра у вези са овим јесу

- ε (епсилон) - параметар који дефинише полупречник, такав да за неку тачку x кажемо да је лопта $B(x, \varepsilon)$ њен „комшилук", а да свака већа није њен комшилук.

  Напомена! На енглеском језику, појам се зове *neighbourhood*, што би се на српски превело као *околина*. Међутим, како овај појам не одговара тополошком појму околине, одлучио сам се да уведем термин комшилук.

- z - минималан број тачака у околини неке тачке - њених „комшија".

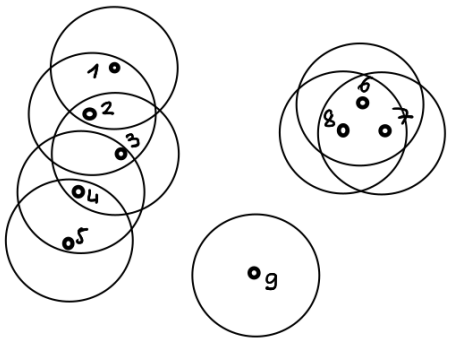
Сада ћемо дефинисати неколико појмова.

- Опсервација **x** са бројем комшија барем z (рачунајући и њу) је **главна тачка** (енг. *core point*).
- Ако је **x** таква да има мање од z комшшија, али се налази у комшилуку неке главне тачке, кажемо да је она **гранична тачка** (енг. *border point*).
- Ако **x** није ни главна ни гранична тачка, кажемо да је **тачка шума** (*noise point*).

На наредној слици дата је једна илустрација поменутих типова тачака.

Сада смо корак ближе томе да извршимо кластеризацију. Треба нам још појмова.

- Тачка (опсервација) A је **директно дохватљива** (*direct density reachable*) из тачке B ако је A у комшилуку од B и ако је B главна тачка.



$z = 3$

2, 3, 4, 6, 7, 8 - главне

1, 5 - граничне

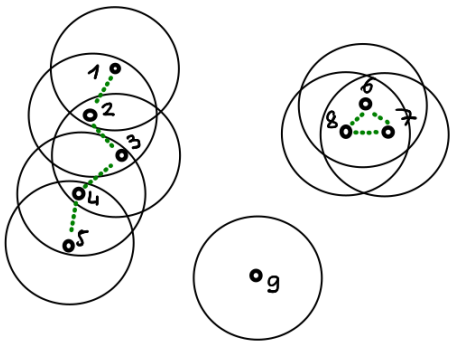
9 - шум

- Тачка A је **дохватљива** (*density reachable*) из тачке B ако постоји низ главних тачака који води од B до A.
- Тачке A и B су **повезане** (*density connected*) ако постоји главна тачка C таква да су обе из ње дохватљиве.

Сада смо коначно спремни да опишемо DBSCAN алгоритам.

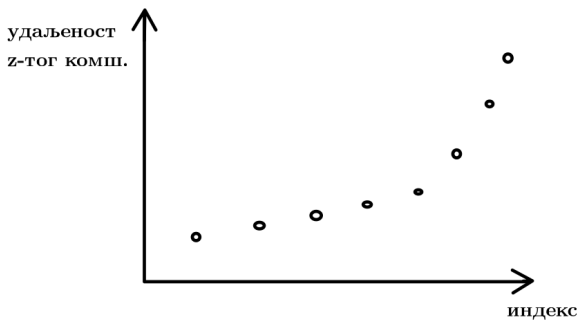
1. За сваку тачку $\mathbf{x}^i$ рачунамо њено растојање од осталих тачака. Уочавамо све тачке које су јој комшије. Сваку која има више од z комшија прогласимо за главну тачку.
2. За сваку главну тачку, ако већ није у кластеру, правимо нови кластер. У њен кластер смештамо све са њом повезане тачке.
3. Понављамо процес кроз све тачке.

Она тачка која није главна и није припала ниједном кластеру, јесте тачка шума и њу не класификујемо. На наредној слици видимо наставак примера са претходне слике. За дато $z = 3$ и одговарајуће ε алгоритам је уочио два кластера и једну тачку шума.



Неке од главних предности овог алгоритма јесу то што не захтева да му се предефинише број кластера, већ га он сам проналази. Овај алгоритам проналази кластере произвољног облика, а у стању је и да детектује аутлајере (тачке шума).

Основни недостатак јесте тај што **није јасно како одабрати** ε **и** z. Наравно да ни овде не постоји генерално правило. Пракса је показала да, уколико имамо велики број опсервација, можемо бирати z које је ред величине броја предиктора. За ε прича иде мало теже. Овде је пракса показала да можемо за сваку опсервацију срачунати њену удаљеност од z-тог комшије, затим сортирати тај низ удаљености. Идеално, добићемо график сличан оном на наредној слици.



Бирамо удаљеност оног z-тог комшије који је „на лакту''. Ако немамо „лакат", наравно, онда ништа, морамо на осећај.

5.2 Кластеризација - практично у R-у

5.2.1 K-means

Користићемо функцију `kmeans` из основног пакета `stats`. Користићемо базу података `USArrests`. Свака опсервација је једна Америчка савезна држава, а предиктора има 4: стопа убистава, стопа напада, стопа урбане популације и стопа силовања.

```
data("USArrests")
```

Одмах треба скалирати податке, јер желимо да сваки предиктор има исти ред величине, због рачунања растојања.

```
df <- data.frame(scale(USArrests))
```

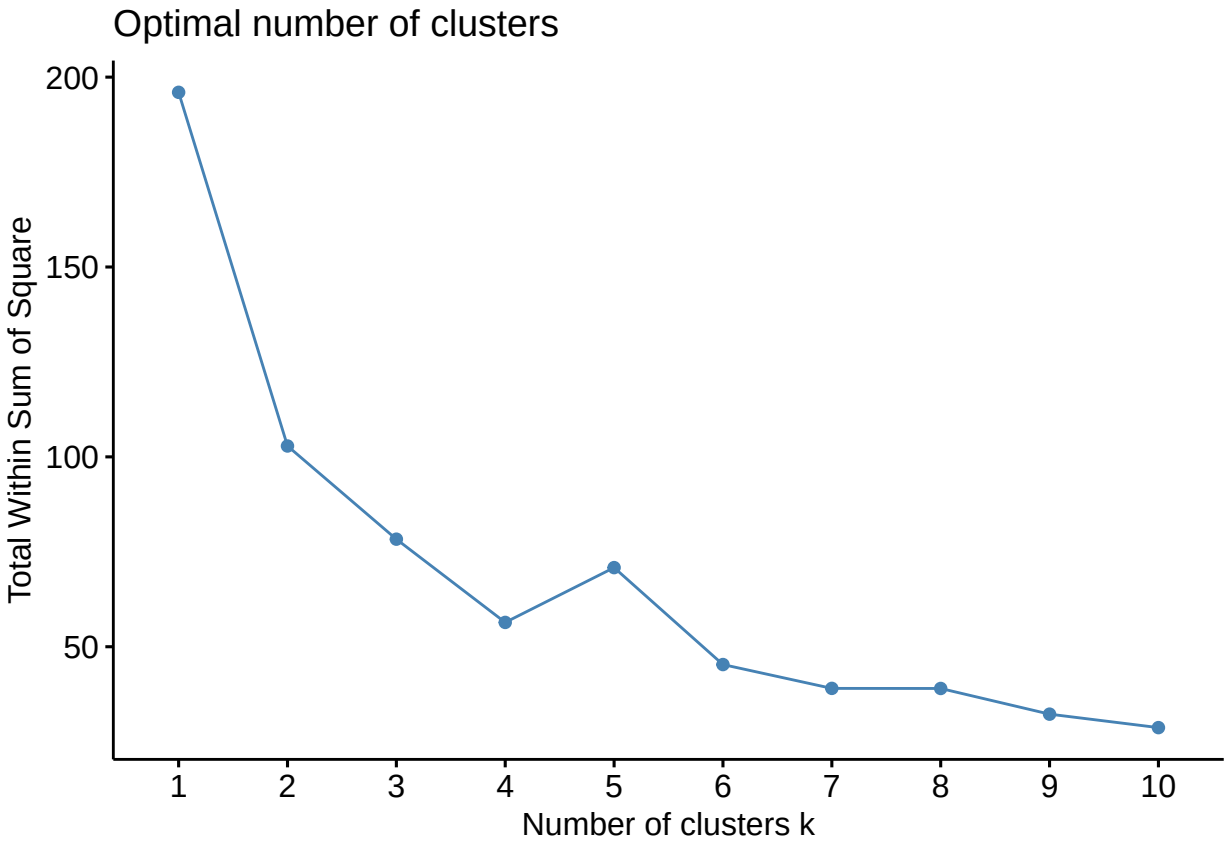
Потпис функције `kmeans` је следећи

$$\texttt{kmeans}(\texttt{x}, \texttt{centers}, \texttt{iter.max} = 10, \texttt{nstart} = 1).$$

Аргумент `x` представља базу података. Аргумент `centers` представља почетни одабир центроида, а ако му се проследи само један број онда то схвата као број кластера, а почетне центроиде бира насумично из врста од `x`. Аргумент `iter.max`, јасно, представља максималан дозвољен број итерација алгоритма, док последњи аргумент представља број почетних одабира центроида. Ако је већи од један, онда се бира она кластеризација која минимизује суму квадрата растојања од центроида, у свим кластерима ([4]). Пожељно је да овај аргумент буде већи од 1.

Користићемо и библиотеку `factoextra` због лепих графичких приказа. За почетак, треба одредити оптималан број кластера „правилом лакта''.

```
library(factoextra)
fviz_nbclust(x = df,
             method = "wss", # "within sum of squares", pogledati help za ostale)
             FUNcluster = kmeans
            )
```



Видимо да сума квадрата достиже минимум за 4 кластера, па расте за 5, те наставља да опада након 6 кластера. Овде није јасно да ли узети 4 или 6 кластера. Без неког нарочитог разлога одлучићемо се за 4 кластера, јер нам је циљ да демонстрирамо метод.

Сада спроводимо кластеризацију.

```
set.seed(123)
km_result <- kmeans(df, 4, nstart = 15)
```

То је у суштини то, још да видимо шта враћа функција `kmeans`. Она враћа листу компоненти, а у њој је

- `cluster` - вектор природних бројева дужине оригиналних података, који говори о томе ком је кластеру опсервација додељена;
- `centers` - финални вектор центроида;
- `totss` - SSTO са ЛСМ; мери укупну суму квадрата растојања свих података;
- `withinss` - вектор унутаркластерних сума квадрата растојања;
- `tot.withinss` - сума претходног вектора;
- `betweenss` - једнака `totss` - `tot.withinss`;
- `size` - вектор величина кластера.

Визуализацију ћемо оставити за након што одрадимо метод PCA.

`GMM+EM`

Пошто смо у теоријском делу видели како алгоритам ради, трудићемо се да што мање ствари програмирамо ручно, већ да користимо уграђене функције. Користимо пакет `mclust`.

```
library(mclust)
```

База коју ћемо користити јесте база `iris`, која садржи податке о цвету рода Ирис. Она има следеће колоне

```
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

Прве 4 говоре о вредностима одређених димензија цвета, док последња представља једну од три врсте у оквиру рода Ирис, а то су:

```
levels(iris$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

Последњу нећемо користити јер представља баш кластере, које ми желимо да одредимо невођено, те је се решавамо.

```
df <- iris[, 1:4]
```

Сада све иде аутоматски, позивом функције `Mclust`. Сви параметри могу се и ручно иницијализовати, али ми ћемо оставити функцији да ради сама, битно је само да знамо да може.

```
mcl_model_iris <- Mclust(df, 3)
```

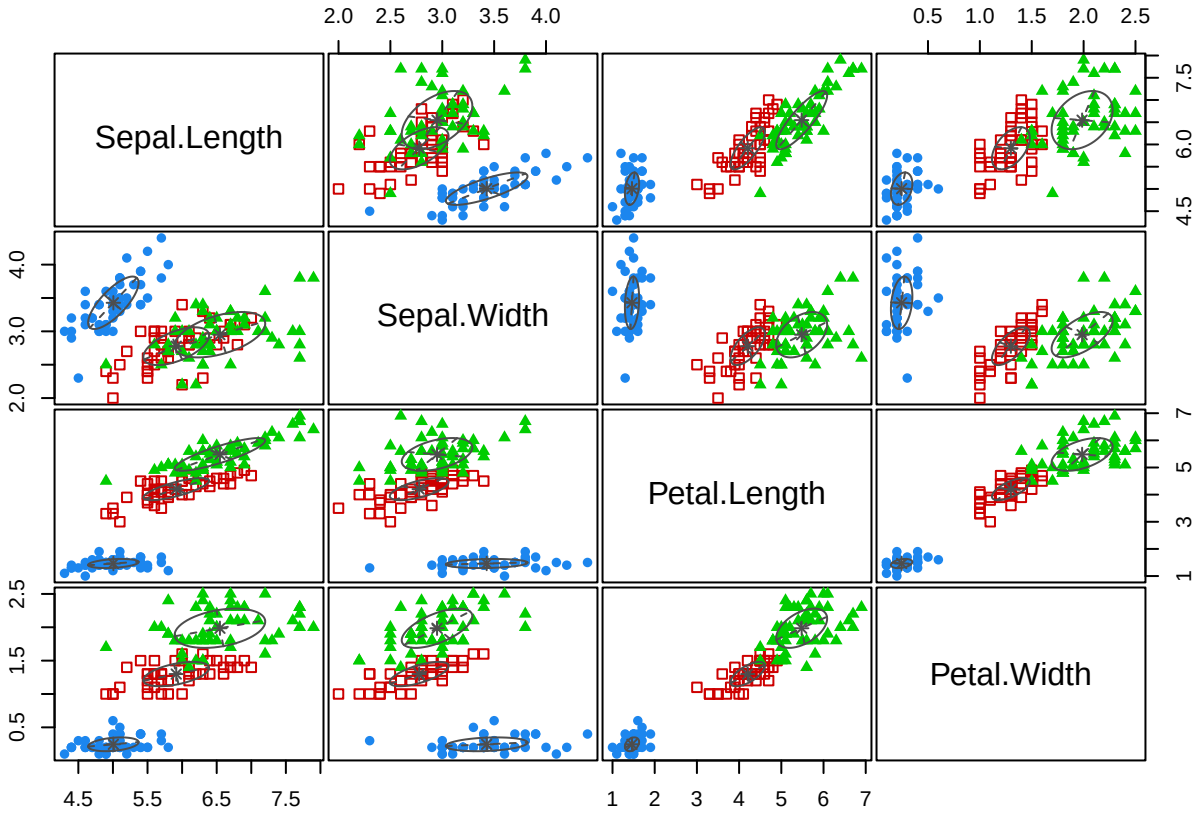
Ако позовемо `class`, добијамо

```
class(mcl_model_iris)
```

```
## [1] "Mclust"
```

Ово је заправо листа неких стандардних повратних вредности, свих оних о којима смо говорили у теоријском уводу, те се нећемо замарати свима, за то постоји `help`. Оно што је згодно јесте то што се објекат ове класе може директно проследити функцији `plot`.

```
plot(mcl_model_iris, what = "classification")
```



Јасно, на овој слици су кластери додељени по принципу највеће вероватноће, па нема флуидног прелаза боја. То се може добити играњем са `ggplot`-ом и повратним вредностима за `Mclust`.

5.2.2 DBSCAN

Овај алгоритам у R-у може бити имплементиран коришћењем два пакета: `dbscan` и `fpc`. Ми ћемо се одлучити за овај други за саму кластеризацију, док ћемо за одабир епсилона користити први. Разлика је мање-више у синтакси. Користићемо и пакет `factoextra` због визуализације кластера, уграђене базе `multishapes`, а `fpc` смо бирали између осталог јер се слаже са овим пакетом у смилсу визуализације. Прецизније, оно што враћа функција из пакета `fpc` можемо прослеђивати функцијама из `factoextra` као аргумент, док за `dbscan` то не можемо.

```
library(fpc)
library(factoextra)
library(dbscan)
```

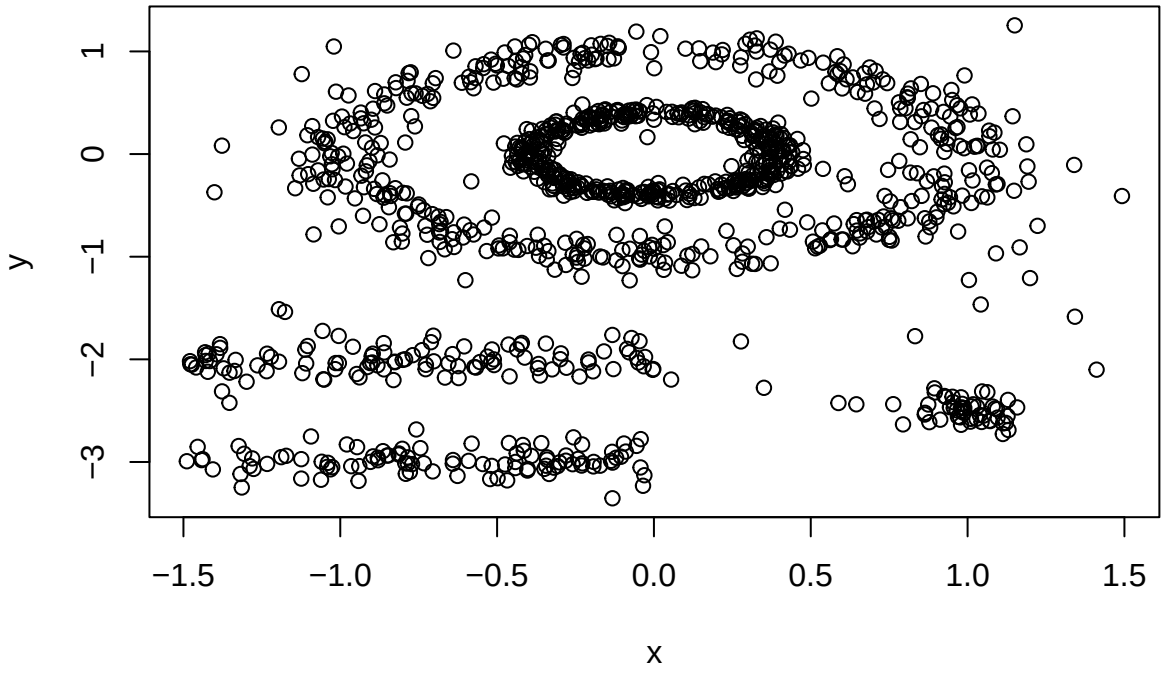
Напомена за кориснике ГНУ/Линукс дистрибуција! За инсталацију ових пакета морате имати инсталиран `gcc-fortran`.

Прво ћемо учитати податке.

```
data("multishapes", package = "factoextra")
df <- multishapes[, 1:2]
```

Узели смо прве две промењиве јер је трећа категоричка и говори о томе која је права вредност кластера у који опсервација спада (подаци су вештачки генерисани па се зна).

```
plot(df)
```



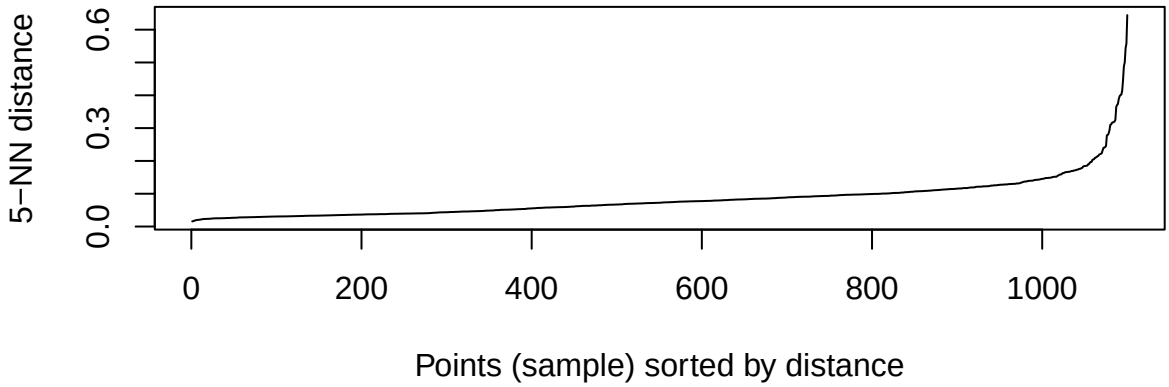
Јасно је да уочавамо 5 кластера. Након кластеровања функцијом из пакета `factoextra` направићемо лепшу слику. Сада вршимо кластеризацију помоћу пакета `fpc`.

```
set.seed(123)
db <- fpc::dbscan(df, eps = 0.15, MinPts = 5)
# class(db) враца "dbscan"
```

Број z дат је аргументом `MinPts`. Бирали смо 5, иако имамо 2 предиктора, јер бисмо спољни „прстен'' желели да класификујемо као један кластер, па за сваки случај, да не направи прекид.

Епсилон смо одабрали уз помоћ „лакат'' технике, а то се спроводи на следећи начин.

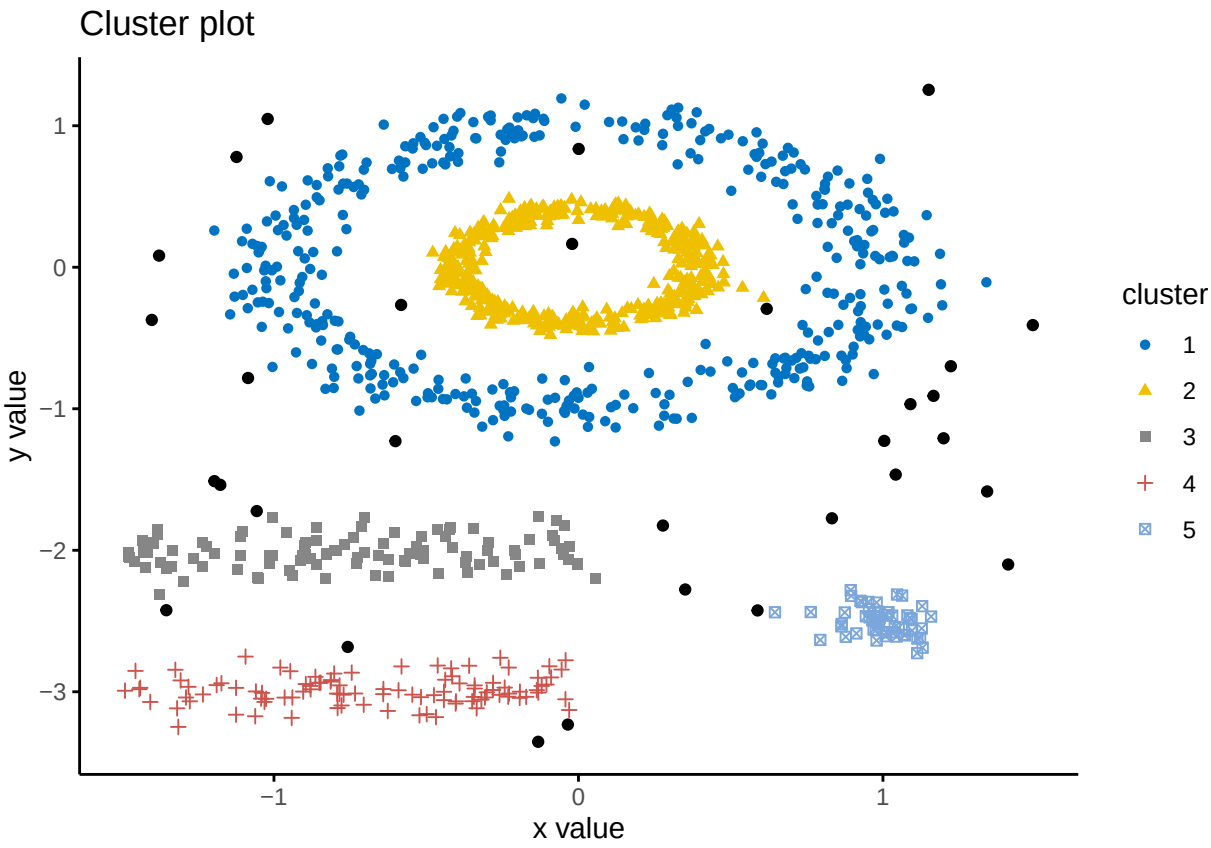
```
dbscan::kNNdistplot(df, k = 5)
```



Видимо да је 0.15 отприлике „на лакту''.

Сада цртамо лепшу слику.

```
fviz_cluster(db,
             data = df,
             stand = FALSE, # ne radimo PCA, ne treba nam (zvati help)
             ellipse = FALSE, # da ne zaokruzuje klastere
             show.clust.cent = FALSE,
             geom = "point",
             palette = "jco",
             ggtheme = theme_classic()
             )
```

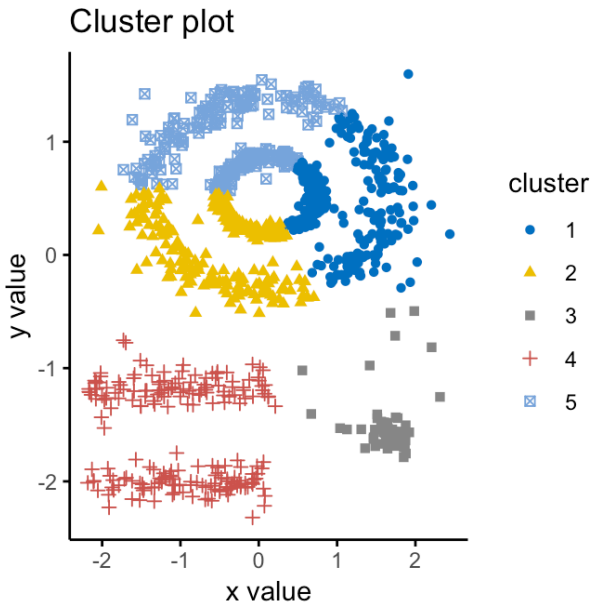


И голим оком да се наслутити да је већину аутлајера погодио како треба. Тек за неке нам је смислено да припадају неком кластеру, а да их је алгоритам прогласио аутлајерима (шумом).

Предвиђање за нове тачке врши се функцијом `predict.dbscan` из пакета `fpc`.

На наредној слици дат је пример шта се деси када исте податке покушамо да кластерујемо методом K-means, за одређене почетне центроиде (нећемо спроводити поступак, само као пример).

Није баш како треба.


Cluster plot
y value
x value
1
0
-1
-2
-2
-1
0
1
2
cluster
1
2
3
4
5

Вежбе 6

Редукција димензионалности

Претпоставимо да желимо да визуализујемо n тачака али имамо p колона: $X_1, \dots, X_p$. Једини начин који смо до сада имали за ту визуализацију јесте био да правимо графике свих могућих парова променљивих. Међутим, ту се појављују два велика проблема:

- Имамо $\binom{p}{2}$ графика; за велико p то је јако велики број графика који је тежак за праћење;
- Губи се јако велика количина информације на овај начин.

Због тога нам треба метода која то све представља информативније. Једна од таквих метода јесте и метода PCA.

6.1 PCA

PCA (*Principal Component Analysis*), или, на српском, Анализа главних компоненти, јесте једна од најпознатијих метода за рад са проблемом димензионалности. Како је она есенцијална за анализу података било ког типа, обрадићемо је детаљно.

Нека имамо $n \times p$ матрицу X у којој се налазе наши предиктори. Претпоставимо да **су подаци центрирани**, то јест да је узорачка средња вредност сваке колоне једнака нули.

Напомена. PCA претпоставља да нема категоричких предиктора. Ако су они присутни треба их привремено склонити, те радити PCA на осталим предикторима, па онда категоричке додати назад.

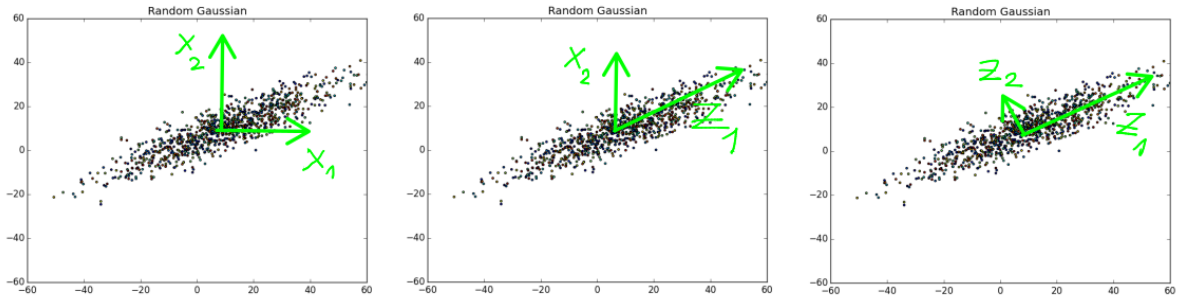
Идеја је да извршимо трансформације предиктора $X_1, \dots, X_p$ облика

$$Z_k = \sum_{i=1}^{p} a_{ki} X_i = \mathbf{a}_k^T X$$

такве да је $DZ_k = \mathbf{a}_k^T \mathbf{Cov}(X) \mathbf{a}_k$ што веће. Да бисмо имали оптимизациони проблем тражења условног екстремума, додаје се и услов $\sum_i a_{ki}^2 = 1$,а ово заправо значи да је трансформација задата матрицом $A = [a_{kl}]$ једна ортогонална трансформација.

Решавањем горњег оптимизационог проблема за $k = 1$ налазимо коефицијенте $a_{11}, \dots, a_{1p}$, па самим тим и вектор Z_1 који зовемо **прва главна компонента**. На наредној слици је илустрација.

Овиме смо пронашли правац највеће распршености наших података. Следећи вектор који тражимо, Z_2 треба да одговара следећем правцу највеће распршености, после првог. Овај вектор зовемо **друга главна компонента**. Њега тражимо тако што стављамо услов $\|\mathbf{a}_2\|_2 = 1$, тражимо да је он ортогоналан



на претходну главну компоненту (са слике јасно зашто), дакле $\mathbf{a}_2^T X \perp Z_1$ и наравно тражимо да је $\mathbf{a}_2^T \mathbf{Cov}(X)\mathbf{a}_2$ што веће. Поступак понављамо итеративно.

Може се показати (ЛСМ) да је прва главна компонента заправо сопствени вектор који одговара највећој сопственој вредности матрице $\mathbf{Cov}(X)$.

Напомена. Подсетимо се, коваријациона матрица је ненегативно дефинитна, те има ненегативне сопствене вредности.

Након тога, уместо „одокативног'' одбацивања предиктора можемо просто задржати првих неколико главних компоненти. Пракса је да се задржи онај број који објашњава барем 80 посто варијабилитета у подацима. Детаљно извођење свега у вези са PCA биће дато на ЛСМ, укључујући и коваријациону матрицу главних компоненти, те и теоријски губитак који имамо одбацивањем последњих l комада.

Основна мана овог алгоритма јесте та што подаци губе на интерпретабилности. Сваки од оригиналних предиктора имао је неко значење: висину, масу, годиште итд, док ови нови, који су њихове линеарне комбинације, не значе ништа. То је цена коју смо морали да платимо.

6.1.1 PCA - пример

Декомпозиција матрице у сврху PCA у `R`-у може да буде спектрална, и тако је имплементирана у функцији `princomp`, док је у функцијама `prcomp` и `PCA` из `FactoMineR` имплементирана сингуларна декомпозиција коваријационе матрице. У суштини је свеједно: ова друга даје мало боље резултате у пракси, те ћемо се ми одлучити за `prcomp`.

Напомена. Ова функција има аргумент `scale` али је он подразумевано `FALSE`! Податке треба или ручно скалирати пре позива ове функције, или аргумент поставити на `TRUE`.

За базу података одлучићемо се за већ познату `USArrests`.

```
data("USArrests")
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

Свака врста је једна америчка савезна држава, а колоне говоре саме за себе.

Следећа ствар коју радимо јесте да скалирамо наше податке.

```
scaled_df <- apply(USArrests, 2, scale)
```

Сада ћемо извршити PCA над овом базом.

```
pca_result <- prcomp(scaled_df)
```

Да видимо шта нам је функција вратила.

```
class(pca_result)
```

```
## [1] "prcomp"
```

Ова класа је заправо листа повратних вредности, а њихова имена можемо добити као:

```
names(pca_result)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

Променљиве `center` и `scale` одговарају средњим вредностима и стандардним девијацијама колона које су постојале пре скалирања (код нас 0 и 1, јер смо скалирали ручно). У то се можемо уверити позивом

```
pca_result$center
```

```
##        Murder       Assault      UrbanPop          Rape
## -7.667478e-17  1.111611e-16 -4.332645e-16  8.938163e-17
```

Видимо да су ове вредности приближно нула. У `rotation` је уписана матрица A. У `sdev` уписане су стандардне девијације сваке од главних компоненти, а у `x` саме главне компоненте.

Треба напоменути да `prcomp` подразумевано прави сопствене векторе који су негативно усмерени („према доле''), тако да, по потреби, можемо узети њихове супротне вредности.

Дисперзија сваке од главних компоненти је, наравно, њена стандардна девијација на квадрат, а пропорцију дисперзије објашњене њоме добијамо једноставно:

```
PVE <- (pca_result$sdev^2) / sum(pca_result$sdev^2)
PVE
```

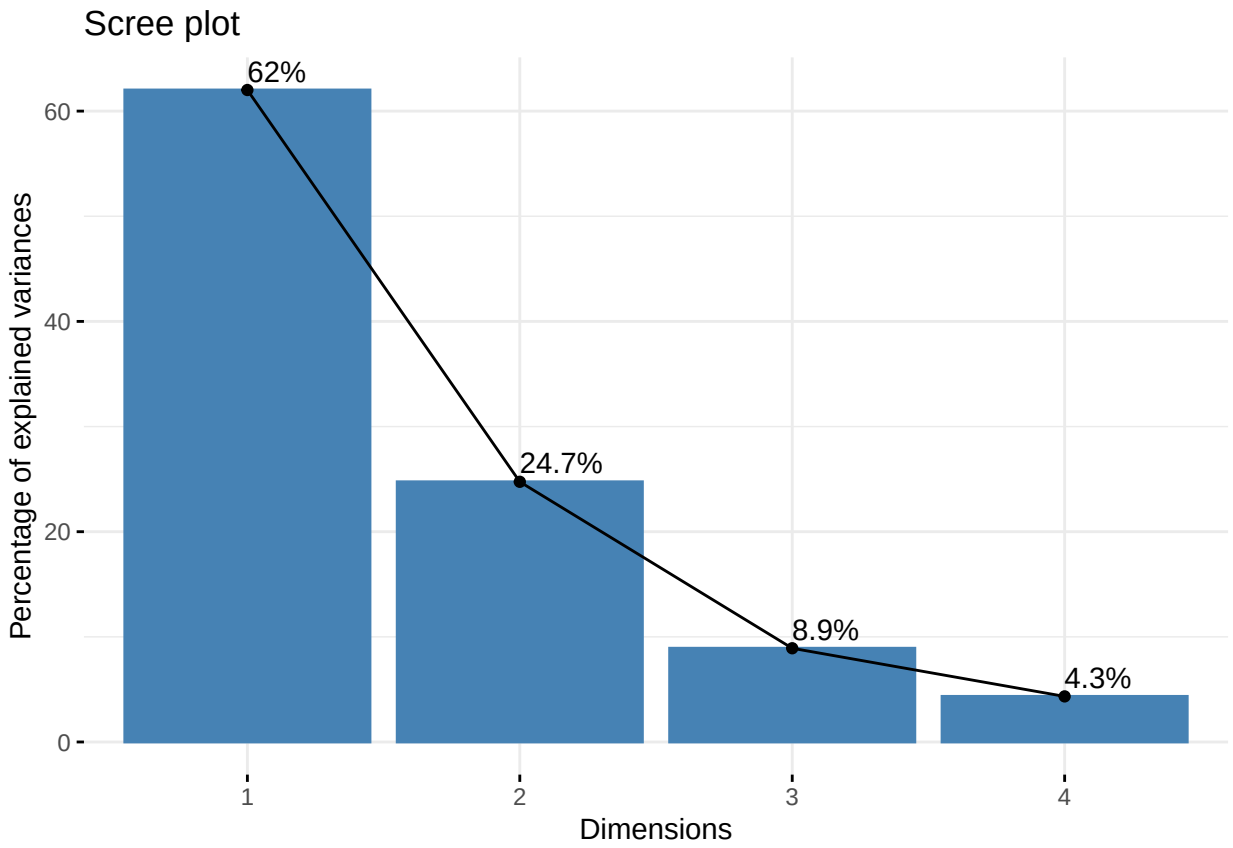
```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

```
sum(PVE)
```

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

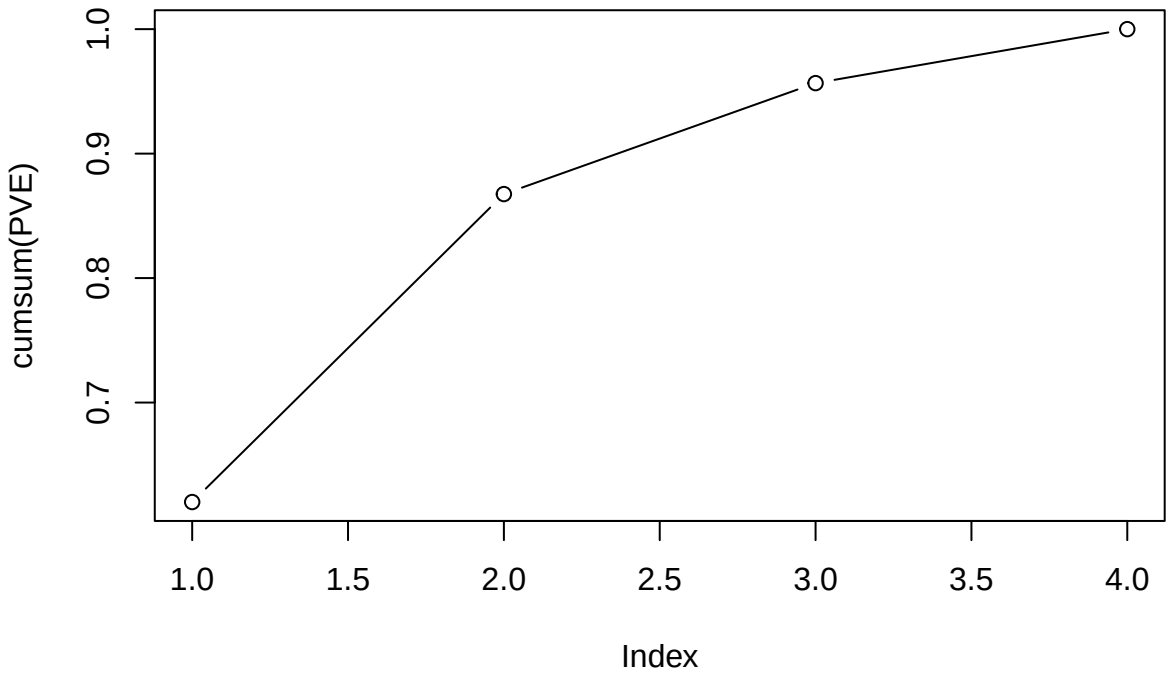
Сада треба одлучити колико главних компоненти задржати. За то је згодан `Scree plot` приказ који даје функција `fviz_eig` из пакета `factoextra`.

```
library(factoextra)
fviz_eig(pca_result, addlabels = TRUE)
```



У овом случају било би оптимално задржати две главне компоненте. За кумулативан приказ ад-хок решење је

```
plot(cumsum(PVE), type = "b")
```



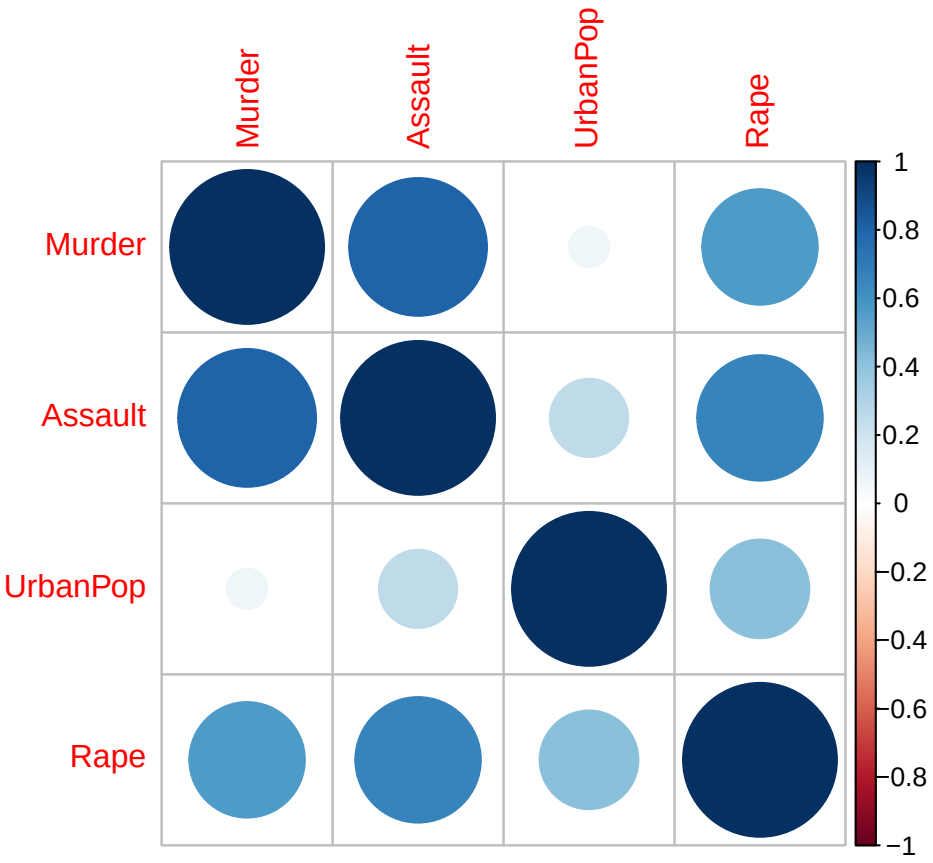
Ово је само демонстрација, у било какве практичне сврхе користили бисмо углађеније графике из `ggplot2` пакета.

Да сада мало обратимо пажњу на корелациону структуру нових и старих предиктора. Погледајмо корелациону матрицу оригиналних предиктора.

```
library(corrplot)
```

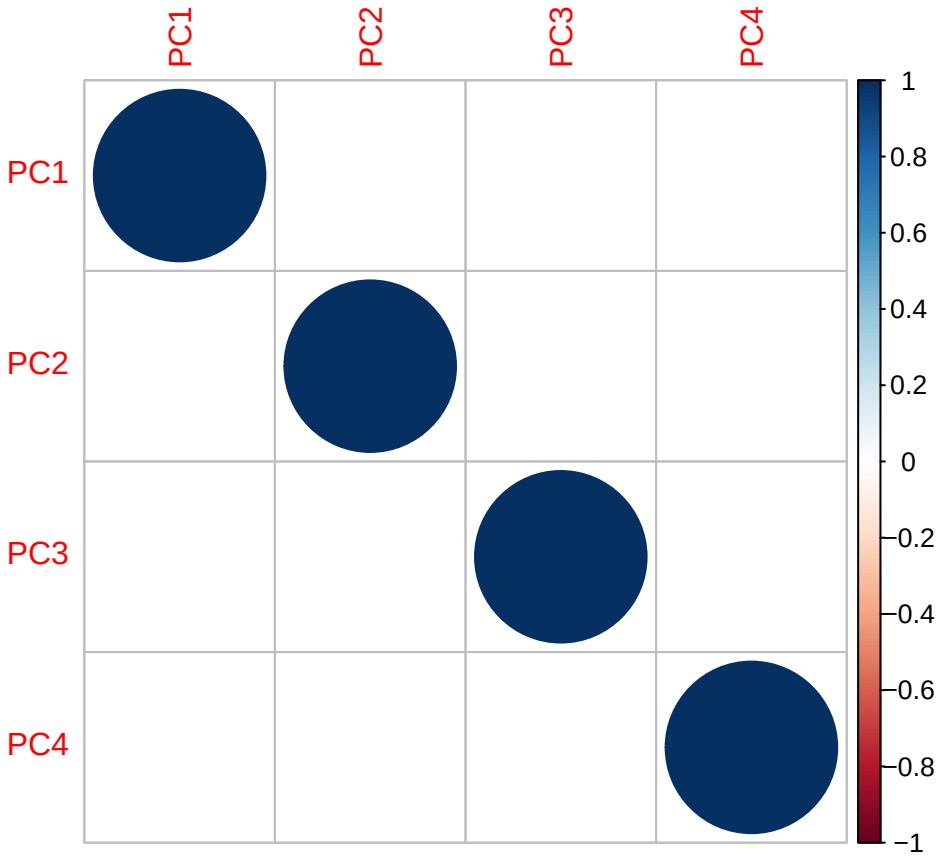
```
## corrplot 0.92 loaded
```

```
corrplot(cor(scaled_df))
```



Видимо да је присутна јака линеарна зависност међу предикторима. Ако сада погледамо корелациону матрицу главних компоненти:

```
corrplot(cor(pca_result$x))
```



Видимо да су главне компоненте некорелисане, што је наравно добро.

6.1.2 PCA - регресија

У претходном одељку смо показали шта је PCA и како функционише, а сада ћемо демонстрирати како се овај метод може користити заједно са регресијом, а у сврху предвиђања. Користићемо надалеко чувени `Boston` скуп података који говори о ценама некретнина, а желимо да предвидимо `medv`, средњу вредност цене некретнине у хиљадама долара.

Прво ћемо учитати податке.

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:ISLR2':
##
##     Boston
```

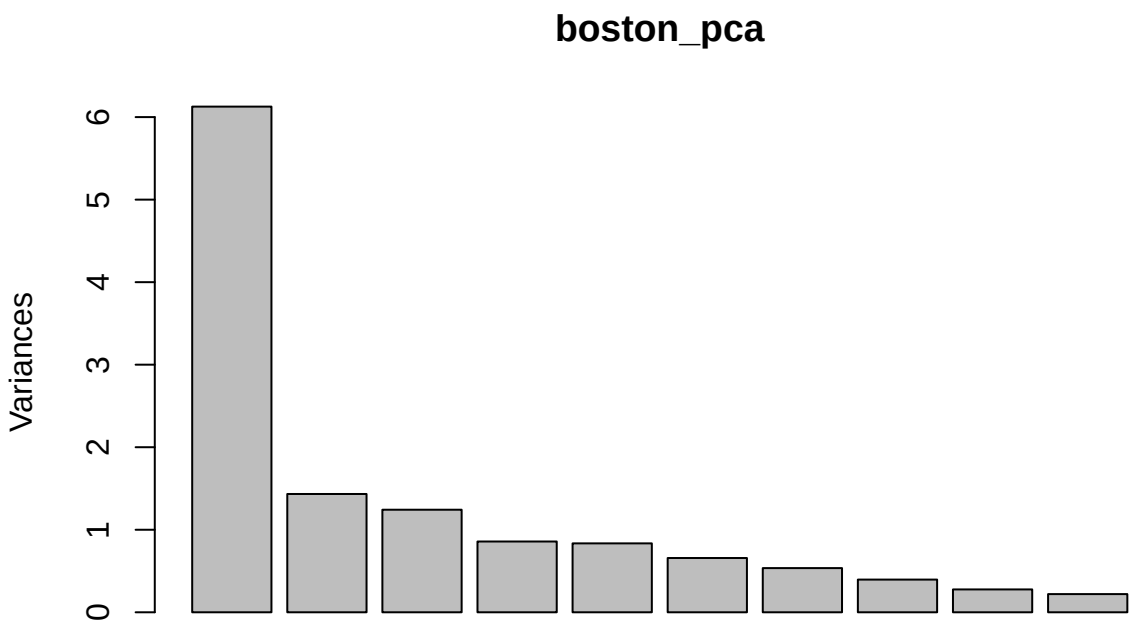
```
data("Boston")
```

Сада ћемо да извршимо PCA на свим колонама осим `medv`, а њу узимамо за зависну.

```
boston_preds <- subset(Boston, select = -c(medv))
boston_pca <- prcomp(boston_preds, center = TRUE, scale = TRUE)
```

Можемо кратко бацити поглед на дисперзије сваке од компоненти.

```
plot(boston_pca)
```



За прављење регресије користићемо се пакетом `caret`.

```
library(caret)
```

Генеричка функција за прављење модела из овог пакета јесте функција `train`, која има јако згодан аргумент `preProcess` којем може да се каже како да претпроцесира податке пре саме регресије.

```
set.seed(123)
model_pcr <- train(medv ~.,
                   data = Boston,
                   method = "lm",    # linearna regresija
                   preProcess = c("center", "scale", "pca")
                   )
```

Да видимо шта је урадио.

```
model_pcr
```

```
## Linear Regression
##
## 506 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13), principal component
##  signal extraction (13)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 506, 506, 506, 506, 506, 506, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   5.401168  0.6604614  3.595097
##
```

```
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
summary(model_pcr)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.018  -2.810  -0.888   1.604  31.392
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 22.53281    0.22382 100.674  < 2e-16 ***
## PC1         -2.27302    0.09051 -25.113  < 2e-16 ***
## PC2          2.19491    0.18714  11.729  < 2e-16 ***
## PC3          3.50099    0.20098  17.419  < 2e-16 ***
## PC4         -1.08068    0.24193  -4.467 9.84e-06 ***
## PC5          2.23308    0.24521   9.107  < 2e-16 ***
## PC6          0.67063    0.27632   2.427  0.01558 *
## PC7          0.09431    0.30620   0.308  0.75820
## PC8         -1.04018    0.35598  -2.922  0.00364 **
## PC9          0.14945    0.42573   0.351  0.72570
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.035 on 496 degrees of freedom
## Multiple R-squared:  0.7057, Adjusted R-squared:  0.7003
## F-statistic: 132.1 on 9 and 496 DF,  p-value: < 2.2e-16
```

Видимо да је користио све главне компоненте за модел, те видимо да има и оних које нису значајне. Сада ћемо поделити податке на тренинг и на тест скуп да направимо предвиђање, те да проверимо оверфит за сваки случај.

```
boston_train_index <- createDataPartition(1:nrow(Boston), p = 0.8, list = FALSE)
boston_train <- Boston[boston_train_index, ]
boston_test <- Boston[-boston_train_index, ]
```

Сада ћемо да направимо модел само на тренинг скупу.

```
set.seed(123)
model_pcr_train <- train(medv ~.,
                         data = boston_train,
                         method = "lm",
                         preProcess = c("center", "scale", "pca")
                         )
model_pcr_train
```

```
## Linear Regression
##
## 406 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13), principal component
```

```
##  signal extraction (13)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 406, 406, 406, 406, 406, 406, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   5.361084  0.6694754  3.59626
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Сада вршимо предвиђање.

```
boston_test_prediktori <- subset(boston_test, select = -c(medv))
boston_test_target <- subset(boston_test, select = c(medv))[, 1]
# kastovanje u "numeric"

predvidjanje <- predict(model_pcr_train, newdata = boston_test_prediktori)
```

Да видимо грешку.

```
sqrt(mean((boston_test_target - predvidjanje)^2))
```

```
## [1] 4.241712
```

Грешка на тренинг скупу била је 5.401168, тако да свакако нисмо у оверфиту. Како видимо да нам се грешке доста машавају, сетимо се да имамо и бољи метод процене грешке на тренинг скупу, а то је кросвалидација.

```
ctrl <- trainControl(
  method = "cv",
  number = 35    # radimo 35-fold
)
```

Направимо нови модел. То је заправо исти модел, само другачије оцењује грешку.

```
set.seed(123)
model_pcr_train_2 <- train(medv ~.,
                           data = boston_train,
                           method = "lm",
                           preProcess = c("center", "scale", "pca"),
                           trControl = ctrl
                           )
model_pcr_train_2
```

```
## Linear Regression
##
## 406 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13), principal component
##  signal extraction (13)
## Resampling: Cross-Validated (35 fold)
## Summary of sample sizes: 394, 394, 394, 394, 394, 395, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
```

```
##   4.934065  0.7352477  3.56909
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
summary(model_pcr_train_2)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.686  -2.805  -1.006   1.519  31.387
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  22.6520     0.2597  87.227  < 2e-16 ***
## PC1          -2.2026     0.1055 -20.887  < 2e-16 ***
## PC2           2.0314     0.2165   9.381  < 2e-16 ***
## PC3           3.8558     0.2355  16.371  < 2e-16 ***
## PC4          -1.4997     0.2763  -5.427 1.00e-07 ***
## PC5          -1.2728     0.2864  -4.443 1.15e-05 ***
## PC6           0.9138     0.3183   2.871  0.00431 **
## PC7           0.4105     0.3516   1.167  0.24372
## PC8           1.0376     0.4060   2.556  0.01097 *
## PC9           0.1813     0.4942   0.367  0.71389
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.233 on 396 degrees of freedom
## Multiple R-squared:  0.6842, Adjusted R-squared:  0.677
## F-statistic: 95.31 on 9 and 396 DF,  p-value: < 2.2e-16
```

Видимо да много боље погађа грешку.

Напомена. За штеловање броја фолдова ја сам користио метод „од ока”. Наравно, у реалном животу, треба користити валидациони скуп, као и друге методе које смо раније помињали („лакат”).

Сада ћемо још да задржимо само оне главне компоненте које објашњавају 80% варијабилности унутар података. Подразумевано се задржава 95%.

```
set.seed(123)
boston_train_for_pp <- subset(boston_train, select=-c(medv, chas))
boston_train_wait <- subset(boston_train, select=c(medv, chas))
# chas je kategoricka, ne valja za pca
pp <- preProcess(boston_train_for_pp, method = "pca", tresh = 0.8)
# Pravimo pretprocesirani boston_train
boston_train_pp <- predict(pp, boston_train_for_pp)
# Dodajemo mu kolone koje smo izbacili
boston_train_pp <- cbind(boston_train_pp, boston_train_wait)


model_pcr_train_3 <- train(medv ~.,
                           data = boston_train_pp,
```

```
                        method = "lm",
                        trControl = ctrl
                        )
model_pcr_train_3
```

```
## Linear Regression
##
## 406 samples
##  10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (35 fold)
## Summary of sample sizes: 394, 394, 394, 394, 394, 395, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   4.979478  0.7315984  3.618851
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

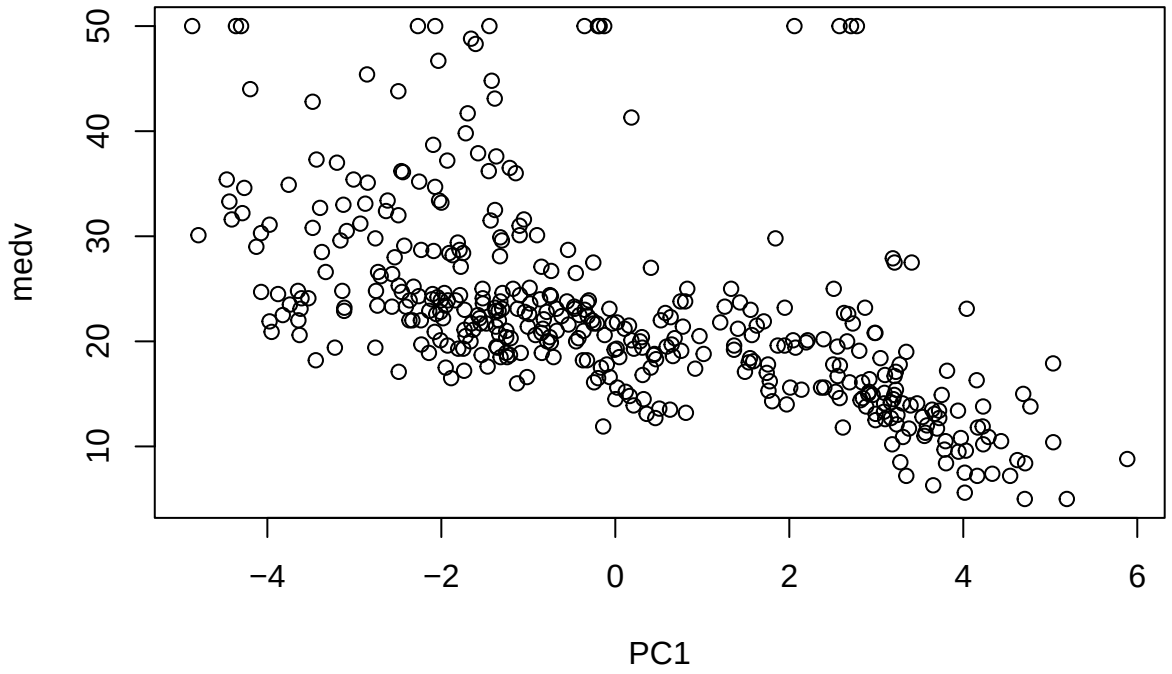
```
summary(model_pcr_train_3)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.936  -2.767  -1.022   1.403  29.060
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  22.4651     0.2681  83.798  < 2e-16 ***
## PC1          -2.2147     0.1050 -21.083  < 2e-16 ***
## PC2           0.7731     0.2255   3.428 0.000671 ***
## PC3           4.4397     0.2410  18.425  < 2e-16 ***
## PC4          -1.4819     0.2847  -5.205 3.13e-07 ***
## PC5          -0.8421     0.3148  -2.675 0.007784 **
## PC6          -0.3434     0.3494  -0.983 0.326206
## PC7          -1.0372     0.4042  -2.566 0.010659 *
## PC8          -0.1843     0.4920  -0.375 0.708166
## PC9          -1.1232     0.5324  -2.110 0.035506 *
## chas          2.8093     1.0647   2.639 0.008652 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.21 on 395 degrees of freedom
## Multiple R-squared:  0.6877, Adjusted R-squared:  0.6798
## F-statistic: 86.97 on 10 and 395 DF,  p-value: < 2.2e-16
```

Видимо да нисмо много покварили модел (можда је чак и мало бољи) у смислу показатеља квалитета, али смо га доста поједноставили. Ово свакако није никакав пример квалитетног предвиђања, већ више демонстрација како се PCA користи у комбинацији са регресијом.

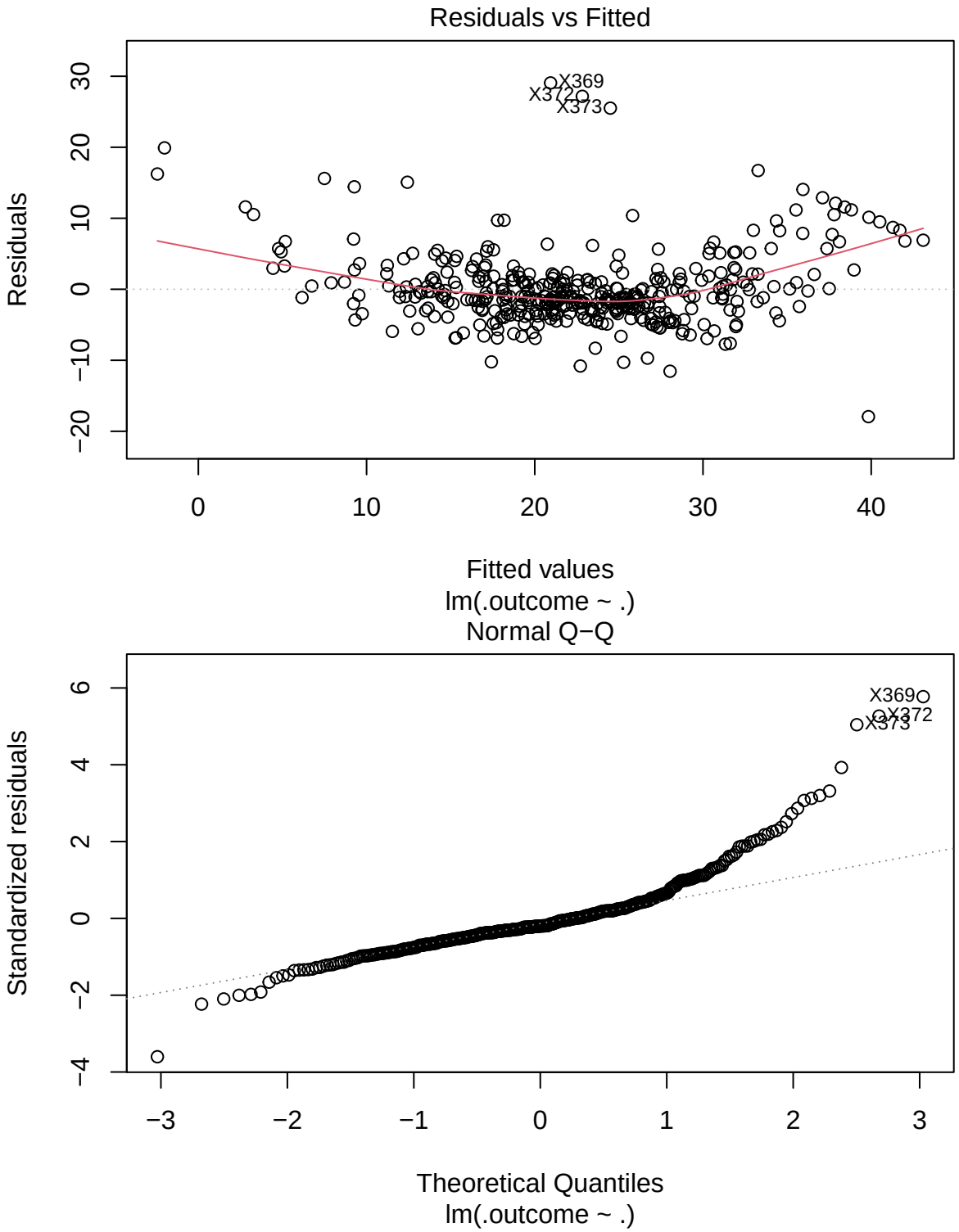
Ако се некоме чини да PCA некако одузима везу података од зависне променљиве, наредни график би требало да га разувери.

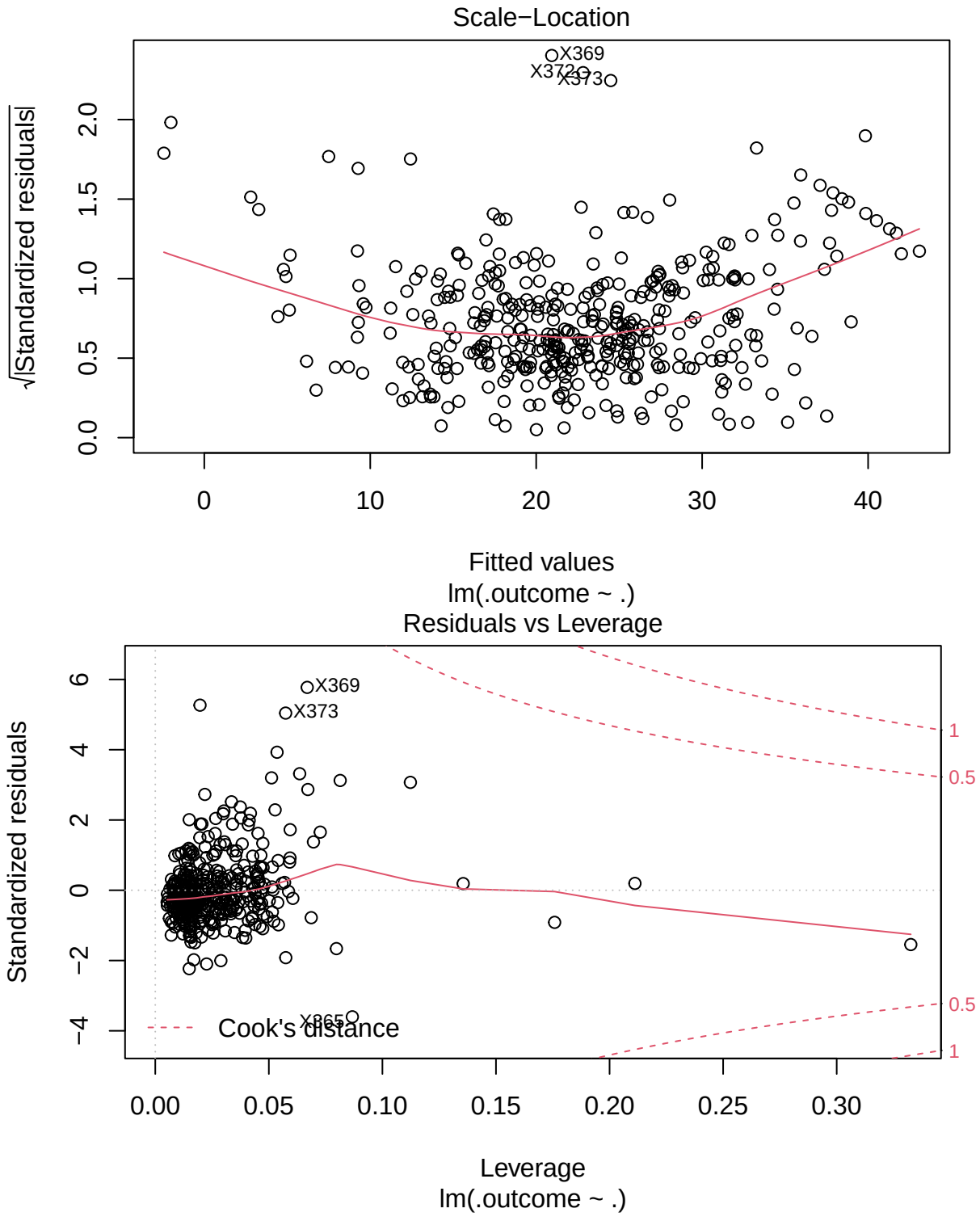
```
plot(medv ~ PC1, data = boston_train_pp)
```



Види се јасна зависност. Такође, види се и зашто је модел лош: зависност није баш линеарна. Ако нацртамо дијагностичке графике, видимо да није добро.

```
plot(model_pcr_train_3$finalModel)
```


Residuals vs Fitted
X369
X372
X373
Residuals
30
20
10
0
−10
−20
0
10
20
30
40
Fitted values
lm(.outcome ~ .)
Normal Q−Q
X369
X372
X373
Standardized residuals
6
4
2
0
−2
−4
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(.outcome ~ .)



Значи да би модел требало даље надограђивати трансформисањем главних компоненти. Ми нећемо, није нам то циљ. Коме је занимљиво нек проба, добро је за вежбу.

6.2 *t*-SNE

Напомена. Концепт излагања преузет из [5].

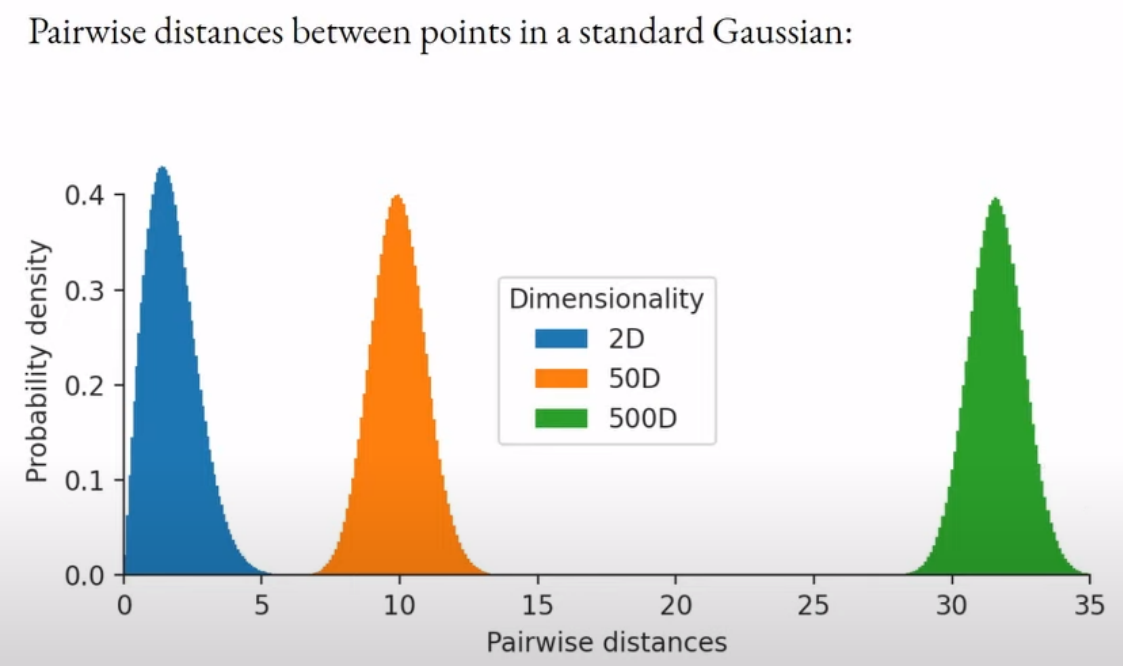
Замислимо да смо добили базу података коју треба да истражимо. Природна је ствар кренути од цртања. У причи о PCA смо видели да је цртање пар-по-пар графика неинформативно и рачунски захтевно. Стога се појавила потреба за методом који податке из вишедимензионог простора „пројектовати'' на дводимензионални простор који ми можемо у потпуности да перципирамо.

Прва идеја која човеку пада на памет јесте да се срачунају сва растојања између оригиналних тачака. Нека, рецимо имамо матрицу чије су врсте $\mathbf{x}_k$, и означимо $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$, где је норма одабрана по потреби датог случаја.

Сада се чини смисленим да пројекције $\mathbf{y}_i$ тачака $\mathbf{x}_i$ тражимо тако да се чувају растојања, то јест да за све парове тачака важи да је

$$\|\mathbf{y}_i - \mathbf{y}_j\| = d_{ij}.$$

То је, међутим, доказано немогуће, и познато је у машинском учењу као „клетва димензионалности'' (*Curse of Dimensionality*). Ми се овде нећемо теоријски уверавати у то, али можемо убедити себе на кратком примеру. На наредној слици дати су хистограми пар-по-пар растојања из генерисаних 2-димензионих, 50-димензионих и 500-димензионих нормалних случајних величина (ПСУ). Видимо да како расту димензије, нема малих растојања. Стога је немогуће очекивати да се нешто што изгледа као други или трећи хистограм утопи у дводимензион простор у коме очекујемо први хистограм.



Први одговор на овај проблем била је PCA, коју је смислио Карл Пирсон 1901, а независно од њега развио и именовао Харолд Хотелинг 1930их година. Међутим, PCA је у својој сржи линеарна метода и нема велику способност „развлачења података''. Њена основна идеја је да се цртају две главне компоненте, а остале да се занемарују. Јасно је и неутренираном оку колико ту губитака може настати.

t-SNE је скраћено од *t-Distributed Stochastic Neighbour Embedding*. За почетак ћемо се упознати са тиме шта је Neighbour Embedding, од сада скраћено NE. Прва и основна премиса за NE јесте да се

у потпуности одрекнемо идеје о очувању растојања и да пробамо нешто друго. То нешто друго биће покушај да се очувају **односи** растојања. То би значило да се тачке са левог краја зеленог хистограма пресликавају у леви крај плавог хистограма, и слично за десни крај и средину. У преводу, биће нам циљ да тачке које су оригинално близу буду близу и при „пројекцији'', а да оне које су биле оригинално далеко тако и остану. Дакле, није идеја да се очува растојање, већ *суседство*, или на енглеском *to preserve Nearest Neighbours*. Отуда и име.

G. Hinton и S.T. Roweis су 2002. у свом раду [15] поставили темеље алгоритма **Стохастичко NE**, у даљем тексту SNE. Овај рад до сада је цитиран преко 1464 пута.

Његова модификација јесте рад [16] (Hinton је аутор оба), који је у тренутку када пишем ове редове цитиран 23201 пут.

За почетак ћемо дефинисати *сличности* између опсервација као вероватноће p_{ij}. Што су две опсервације ближе, њихова сличност је већа. На исти начин ћемо дати и сличности пројекција, q_{ij}. Још ћемо видети како су ове сличности конкретно дефинисане.

Ако претпоставимо да смо њих задали, сада је идеја да нађемо пројекције које минимизују „разлику'' p_{ij}-ова и q_{ij}-ова. Сада се поставља ново питање: шта узети за функцију губитака, то јест шта ће да нам мери разлику сличности пројекција и сличности оригинала. Код SNE-а избор је пао на Кулбак-Лајблерово разилажење, које је дефинисано са

$$L = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

Ко упише курс Теорија информације упознаће се боље са овим појмом, а овде је довољно знати само шта мери. Битно је уочити и да Кулбак-Лајблерово разилажење није симетрично, па није метрика на простору вероватносних мера које су дискретног типа. Зато га и зовемо разилажење. Из формуле се види да је акценат на томе да се плаћа велика цена ако се блиске тачке пројектују далеко. Цена обрнуте грешке је нешто мања.

Сада је ред да покажемо како се дефинишу сличности међу тачкама. Узимамо

$$p_{j|i} := \frac{\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2\right)}{\sum_{k\neq i} \exp\left(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2\right)}.$$

Сигме се бирају да се постигне жељена Шенонова ентропија. То ћемо радити на Теорији информације, а овде се може замислити да уместо сигме стоји било која константа. Интуитивно: бирамо сигме тако да свака опсервација има K блиских суседа, а остале далеке, где је K фиксно, подразумевано 30. Како ове вероватноће нису симетричне, ми их симетризујемо и добијемо

$$p_{ij} := \frac{p_{i|j} + p_{j|i}}{2n}.$$

Ове вероватноће се по конструкцији сабирају на 1.

У многим ситуацијама може се проћи јефтиније, те се може дефинисати да је $p_{j|i} = 1/K$ за K најближих, а нула за остале. Често резултати нису много лошији.

Да видимо још како дефинишемо q_{ij}-ове. Дефинишемо

$$q_{ij} := \frac{k(\|\mathbf{y}_i - \mathbf{y}_j\|)}{\sum_{k\neq l} k(\|\mathbf{y}_k - \mathbf{y}_l\|)},$$

где SNE бира

$$k(d) = e^{-d^2},$$

а t-SNE бира

$$k(d) = \frac{1}{1+d^2}.$$

Напомена. Основна разлика је у тежини репова.

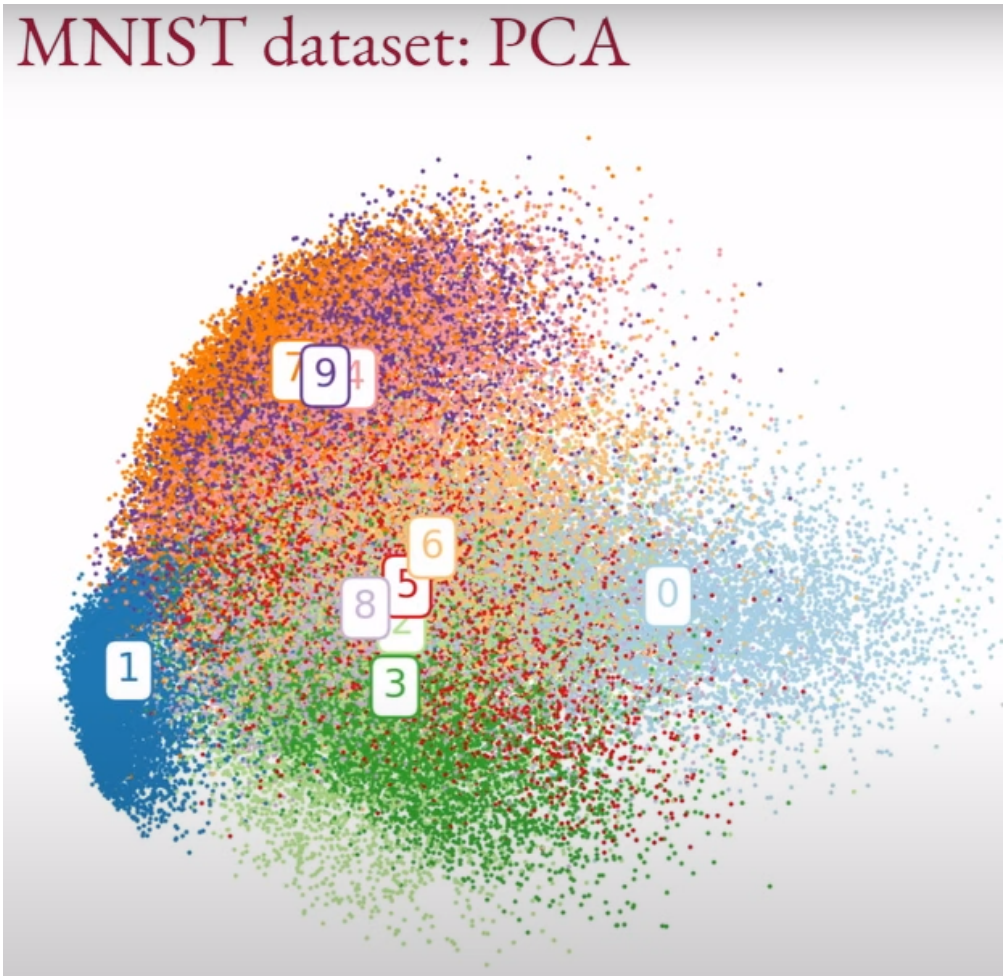
Остало је још објаснити како се оптимизује функција губитака. То се ради методом градијентног спуста, коју ћемо да радимо на наредном часу.

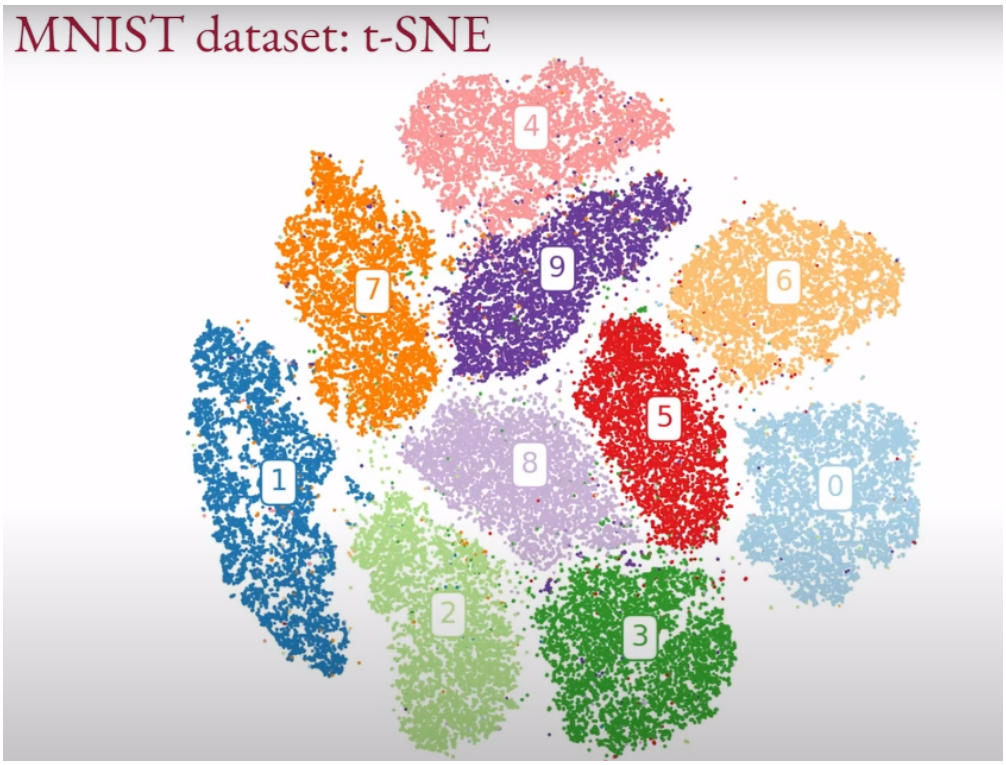
За сада још мало слика.

6.2.1 MNIST dataset

Modified National Institute of Standards and Technology скуп података представља скуп од 70 хиљада сличица 28×28 пиксела, лабелованих, које представљају ручно исписане цифре.

Да видимо шта уради PCA, а шта t-SNE.


MNIST dataset: PCA
9
6
5
8
3
1
0


MNIST dataset: t-SNE
4
7
9
6
5
1
8
0
2
3

Вежбе 7

Рад са недостајућим подацима

7.1 Увод

У статистици, *импутација* је процес замене недостајућих података заменским вредностима. Непотпуни подаци честа су појава у било ком стварном проблему, било због природе самог проблема, било због људског фактора. Такви подаци производе много проблема, а неки од главних су повећана пристрасност и смањена ефикасност оцена, као и значајно тежа анализа података.

Договоримо се око терминологије. Податке које анализирамо сматраћемо смештеним у матрицу, где свака врста те матрице представља једну опсервацију[1] (енг. *observation*, рус. *наблюдение*), а свако поље те врсте представља регистровану вредност одговарајућег обележја за дату опсервацију. Одавде следи да свака колона садржи регистроване вредности једног обележја кроз различите опсервације. Неки од тих података могу да недостају.

Рубин и Литл су 1987. године у [9] недостајуће податке класификовали у три категорије, у зависности од тога од чега све зависи вероватноћа недостајања. Изложимо њихову класификацију.

Нека X означава горе наведену $n \times p$ матрицу. Нека X_{miss} означава податке који недостају (те вредности постоје, само су нама недоступне), а нека X_{obs} онзначава податке који су нам доступни. То свакако неће бити матрице, али нам за сада није битно у ком су облику, битно нам је да нам X_{miss} садржи вредности у X које не знамо, а X_{obs} вредности које знамо.

Нека је R једна $n \times p$ матрица чије је поље r_{ij} једнако 1 уколико имамо податак x_{ij}, а 0 уколико тај податак недостаје. Матрица R може да зависи од неколико ствари, од којих су нама од интереса зависност од доступних података и зависност од недостајућих података. Нека су у θ смештени параметри од којих зависи расподела од X.

1. Уколико R не зависи ни од доступних ни од недостајућих података, кажемо да подаци задовољавају **MCAR** услов (енг. *missing completely at random*). Прецизније, подаци су MCAR уколико је
$$\mathbf{P}\{R \mid X_{obs}, X_{miss}, \theta\} = \mathbf{P}\{R \mid \theta\}.$$

2. Уколико R зависи од посматраних, али не и од недостајућих података, кажемо да подаци задовољавају **MAR** услов (енг. *missing at random*). Формално, то значи да је
$$\mathbf{P}\{R \mid X_{obs}, X_{miss}, \theta\} = \mathbf{P}\{R \mid X_{obs}, \theta\}.$$

[1] Никако *обзервацију*. Консултовати [18]. Синоними би били *посматрање*, *опажање* итд, али се латински термин одомаћио у статистици.

3. Уколико R зависи и од доступних и од недостајућих података, кажемо да су подаци **MNAR** (енг. *missing not at random*).

$$\mathbf{P}\{R \mid X_{obs}, X_{miss}, \theta\} = \mathbf{P}\{R \mid X_{obs}, X_{miss}, \theta\}.$$

Детаљније дефиниције, појашњења и примери у вези са ова три појма могу се пронаћи у [21].

Пример 7.1. Један од стандардних модела недостајања података јесте онај који почива на логистичкој функцији, а омогућава да се генеришу и MCAR и MAR и MNAR недостајања, за одговарајући избор параметара.

Нека су нам подаци $X = (X_1, X_2)$ дати за n опсервација и $p = 2$ овележја и нека је $R = (R_1, R_2)$, где су X_1, X_2, R_1 и R_2 колоне (последње две - колоне нула и јединица). Тада генеришемо недостајања као:

$$P(R_2 = 0) = \psi_0 + \psi_1 \frac{e^{X_1}}{1 + e^{X_1}} + \psi_2 \frac{e^{X_2}}{1 + e^{X_2}},$$

где је $\psi = (\psi_0, \psi_1, \psi_2)$ параметар са ненегативним компонентама[2]. Наравно, подразумевамо да је горња једнакост узета у одговарајућим компонентама вектора. Уколико желимо да генеришемо MCAR недостајање у другој колони, поставићемо вредност параметра на $(\psi_0, 0, 0)$, за MAR на $(\psi_0, \psi_1, 0)$, $\psi_1 \neq 0$, а за MNAR на (ψ_0, ψ_1, ψ_2), $\psi_2 \neq 0$.

За добијене податке рачунамо $P(R_2 = 0)$, те ако је веће од одабраног прага (популарно - трешхолда), податак бришемо.

7.2 Игнорабилно и неигнорабилно недостајање

Наставићемо дискусију имајући на уму претходни пример. Ми, у суштини, имамо два типа параметара (потенцијално вишедимензионих). Први јесте параметар ψ од којег зависи расподела за R. Други и битнији параметар јесте параметар θ од којег зависи расподела самих података X.

Свака анализа података, барем између осталог, има за задатак да оцени вредност параметра θ, који је увек у вези са неком знајаном статистиком на подацима: стредњом вредношћу, дисперзијом, коефицијентом спљоштености код неких расподела итд. Стога, веома је битно имати квалитетне оцене параметра θ. С друге стране, параметар ψ је „сметајући'', јер је његово присуство последица несавршености самих података са којима радимо.

Сходно претходно реченом, ваљало би знати када можемо сазнати све о θ, а да не знамо ништа о ψ, то јест када можемо сазнати параметре података, а да не знамо параметре недостајања. Ако хоћемо да се изразимо строго формално, можемо уочити да подаци који су нама доступни заправо нису подаци X_{obs}, већ здружени подаци[3] (X, R). У општем случају, расподела од (X, R) ће зависити од (θ, ψ).

Дефиниција 7.1. У горњим ознакама, недостајање је *игнорабилно* ако расподела за (X, R) зависи само од θ.

Рубин и Литл су у [14] показали довољне услове за игнорабилност у случају игнорабилности при закључивању методом максималне веродостојности. То ћемо дати у следећој теореми.

Теорема 7.1. *Недостајање је игнорабилно за закључивање коришћењем максималне веродостојности ако важи:*

1. *Недостајање је MCAR или MAR.*

[2] И, наравно, природним ограничењима, јер вероватноћа не сме бити већа од 1

[3] Како недостајања могу бити „разбацана" свуда по X ово неће чинити матрицу, али се може направити матрицом тако што се на места где је $r_{ij} = 0$ у X врше разне манипулације индикаторима. То сада није од велике важности.

2. *Параметри θ и ψ су различити, у смислу да је параметарски простор за (θ, ψ) једнак Декартовом производу параметарских простора за θ и за ψ.*

Показали су и да за правилно закључивање на Бајесовски начин, поред ова два услова мора да важи и априорна независност параметара θ и ψ.

Напомена. У пракси, услов различитости параметара је углавном испуњен, те се сва пажња усредсређује на констатацију тога да ли су недостајања MAR. Тестови тог типа превазилазе овај рад.

Још једна напомена. Уочимо да то што је недостајање игнорабилно не значи да можемо потпуно да га занемаримо, јер чак да су подаци и MCAR, уколико је недостајање 70% све оцене ће бити неквалитетне. Рубинови и Литлови услови хоће да кажу да неће бити *систематске* пристрасности у оценама, али оне свакако могу бити неупотребљиве због превелике дисперзије или преспоре конвергенције ка правој вредности параметра.

7.2.1 Последице игнорабилности

Концепт игнорабилности игра значајну улогу у конструкцији импутационих модела. То је зато што је наша генерална идеја та да одредимо расподелу $X_{miss}|X_{obs}, R$, те да из ње генеришемо импутационе вредности. Рубин је показао ([8]) да при претпоставци игнорабилности важи да расподела за $X_{miss}|X_{obs}, R$ не зависи од R, што специјално значи да је она једнака расподели за $X_{miss}|X_{obs}, R = 1$, па апостериорну (у смислу након недостајања) расподелу можемо моделовати само на основу доступних података. Уколико не важи услов игнорабилности, она се мора моделовати узевши и R у обзир.

Примере игнорабилног и неигнорабилног недостајања овде нећемо наводити јер премашују оквире (а нарочито циљеве) овога рада, а заинтересовани се упућују на [21], страна 40.

7.3 Методи обраде недостајућих података који не подразумевају импутацију

Сада ћемо се кратко осврнути на два најчешћа начина за руковање недостајућим подацима који не подразумевају импутацију, а то су уклањање целих опсервација и уклањање по паровима.

7.3.1 Уклањање целих опсервација

Један од најједноставнијих начина за руковање недостајућим подацима јесте **уклањање целих опсервација** (енг. *listwise deletion/complete-case analysis*, рус. *анализ полных наблюдений*). Тај метод подразумева уклањање целе једне опсервације из узорка, уколико јој недостаје макар једно поље. Може се показати да уколико подаци које посматрамо задовољавају MCAR услов, оцене добијене на овај начин јесу непристрасне. У случају да MCAR услов није задовољен (врло често[4]), оцене нису чак ни непристасне, а сви остали проблеми остају и продудбљују се.

Главна погодност овог метода је лакоћа рада. Уклањање целе опсервације уклања потребу за специјализованим софтвером за рад са некомплетним подацима, као и за посебним техникама за рад с њима. Ипак, избацивање целих опсервација, ако ништа друго, смањује обим узорка, а дисперзија великог броја оцена расте са тим смањење. Сходно томе, у већини ситуација губици су већи од предности које доноси, тако да се овај метод сматра једним од најлошијих.

7.3.2 Уклањање по паровима

Следећи метод за руковање недостајућим подацима јесте тзв. **уклањање по паровима** (енг. *pairwise deletion/available-case analysis*, рус. *попарное удаление пропущенных наблюдений*). Суштина овог метода

[4]Класичан пример незадовољености MCAR услова јесу анкете. Врло често у анкетама појављују се питања која велики број људи сматра осетљивим, те одбија да на њих одговори. Рецимо, људи веома често одбијају да говоре о висини својих примања.

јесте да се опсервација уклања уколико јој поље недостаје за неку конкретну анализу, а иначе не. Канонски пример јесте рачунање коваријације међу колонама (обележјима): уколико за барем једну опсервацију недостаје вредност једног од два обележја, избацујемо ту опсервацију за оба.

Овај метод зна дати боље резултате од претходног, али врло често зна дати и потпуно бесмислене оцене параметара; на пример, дешава се да оцена коефицијента корелације буде већа од 1. Заиста, оцена коефицијента корелације између колона X_k и X_l дата је са

$$\frac{\sum_i (x_{ik} - \bar{x}_k)(x_{il} - \bar{x}_l)}{\bar{s}_{X_k} \bar{s}_{X_l}},$$

што у случају потпуних података не прави проблем. Међутим, неки софтверски пакети за оцене узорачких дисперзија из имениоца користе само оне врсте које су познате и за једну и за другу колону, док други дисперзију за једну колону оцењују на основу у њој познатих врста, аналогно за другу. Тада се може десити да оцена коефицијента корелације „испадне'' из интервала $[-1, 1]$, из очигледних разлога.

Такође, може се десити да овај метод да оцену коваријационе матрице која је непозитивно (уместо позитивно/ненегативно) дефинитна, што даље ствара проблеме код свих врста анализе података које користе коваријациону матрицу. Детаљније се може прочитати у [19] на страни 40.

7.3.3 Једнострука и вишеструка импутација

У неким случајевима смислено је недостајућу вредност заменити тачно једном конкретном вредношћу (углавном статистиком која зависи од доступних података), док у другим ситуацијама постоји више потенцијалних кандидата којима има смисла попунити празнину. Стога, понекад је довољно просто уписати вредност у празно поље, док је у другим ситуацијама потребно одабрати најбољу могућу оцену недостајућег податка, или све смислене оцене на неки начин објединити у једну. Отуда и потреба за појмом вишеструке импутације.

7.4 Класичне методе

Сада ћемо изложити пар класичних метода за импутацију недостајућих података, који не подразумевају употребу регресије, а то су импутација средњом вредношћу и импутација узорачком медијаном. Ове методе су прилично јасне саме по себи, тако да ће ова глава бити прилично кратка.

Код импутације средњом вредношћу посматрају се недостајућа поља по колонама. Срачуна се средња вредност поља која су доступна, и она се упише у сва недостајућа поља. Још једна идеја може деловати смислено, а то је да се поменута средња вредност упише само у једно празно поље, те да се поново срачуна средина доступних поља, која сада укључују и оно које смо попунили, те да тако итерирамо све док не попунимо сва доступна поља. Међутим, тривијално је показати да то попуњава празна поља на идентичан начин као и прва (и једноставнија) идеја.

Код импутације узорачком медијаном идеја је слична претходној: срачуна се узорачка медијана сваке од колона и упише на сва празна места. Као и малопре, то је еквивалентно уписивању поље по поље и рачунању медијане сваки пут.

7.5 kNN импутација

Нека је x врста матрице X и нека је x^* настало од x избацивањем поља којима недостаје вредност. Импутацију вршимо у 2 корака:

1. Рачунамо растојање[5] између x^* и сваке од врста у X^c (подматрица комплетних врста у X). Идентификујемо k најближих, $k \in \mathbb{N}$.

[5] У норми по избору - најчешће L^2.

2. Недостајуће поље од x замењујемо медијаном[6] идентификованих k врста.

Овај метод је прилично смислен и једноставан за разумети, али има једну великуману: одабир одговарајућег k. Не постоји „рецепт'' за тај одабир, а он се углавном врши на основу додатних сазнања о подацима (и обима узорка, наравно).

7.6 Вишеструка импутација

Тешко је рећи ко се и када први сетио да недостајуће податке мења нечим смисленим. Било како било, „оцем импутације'' као математичке дисциплине сматра се амерички математичар[7] Доналд Б. Рубин (1943 -), тренутно професор емеритус на Универзитету Харвард. Уколико се неко и не слаже с тиме да је Рубин отац импутације, сви се слажу да је Рубин отац вишеструке импутације.

Идеја вишеструке импутације почива у математичком појму неодређености[8] (ентропије). Пре извршења било ког експеримента (у вероватносном смислу), постоји неодређеност о његовом исходу. Након извршења експеримента та неодређеност се губи и прелази у информацију коју његова реализација носи. У нашем случају, то би значило да се неодређеност губи онда када сазнамо праву вредност недостајућег податка и упишемо је где треба. Како то није реалан случај, јер не знамо која вредност је ту била, ми „погађамо''. Свакако да две смислене процене дају више информација о непознатој вредности од једне смислене процене. Отуда и идеја да се, уместо једном вредношћу, недостајући податак мења скупом вредности.

Напомена. Сви горњи појмови (неодређеност, информација, „носити информацију") могу се формализовати, али то је тема за неки други рад.

Идеја вишеструке импутације потекла је из Рубиновог ума 1977. године, а своје коначно обличје доживела је у [7], што је допуњавано и издавано још неколико пута.[9]

Иако смислена идеја, вишеструка импутација пред себе поставља неколико озбиљних изазова:

1. Како генерисати сваку од импутационих вредности?
2. Који је њихов оптималан број?
3. Како комбиновати све те импутационе вредности у једну (јер је то крајњи циљ, наравно)?
4. Како мерити квалитет технике?

На ова питања је углавном одговорено, а на већину је одговорио сам Рубин. Ми ћемо укратко описати сваки од корака, али за почетак морамо поставити амбијент у коме радимо.

Параметар популације Q јесте било која функција вредности неког обележја на целој популацији (нпр. средња вредност, медијана итд). Када вучемо узорак, ми не можемо израчунати тачну вредност тог обележја, већ само њену *оцену* $\hat{Q}$. Оваква оцена треба да поседује одређен квалитет, а Рубин од ње захтева да буде *непристрасна* и *валидна у смислу поверења*. Непристрасност подразумева да је очекивање ове оцене (просек вредности преко свих могућих узорака[10]) једнако Q, односно[11] $E(\hat{Q}|X) = Q$.

[6] Може и средњом вредношћу, али медијана је робуснија.

[7] Почео да студира физику, дипломирао психологију, докторирао статистику. Свашта нешто по вокацији, математичар у пракси.

[8] Нећемо је овде формално дефинисати, већ ћемо је само описати, што је довољно за наше потребе. За детаљније видети [17].

[9] Отуда се и помиње више пута у литератури - нешто што има у једном издању, нема у другом.

[10] Дакле, имамо дискретну случајну величину где су њене вредности све могуће вредности оцене $\hat{Q}$ за различите узорке, а вероватноће су вероватноће извлачења тих узорака.

[11] Ознака представља малу злоупотребу нотације, али је задржавамо јер је користе и Рубин и Бурен, због лакшег комбиновања литературе.

Да бисмо објаснили валидност у смислу поверења, узмимо да је U оцена дисперзије за $\hat{Q}$. Кажемо да је оцена валидна у смислу поверења ако очекивање ове оцене, као њен просек преко свих узорака, задовољава:

$$E(U|X) \geq D(\hat{Q}|X),$$

где је $D(\hat{Q}|X)$ дисперзија саме оцене, проузрокована узорковањем. Статистички тест са номиналним прагом значајности α треба да одбацује нулту хипотезу у *највише* $100\alpha\%$ случајева у којима је она тачна. Процедура је валидна у смислу поверења ако ово важи. За више детаља видети [11], страна 3.

7.6.1 Рубинова правила

Убацимо сада у причу и недостајуће податке. Права вредност параметра популације Q је непозната. Претпоставимо да смо направили њену оцену $\hat{Q}$. Количина информације (која се може и формализовати) коју $\hat{Q}$ даје о Q зависи од тога шта знамо о X_{miss}. Када бисмо га могли реконструисати са стопроцентном тачношћу, онда бисмо знали и параметар. Како то, наравно, није могуће, ми морамо да посматрамо расподелу за Q при разним X_{miss}.

Могуће вредности параметра Q на основу нама знаних података X_{obs} садржане су у расподели $P(Q|X_{obs})$. Она се може записати као

$$P(Q|X_{obs}) = \int P(Q|X_{obs}, X_{miss})P(X_{obs}|X_{miss})dX_{miss}, \tag{7.1}$$

а то на основу формуле потпуне вероватноће. Горњи интеграл је само формалан запис, и по потреби постаје и сума, у дискретном случају[12].

Напомена. Шта је шта у горњој једначини? $P(Q|X_{obs})$ јесте апостериорна расподела за наш параметар, при услову доступних података. Ово је оно што нас занима. Откуд сад то? Па, приметимо какав нам је до сада ток мисли: желимо да оценимо параметар, али имамо недостајуће податке. Ми онда прво оценимо њих, па затим на основу попуњених података и параметар. То је дупли посао! Сва информација која је нама доступна садржана је у X_{obs}. Ако њу искористимо за добијање оцене за X_{miss}, то је само због тога што ми тиме знамо да рачунамо, а не зато што говори нешто више. Због тога желимо да „непречимо" пут, те да оценимо параметар на основу доступних података. ОК, и?

Рецимо да је од интереса оцењивање средње вредности обележја на популацији. Ако га оценимо аритметичком средином на доступним подацима, правимо потенцијално огромну грешку, јер подаци могу да недостају веома „пристрасно" (нпр. недостају сви већи од неке вредности). Стога тражимо расподелу параметра при доступним подацима, те тек онда оцењујемо шта треба.

$P(Q|X_{obs}, X_{miss})$ представља теоријску „идеалну" расподелу параметра при свим доступним подацима. $P(X_{miss}|X_{obs})$ представља расподелу недостајућих података при доступним.

Једначину је згодно тумачити здесна налево. Претпоставимо да користимо расподелу $P(X_{miss}|X_{obs})$ да „вучемо'' импутационе вредности за X_{miss}, које ћемо означити са $\dot{X}_{miss}$. Онда можемо користити $P(Q|X_{obs}, \dot{X}_{miss})$ да срачунамо Q за дато $\dot{X}_{miss}$. Понављамо ове кораке пролазећи кроз све могуће вредности за $\dot{X}_{miss}$, те саберемо/интегралимо. Овиме смо компликовану величину са леве стране у нашој једначини, изразили преко једноставнијих расподела, из којих можемо да вучемо узорак.

Може се показати да важи следећа формула:

$$E(Q|X_{obs}) = E(E(Q|X_{obs}, X_{miss})|X_{obs}).$$

Да не би било забуне, објаснићемо шта горња једнакост представља. Дакле, очекивање условне расподеле параметра Q при услову доступних података једнако је условном, при услову доступних података,

[12] И сума је интеграл по бројачкој мери, тако да заправо говоримо о интегралу чији је носач скуп свих вредности које може узети X_{miss}, макар и вишедимензионих. dX_{miss} заправо представља интеграцију по оној Лебег-Стилтјесовој вероватносној мери коју генерише функција расподеле од X_{miss}, макар и вишедимензиона.

очекивању средње вредности параметра Q за разна „извлачења'' недостајућих података (нотација унутрашњег очекивања је злоупотребљена, као и раније).

Претходна једнакост даје нам идеју како комбиновати резултате вишеструких импутација. Претпоставимо да је $\hat{Q}_l$ оцена добијена из l-те поновљене импутације. Онда је комбинована оцена дата са

$$\bar{Q} = \frac{1}{m}\sum_{l=1}^{m}\hat{Q}_l.$$

Напомена. Дозвољавамо да параметар буде вишедимензион; тада је свако $\hat{Q}_l$ један $k \times 1$ вектор, а уместо оцене дисперзије треба посматрати оцену коваријационе матрице.

Тада за дисперзију апостериорне расподеле важи позната формула Теорије вероватноћа:

$$D(Q|X_{obs}) = E(D(Q|X_{obs}, X_{miss})|X_{obs}) + D(E(Q|X_{obs}, X_{miss})|X_{obs}).$$

Први сабирак представља просечну дисперзију параметра Q, где се просек узима преко различитих X_{miss} (оних које смо раније „вукли''). Други сабирак преставља дисперзију (коваријациону матрицу) међу очекивањима параметра на основу комплетних података, пролазећи кроз различите X_{miss}. Први сабирак на енглеском носи назив *within-variance*, а други *between-variance*.

Нека $\bar{U}_\infty$ и B_∞ означавају граничне вредности оцена горњих двају сабирака када број импутација m тежи бесконачности. Тада се апостериорна (при услову доступних података, да се подсетимо) дисперзија за Q може оценити са

$$T_\infty = \bar{U}_\infty + B_\infty.$$

Сада се јасно наслућује како ћемо оценити T_∞ на основу коначног m. Рачунамо да је

$$\bar{U} = \frac{1}{m}\sum_{l=1}^{m}\bar{U}_l,$$

где је $\bar{U}_l$ оцена коваријационе матрице од $\hat{Q}_l$, у l-тој импутацији. Стандардна непристрасна оцена другог сабирка јесте

$$B = \frac{1}{m-1}\sum_{l=1}^{m}(\hat{Q}_l - \bar{Q})(\hat{Q}_l - \bar{Q})^T,$$

где претпостављамо потенцијалну вишедимензионалност параметра, па отуда и транспонат.

Било би примамљиво рећи да је (непристрасна) оцена за T_∞ једнака збиру $\bar{U} + B$. Међутим, ту би се заборавила чињеница да је и само $\bar{Q}$ оцењено, те само апроксимира $\bar{Q}_\infty$. Рубин је у [8] показао да је члан који у суми „фали'' систематски, и једнак B_∞/m. Ово значи да за велико m он бива занемарљив, али за мало m добијамо мању дисперзију него што јесте. Како B апроксимира B_∞, можемо писати

$$T \approx \bar{U} + B + B/m = \bar{U} + \left(1 + \frac{1}{m}\right)B.$$

Сада ћемо на једно место записати две централне једначине из претходне приче:

$$\bar{Q} = \frac{1}{m}\sum_{l=1}^{m}\hat{Q}_l, \tag{7.2}$$

$$T \approx \bar{U} + \left(1 + \frac{1}{m}\right)B. \tag{7.3}$$

$\bar{Q}$ је оцена параметра добијена вишеструком ипмутацијом, а T оцена дисперзије те оцене. Горње две једначине познате су као *Рубинова правила*.

7.6.1.1 Три извора варијабилности

Имамо, дакле, параметар од интереса, Q који желимо да оценимо на основу некомплетних података. Испоставља се да његова дисперзија долази из трију извора:

1. $\bar{U}$, дисперзија која је узрокована чињеницом да узимамо узорак уместо да посматрамо целу популацију; стандардна је статистичка мера варијабилности;
2. B, додатна варијабилност узрокована тиме што у нашем узорку имамо недостајање;
3. B/m, додатна варијабилност потекла од симулација, то јест из чињенице да је само $\bar{Q}$ оцењено за коначно m.

Последња ставка је неопходна да би вишеструка импутација „радила'' за мале m, јер ће иначе p-вредности бити исувише мале, а интервали поверења неприродно кратки (јер је дисперзија наизглед мања, а заправо није). Што је веће m, то је мањи утицај симулација на укупну дисперзију. Ранији савети говорили су да се за m узме 3. 5, или 10, док данашње искуство показује да је боље узети веће m и препоручује се 50. Ове препоруке су потекле из искуства, а не из формалног рачуна.

7.6.1.2 Валидна импутација

На почетку ове главе смо имали појам валидности оцене у смислу поверења, што је значило да стварни ниво поверења треба да буде једнак номиналном, или већи. У тој уводној причи нисмо имали недостајуће податке, те ћемо сада разрадити појам валидности у случају присуства недостајања. У суштини, анализу података можемо посматрати на три нивоа, а који су дати у наредној табели.

Популација	Комплетан узорак: $X = (X_{obs}, X_{miss})$	Некомплетан узорак: X_{obs}
Q	$\hat{Q},\ U \sim D(\hat{Q})$	$\bar{Q}, \bar{U}, B \sim D(\bar{Q})$

Табела 7.1: Нивои анализе података

„Најлакши'' ниво за рад јесте онај у коме имамо извршен попис, то јест имамо вредности нашег (наших) обележја на целој популацији. Ту не постоји никаква неодређеност о вредности параметра Q, те је ту ситуација јасна. Наредни ниво јесте ниво у коме извлачимо узорак из популације. Тада вредност $\hat{Q}$ која представља (макар и непристрасну, што и захтевамо барем асимптотски) оцену параметра Q не носи сву информацију о параметру, зато што не знамо у ком степену оцена „одступа'' од праве вредности параметра. О томе нам говори U, оцена коваријационе матрице/дисперзије оцене $\hat{Q}$. Тада пар $(\hat{Q}, U)$ садржи све што знамо о параметру.

Онај ниво који нама и јесте од интереса, а који ћемо с правом описати као „најтежи'', јесте онај када не само да имамо узорак, већ је он још и некомплетан. Сада оно што је било оцена, узима улогу онога што треба оценити, те добијамо $\bar{Q}$ као оцену за $\hat{Q}$, $\bar{U}$ као оцену за U, као и нову величину B, која „купи" варијабилност насталу чињеницом да недостајање постоји.

Процес импутације би требало, ако ништа друго, да барем даје адекватне оцене за $\hat{Q}$ и U. Изложићемо дефиницију дефиницију валидности дату у [21]. Импутациони процес је *валидан у смислу поверења* ако приближно (асимптотски), важе следећа три услова:

$$E(\bar{Q}|X) = \hat{Q} \tag{7.4}$$

$$E(\bar{U}|X) = U \tag{7.5}$$

$$\left(1 + \frac{1}{m}\right) E(B|X) \geq D(\bar{Q}). \tag{7.6}$$

Напомена. Овде се хипотетички комплетни подаци сматрају фиксним, то јест реалним бројевима при узимању очекивања. Шта је онда случајно? Случајна је матрица R, индикатор недостајања, и очекивање заправо иде по њеној расподели (јасно је да се за R мора претпоставити расподела).

Услови валидности су аналогни онима у случају без недостајања, те их нећемо посебно објашњавати. Нагласићемо само да валидност процеса импутације зависи и од самих параметара $(\hat{Q}, U)$, али и од расподеле недостајања. Стога се може десити да је иста процедура некад валидна, а некад није. Тестови валидности импутационих механизама свакако превазилазе овај рад, а радознали се упућују на [21].

7.6.1.3 Односи варијабилности

Задржаћемо досадашње ознаке, те посматрати количник:

$$\lambda = \frac{B + B/m}{T}.$$

Овај количник можемо тумачити као пропорцију варијабилности у подацима чије се порекло може приписати недостајању података. Он је једнак нули или уколико недостајања нема, или уколико са сигурношћу можемо реконструисати недостајуће податке.

Количник

$$r = \frac{B + B/m}{\bar{U}}$$

назива се *релативни пораст варијабилности потекао из недостајања* (енг. relative increase in variance due to nonresponse). Важи веза $r = \lambda/(1-\lambda)$.

Још једна мера коју ћемо размотрити јесте *удео информације о Q који немамо због недостајања.*[13] Ова мера дефинисана је са

$$\gamma = \frac{r + 2/(\nu + 3)}{1 + r}.$$

За ову меру неопходно је знати (оцену за) број степени слободе, о којем ћемо дискутовати у наредном пододељку.

Иако нисмо експлицитно нагласили, јасно је да су горње мере дефинисане за параметар који је скалар. Ако је параметар димензије k, тада се добијају аналогне величине:

$$\bar{\lambda} = \left(1 + \frac{1}{m}\right) \mathrm{tr}(BT^{-1})/k, \quad \bar{r} = \left(1 + \frac{1}{m}\right) \mathrm{tr}(B\bar{U}^{-1})/k.$$

За детаљније описе ових величина треба погледати [21], страна 47, као и литературу на коју се тамо упућује.

7.6.1.4 Степени слободе

Број степени слободе представља број опсервација (елемената узорка) умањен за број параметара у моделу. Рачунање броја степени слободе не може бити исто и кад имамо и кад немамо недостајање. У случају недостајућих података, Рубин је у [8] (једначина 3.1.6) извео да је

$$\nu_{old} = (m-1)\left(1 + \frac{1}{r^2}\right) = \frac{m-1}{\lambda^2},$$

где смо користили индекс *old* (енг. за *стар*), јер је оцена претрпела извесне корекције. Најнижа могућа вредност је $\nu_{old} = m - 1$, што значи да се сва варијабилност у подацима може приписати недостајању. Супротно од тога је ν_{old} значи да сва варијабилност потиче од узорковања, па или недостајања нема, или смо недостајуће податке савршено реконструисали.

[13] Појам је детаљније описан у [8], једначина 3.1.10.

Бернард и Рубин ([12]) су уочили да претходни израз може узети вредности веће од обима комплетног узорка, те да оцена има смисла само за велике узорке. За мале узорке неопходна је корекција оцене, да би иста имала смисла. Означимо са ν_{com}, број степени слободе за $\bar{Q}$ у хипотетички комплетним подацима. Ако је узорак обима n, а параметар димензије k, можемо ставити $\nu_{com} = n - k$. Барнард и Рубин су извели да је тада оцена броја степени слободе која урачунава информацију која недостаје дата са

$$\nu_{obs} = \frac{\nu_{com} + 1}{\nu_{com} + 3}\nu_{com}(1 - \lambda).$$

Оцена броја степени слободе коју они препоручују за тестирање јесте

$$\nu = \frac{\nu_{old}\nu_{obs}}{\nu_{old} + \nu_{obs}}. \tag{7.7}$$

Она је увек мања или једнака од ν_{com}, што смо и хтели. Ако је $\lambda = 0$, тада је $\nu = \nu_{com}$, што нам одговара јер знамо да том случају одговара да недостајање не утиче на варијабилност. Супротан случај јесте $\lambda = 1$ и тада је $\nu = 0$. Тада је сва варијабилност потекла од недостајања, те тада нема никаквог смисла радити било какве тестове, јер не поседујемо никакву информацију о параметру.

Постоји још корекција ове оцене, али ми их нећемо детаљније обрађивати. Набројане су у [21] на страни 48. Проћи ћемо кроз један пример у R-у.

7.6.1.5 Показни пример

У програмском језику R постоји мноштво пакета који се баве импутацијом недостајућих података, а онај који је специјализован за вишеструку импутацију назива се mice. Његов аутор је сам Стеф ван Буурен, аутор [21] и ученик Доналда Б. Рубина.

У бази nhanes налазе се подаци о разним здравственим параметрима пацијената. Наравно, у бази има недостајања. Два предиктора су категоричка (кодирана), а два нумеричка. Број опсервација је 25. Сада ћемо видети како се врши вишеструка импутација.

```
library(mice)
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
imp <- mice(nhanes, print = FALSE, m = 10, seed = 10000)
```

imp је класе mids, што је скраћеница за *Multiply imputed data set*. Функција сама бира механизам импутације, то јест начин на који се врше извлачења. То може да се предочи функцији као аргумент, али о том - потом. Подразумевани механизам познат је под називом *Predictive mean matching*, који ћемо ми и описати у наредној глави.

```
fit <- with(imp, lm(bmi ~ age))
```

Рачуна се оцена параметара за сваки импутирани dataset, на основу задате формуле. Овде су параметри два регресиона коефицијента, а формула је стандардна из линеарне регресије. Класа објекта fit јесте mira, што је скраћено за *Multiply imputed repeated analyses*.

```
estimate <- pool(fit)
estimate
```

```
## Class: mipo    m = 10
##          term  m  estimate      ubar         b        t dfcom       df
## 1 (Intercept) 10 29.495604 3.5170075 2.1633415 5.896683    23 10.30216
## 2         age 10 -1.787729 0.9353743 0.3880015 1.362176    23 12.57841
##         riv    lambda       fmi
## 1 0.6766194 0.4035617 0.4932371
## 2 0.4562897 0.3133234 0.4014809
```

Видимо да је класа наше оцене `mipo`, што је скраћено од *Multiple imputation pooled object*. Наравно, pooling се врши уз помоћ раније наведених Рубинових правила. Прођимо сада кроз све елементе у `estimate`. Јасно, `m` означава број вишеструких импутација које смо задали. `estimate` даје оцене регресионих коефицијената, које смо и тражили. Оне одговарају оцени $\bar{Q}$ из претходних поглавља. Колоне `ubar`, `b` и `t` представљају оцене дисперзија које смо имали раније под истим именом (bar = црта изнад), узете за сваки коефицијет посебно (јер би иначе биле 2×2 матрице). Колона `dfcom` одговара оцени ν_{com}, а колона `df` одговара оцени ν добијеној Барнард-Рубиновом корекцијом. Релативни пораст варијабилности r дат је колоном `riv` (*relative increase in variance*). `lambda` је јасно. Мера γ дата је колоном `fiv`, што је од енглеског *fraction of missing information*. Наравно, и она је рачуната по параметру.

7.6.2 Интервали поверења и тестови

Поред саме тачкасте оцене параметра која нас занима, веома често је од интереса пронаћи интервал поверења за тај параметар/оцену, као и тестирати да ли параметар има неку вредност, или припада неком скупу. Да би се то успрешно спровело, треба знати расподелу оцене.

У свим до сада познатим методама претпоставља се да важи да је

$$Q - \hat{Q} \sim \mathcal{N}(0, U),$$

где су све ознаке исте као из претходних одељака.

Напомена. Када горњи услов није испуњен, тражи се нека функција параметра за коју горња нормалност барем приближно важи.

Суштински, ситуација може да се грана у два случаја: први, у којем је параметар скалар и други, у којем је параметар вектор (вишедимензион). Други случај је знатно компликован и ми се њиме нећемо овде бавити, а заинтересовани о њему могу пронаћи у [21], у 5. глави. Дакле, претпоставићемо да је параметар једнодимензион.

7.6.2.1 Закључивање у случају једнодимензионог параметра

У овоме случају U је број, а не матрица, а нормална расподела о којој је било речи је једнодимензиона.

Напомена. Случај једнодимензионог параметра може се примењивати и када је параметар вишедимензион, али се тада процедура понавља за сваку његову компоненту као параметар за себе.

Рубин је показао да асимптотски важи релација

$$\frac{Q - \bar{Q}}{\sqrt{T}} \sim t_\nu,$$

где је t_ν Студентова расподела са ν степени слободе, где је ν дефинисано једначином 7.7.

Стандардно, $100(1-\alpha)\%$ интервал поверења за Q рачуна се као

$$\bar{Q} \pm t_{\nu,1-\alpha/2}\sqrt{T},$$

где је $t_{\nu,1-\alpha/2}$ ознака за одговарајући квантил Студентове расподеле са ν степени слободе.

Уколико тестирамо нулту хипотезу $Q = Q_0$, за неку конкретну вредност Q_0, Рубин је показао да се p-вредност теста налази по формули:

$$p = P\left[F_{1,\nu} > \frac{(Q_0 - \bar{Q})^2}{T}\right],$$

где је $F_{1,\nu}$ ознака за случајну величину са Фишеровом расподелом са једним и ν степени слободе.

У `R`-у, претходне показатеље добијамо позивом функције `summary`, на објекат класе `mipo`. Ми ћемо искористити `estimate` из претходног примера који смо имали.

```
summary(estimate)
```

```
##          term  estimate std.error statistic       df      p.value
## 1 (Intercept) 29.495604  2.428309  12.14656 10.30216 1.957215e-07
## 2         age -1.787729  1.167123  -1.53174 12.57841 1.503430e-01
```

У нашем примеру (подсетимо се, проста линеарна регресија), променљива `statistic` представља класичну Валдову тест статистику на коју смо навикли код линеарне регресије. Уколико желимо да нам `summary` исписује и интервале поверења, довољно је као додатни аргумент додати `conf.int = TRUE`.

7.6.3 Квалитет импутационог метода

Рубин је показао да се бенефити вишеструке импутације „виде'' тек у случају да су импутациони методи валидни, у смислу дефиниције коју смо раније имали. Провера валидности углавном се врши симулацијама. Суштински, постоје два механизма која утичу на доступне податке: механизам узорковања и механизам недостајања. Симулацијама могу да се третирају сваки посебно, али и комбиновано. То нас доводи до три суштинска симулациона дизајна.

1. **Само механизам узорковања.** Одаберемо познат параметар, то јест расподелу са познатим параметром. Вадимо узорке $X^{(s)}$, на основу њих правимо оцене $\hat{Q}^{(s)}$ и $U^{(s)}$ и рачунамо просек за велико s. Требало би да добијемо вредности параметра и његове коваријационе матрице које су блиске правој.

2. **Оба механизма комбиновано.** За познат параметар вучемо узорке $X^{(s)}$, затим у сваком од њих генеришемо недостајања, те добијамо $X_{obs}^{(s,t)}$; импутирамо, оцењујемо $\hat{Q}^{(s,t)}$ и $T^{(s,t)}$ и упросечимо преко свих s и t.

3. **Само механизам недостајања**. За дату оцену $(\hat{Q}, U)$ генеришемо недостајања на подацима из које смо је добили, импутирамо велики број пута, те упросечимо добијене $(\bar{Q}, \bar{U})^{(t)}$, $B^{(t)}$. Предност овог приступа јесте што нам не треба модел података/параметра на популацији.

Неке честе мере квалитета које се користе јесу пристрасност оцене $\bar{Q}$, стопа покривања (пропорција интервала поверења који садрже праву вредност), просечна ширина интервала поверења, као и често виђани корен средње квадратне грешке. Сваки од показатеља има и своје предности и своје мане, које често зависе и од конкретне расподеле података, те их нећемо детаљније ни разматрати да не бисмо превише отишли у ширину.

7.7 Predictive mean matching

7.7.1 Како вишеструко импутирати? Показни пример у `R`-у

До сада смо, што код Рубинових правила, што код приче о евалуацији импутационог метода, говорили о кораку „импутирамо податке'', а затим са тако комплетним узорком радимо нешто. Ми смо тај

механизам којим импутирамо сматрали датим, само нама у датом тренутку непознатим/небитним (на енглеском се то зове *blackboxing*). Сада је дошло време да кажемо нешто о томе како се то ради. Кренућемо поступно, на примеру.

Напомена. Пример је преузет из [21], али је у извесној мери модификован. Није било сврхе смишљати неки оригиналнији, јер сваки погађа исту поенту, а база из овог примера је лепа за показивање идеја, због јаке линеарне везе предиктора.

Одабраћемо базу `whiteside` из пакета `MASS`. Човек презимена Уајтсајд[14] (енг. *Whiteside*) у дворишту своје куће у Лондону сваке недеље је мерио спољну температуру и количину утрошеног гаса за грејање. Мерења су у току грејних сезона 1960. и 1961. године. У подацима постоји и трећа променљива која је категоричка и говори о томе да ли је дато мерење вршено пре постављања изолације на кућу, или после. Учитајмо податке.

```
library(MASS)
podaci <- MASS::whiteside
```

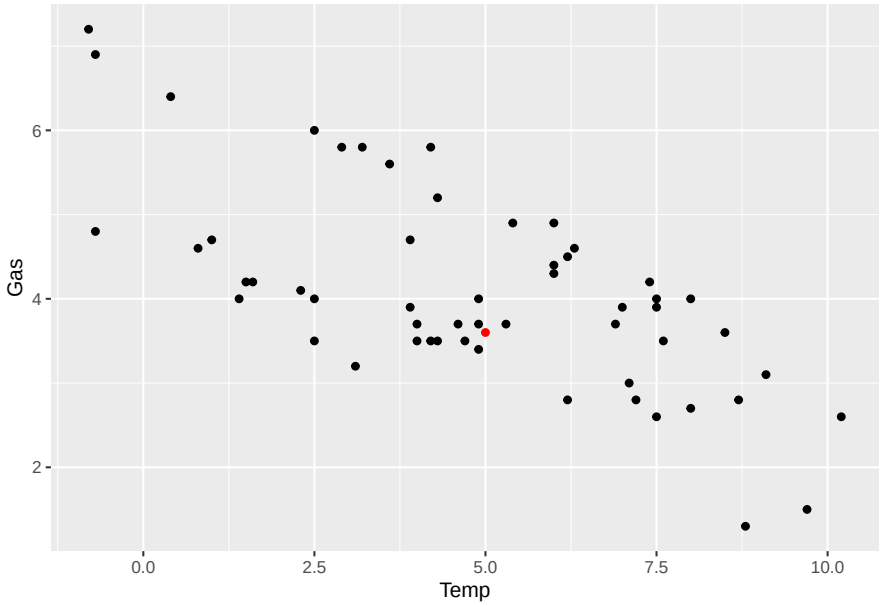
Како су ови подаци комплетни (господин W. је био ревносан), то ћемо формално обрисати количину потрошеног гаса у опсервацији 47. Довољна нам је једна некомплетна опсервација да покажемо шта желимо, а више њих би само оптеретило запис и оштетило концентрацију.

```
podaci_NA <- podaci
podaci_NA$Gas[47] <- NA
```

За почетак нацртајмо податке, без категоричке променљиве.

```
library(ggplot2)

ggplot(podaci_NA, aes(x = Temp, y = Gas))+
  geom_point()+
  geom_point(data = podaci[47, ], color = "red")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



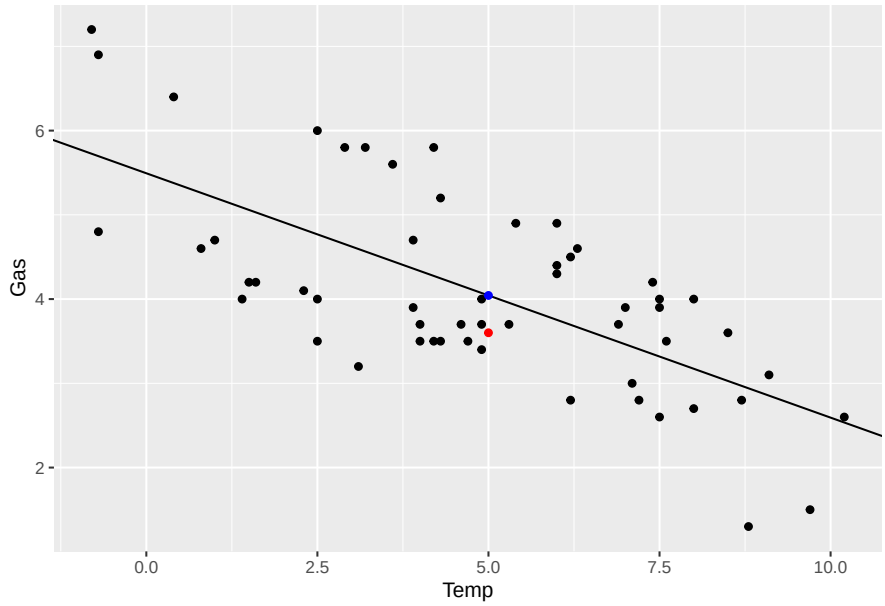
[14] Ја сам присталица руског начина транскрипције страних имена, то јест да се очувава изговор имена, а не сличност записа. Слично бих записао и *Уолт Уитман* уместо *Волт Витман*.

7.7.1.1 Регресиона права

Оно што нам прво пада на памет јесте да на основу доступних опсервација правимо линеарни модел `Gas` $\sim$ `Temp`, те да вредност те праве у 47. тачки узмемо као импутациону.

```
model <- lm(Gas ~ Temp, data = podaci_NA)
koef <- coef(model)
imput <- data.frame(Temp = podaci$Temp[47],
                    Insul = podaci$Insul[47],
                    Gas = koef[1] + koef[2] * podaci$Temp[47])

ggplot(podaci_NA, aes(x = Temp, y = Gas))+
  geom_point()+
  geom_point(data = podaci[47, ], color = "red")+
  geom_abline(intercept = koef[1], slope = koef[2])+
  geom_point(data = imput, color = "blue")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Овај начин импутације може се учинити „најсмисленијим'' и „најправилнијим'', а заправо не може бити даље од тога. Под претпоставком да важи линеарни модел, ова импутирана вредност (на слици плаво), јесте најбоља у средњеквадратном смислу. Међутим, она ни најмање не осликава несигурност коју имамо о недостајућој вредности, те не може бити коришћена као импутациона за вишеструку импутацију. Просто - функција у једној тачки не може узети више од једне вредности - па шта онда узети за импутационе вредности?

Може се учинити да овај метод добро „ради'' за једноструку импутацију, нарочито када је линеарна веза више него уочљива. Међутим, и ту треба бити опрезан. Уколико импутирамо вредношћу са регресионе праве, нарочито ако имамо велики број недостајућих вредности, ми **вештачки повећавамо линеарну везу међу предикторима**. То може значајно пореметити резултате стандардизованих тестова, те показати да су неки предиктори безначајни у присуству других (јер су јако корелисани), иако то нису.

7.7.1.2 Регресиона права + бутстреп + шум

Следећа идеја која нам може пасти на памет јесте да не импутирамо баш вредношћу са регресионе праве, већ да на ту вредност додамо неки случајан шум. Идеално би било да знамо расподелу података око регресионе праве, али ми то наравно не знамо. Та расподела може да се оцени, те да се из ње извуче m вредности које се користе као импутационе.

Изложена идеја је боља, али не сасвим добра. Да смо имали неки други узорак извучен из популације, оцене регресионих коефицијената и стандардне девијације грешака биле би различите. Што је мањи обим узорка, различитост је уочљивија. Стога на неки начин треба и ту варијабилност регресионих параметара укључити у импутациони механизам. Ту постоје два приступа:

1. *Бајесовски.* Извлачи параметре директно из њихове апостериорне расподеле.

2. *Бутстреп (Bootstrap).* Вуче узорке из доступних података и за сваки рачуна коефицијенте.

За оба случаја вучемо m параметара/узорака, и онда на основу њих импутирамо по принципу регресиона права + шум, с тим што се сада праве разликују, а и дисперзија шума јер је различито оцењена (у 2. случају).

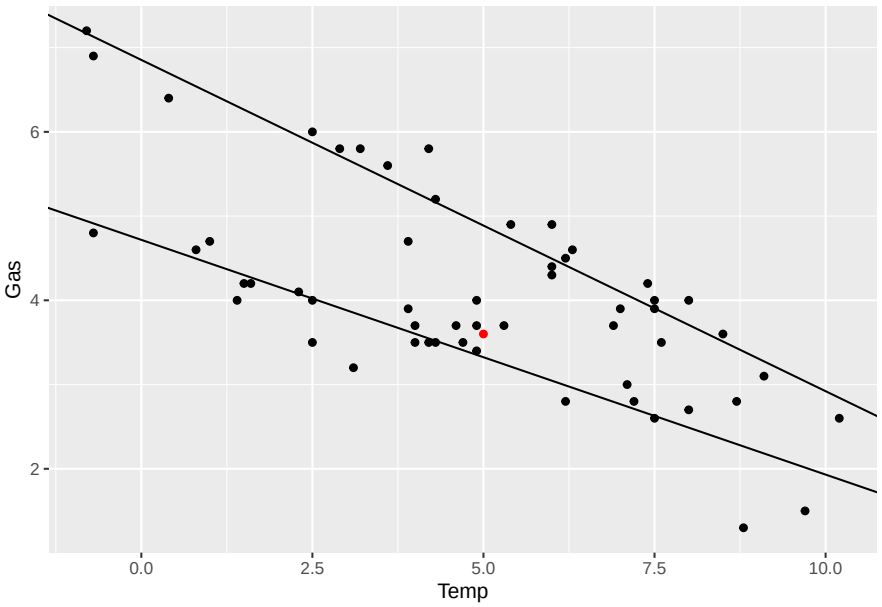
Напомена. Често се, уместо из претпостављене (нормалне) расподеле грешака са оцењеном дисперзијом, шум извлачи узорковањем из резидуала модела. Наравно, претпостављамо да смо извршили потребне трансформације које гарантују барем приближну адекватност линеарног модела.

7.7.1.3 Регресиона права + још предиктора + бутстреп + шум

Следећа финеса коју можемо да додамо јесте још један предиктор, а да претходну причу поновимо са њим. Код нас је уочљиво да права `Gas` $\sim$ `Temp` варира у зависности од категорије променљиве `Insul`, што можемо да видимо и са слике.

```
podaci_NA_before <- podaci_NA[podaci_NA$Insul == "Before", ]
model_before <- lm(Gas ~ Temp, data = podaci_NA_before)
koef_before <- coef(model_before)

podaci_NA_after <- podaci_NA[podaci_NA$Insul == "After", ]
model_after <- lm(Gas ~ Temp, data = podaci_NA_after)
koef_after <- coef(model_after)

ggplot(podaci_NA, aes(x = Temp, y = Gas))+
  geom_point()+
  geom_point(data = podaci[47, ], color = "red")+
  geom_abline(intercept = koef_before[1], slope = koef_before[2])+
  geom_abline(intercept = koef_after[1], slope = koef_after[2])
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Сада се може прво проверити (наравно ако је могуће) у којој је категорији недостајућа опсервација, те, на основу регресионе праве настале из података те категорије, генерисати m импутационих вредности по принципу права + бутстреп (Бајес) + шум.

7.7.1.4 Извлачење из доступних података

Прво провлачимо регресиону праву добијену на основу доступних података, и рачунамо регресионо предвиђање за недостајућу вредност. Затим извршимо предвиђање и за доступне вредности. Бирамо d доступних опсервација таквих да су њихове предвиђене вредности најближе предвиђеној вредности за недостајућу. На случајан начин бирамо једну од тих d и њено поље од интереса импутирамо на упражњено место.

Овај метод је на енглеском језику познат под називом ***Predictive mean matching*** и ми ћемо га у овом раду детаљније анализирати, у случају да више предиктора има недостајуће вредности.

7.7.2 Општи случај

Претходно поглавље посветили смо интуитивном увођењу алгоритма, тако што смо поново прошли кроз мисаони процес који је довео до његовог открића. Сада ћемо мало детаљније изложити алгоритам, испричати у којим се ситуацијама користи, а у којим не, те такође које су му предности и мане.

Име алгоритма, *Predictive mean matching*, предложио је Родерик Ј. А. Литл 1988. године ([10]), а оригинална идеја алгоритма је Рубинова из 1986.

За почетак, претпоставићемо да само једна променљива има недостајуће податке, а да су остале комплетне. Касније ћемо објаснити како се алгоритам може уопштити на податке који имају недостајања у више колона. Означимо променљиву (колону) која има недостајуће податке са Y, а остале, узете заједно као матрицу, означимо са X. Уколико је n обим узорка који имамо, са n_1 ћемо означити број доступних поља у Y, а са X_{obs} ћемо означити оне врсте у X које имају исти индекс као доступне у Y. Аналогно уводимо и n_0 као број недостајућих опсервација у Y и њега прати X_{miss}.

Елем, импутација PMM методом у општем случају спроводи се тако што се прави линеарни модел $Y_{obs} \sim X_{obs}$, те се и за доступне и за недоступне опсервације предвиђају вредности за Y у одговарајућим вредностима предиктора. Затим се, за предвиђену вредност недостајућег поља тражи d најближих. Ово „најближих'' може да се схвати на разне начине, а све у зависности од тога коју смо метрику одабрали.

Литл је у [10] предлагао, између осталог, Махалонобисово растојање[15]. Ипак, пракса је показала да се компликовањем не добијају исувише бољи резултати, па је у функцију `mice.impute.pmm` у пакету `mice` ушла стандардна апсолутна вредност, као једина метрика (не може да се бира).

Тих d најближих опсервација називамо *кандидатима за донора*. Дакле, свако недостајуће поље има својих d кандидата за донора. Затим се на случајан начин од њих бира један, чије се одговарајуће поље узима и импутира на упражњено место. Тај један назива се донор. Прича се понавља за свако недостајуће поље.

Напомена. Кандидати за донора се могу бирати и на друге начине, али ми смо посебно изложили онај који је рачунарски имплементиран. Андриц и Литл ([20]) разликују 4 начина за избор кандидата за донора (па и самог донора) за индекс j:

1. Бирамо неки праг (трешхолд) η и узимамо за доноре све i за које је $|\widehat{y}_i - \widehat{y}_j| < \eta$. Подразумевамо да је $|\cdot|$ она метрика која је одабрана. Кандидати су сви они који задовољавају једнакост, а од њих се на случајан начин бира донор.

2. Бирамо d најближих, и међу њима донора на случајан начин. Ово је имплементирано у `R`-у. Подразумевано је $d = 5$.

3. Као специјални случај претходног, бирамо један најближи. Јасно, он одмах постаје донор. Ово је познато под називом *hot deck* импутација, тако да је она специјални случај PMM импутације.

4. Бирамо само једног донора, дакле без кандидата, али тако што вероватноћа избора зависи од $|\widehat{y}_i - \widehat{y}_j|$. За ово је неопходно претпоставити неку расподелу.

Поред тога што смо видели како се све може одабрати донор (и који је начин имплементиран у `R`-у), корисно је погледати и на који се све начин може вршити упаривање оцена за доступне и недоступне податке, где под упаривањем мислимо на то како бирамо коефицијенте за предвиђање на доступним, а како на недоступним подацима.[16] И овде се могу разликовати 4 типа упаривања:

1. **Упаривање типа 0.** За доступне: $\widehat{Y} = X_{obs}\widehat{\beta}$; за недоступне: $\widehat{Y} = X_{miss}\widehat{\beta}$, где је $\widehat{\beta}$ оцена у моделу $Y_{obs} \sim X_{obs}$.

2. **Упаривање типа 1.** За доступне: $\widehat{Y} = X_{obs}\widehat{\beta}$; за недоступне: $\dot{Y} = X_{miss}\dot{\beta}$, где је $\dot{\beta}$ случајно одабрана вредност параметра из апостериорне расподеле, под условом доступних података. Алтернатива овом бајесовском приступу јесте бутстреп.

3. **Упаривање типа 2.** За доступне: $\dot{Y} = X_{obs}\dot{\beta}$; за недоступне: $\dot{Y} = X_{miss}\dot{\beta}$.

4. **Упаривање типа 3.** За доступне: $\dot{Y} = X_{obs}\dot{\beta}$; за недоступне: $\dot{Y} = X_{miss}\ddot{\beta}$. Овде се врше два извлачења коефицијената из апостериорне расподеле, једно за доноре и једно за примаоце.

Упаривање типа 0 занемарује варијабилност оцене $\widehat{\beta}$ за различите узорке и води ка лошим статистичким закључцима ([21]). Тип 2 покушава да реши овај проблем, мада може направити нови проблем за мали број предиктора, јер ће пречесто да фаворизује исте доноре. Тип 1 представља компромис претходна 2, и он је подразумеван у функцији `mice.impute.pmm`. Други се могу задати променом аргумента `matchtype`.

7.7.2.1 Предности и мане

Основна предност PMM алгоритма јесте та што на известан начин „штити'' механизам од потенцијалне грешке модела. Ако линеарни модел није баш најаднекватнији, ако је „довољно близу'' кандидати за донора ће бити приближно исти. Стога, за разлику од чисто параметарских алгоритама, PMM може да

[15] За дефиницију Махалонобисовог растојања може се погледати [25].

[16] Реч „упаривање" свакако није најадекватнија, али смо је овде задржали да би пратила широко коришћен енглески термин *matching*.

толерише благе грешке у моделовању зависности међу самим подацима. Такође, никада нећемо добити немогуће вредности као кандидата за импутационе.

Основна мана јесте што се у појединим ситуацијама донори понављају за различите недостајуће опсервације, што може смањити варијабилност унутар података и пореметити већину тестова. Такође, PMM треба избегавати када је број предиктора мали.

7.7.3 Алгоритам

Након што смо идејно објаснили како PMM функционише, ред је да дамо и формалан опис алгоритма. Алгоритам је следећи:

1. Срачунати производ $S = X_{obs}^T X_{obs}$.
2. Срачунати $V = (S + \kappa \mathbf{diag}(S))^{-1}$, за неко мало κ. Његова улога је да отклони потенцијалну мултиколинеарност (чија је последица то да нема инверз).
3. Спроводимо назубљену (ridge) линеарну регресију $Y_{obs} \sim X_{obs}$. Рачунамо регресионе коефицијенте $\widehat{\beta} = V X_{obs}^T Y_{obs}$.
4. Нека је q број колона у X_{obs}. Извлачимо $\dot{g} \sim \chi^2_\nu$, где је $\nu = n_1 - q$.
5. Рачунамо $\dot{\sigma}^2 = (Y_{obs} - X_{obs}\widehat{\beta})^T (Y_{obs} - X_{obs}\widehat{\beta})/\dot{g}$.
6. Вучемо q независних вредности из нормалне $\mathcal{N}(0,1)$ расподеле и смештамо их у вектор $\dot{z}_1$.
7. Рачунамо $V^{1/2}$ преко Чолески декомпозиције.
8. Рачунамо $\dot{\beta} = \widehat{\beta} + \dot{\sigma}\dot{z}_1 V^{1/2}$.
9. Рачунамо $\dot{\eta}(i,j) = |X_{obs,i\to}\widehat{\beta} - X_{miss,j\to}\dot{\beta}|$, за све одговарајуће i, j.
10. Конструишемо n_0 вектора Z_j, при чему сваки садржи d „донора кандидата" из Y_{obs}, тако да је $\sum_d \dot{\eta}(i,j)$ минимално.
11. Из сваког Z_j бирамо донора i_j на случајан начин.
12. Импутирамо $y_j = y_{i_j}$, за свако $j \in \{1, 2, \ldots, n_0\}$.

7.7.3.1 Вишеструка импутација

Вишеструка импутација се добија тако што се део алгоритма од 4. корака па до краја понови m пута, те поступи по Рубиновим правилима.

7.7.4 Недостајања у више колона

До сада смо претпостављали да су наши подаци облика (Y, X) и да недостајања постоје само у Y. Сада треба пронаћи начин како да алгоритам уопштимо и на случајеве када недостајања постоје у више колона. Претпоставимо, зато, да су наши подаци облика (Y, X), али да је Y вишедимензион. Слично као и раније, вршимо линеарну регресију $Y_{obs} \sim X_{obs}$, а даље тражење донора настављамо као и малопре ([10]).

Напомена. Што је више колона у Y, а мање у X, резултати су (наравно) лошији, те се овај алгоритам и не препоручује у таквим ситуацијама. Он ће потпуно „пући" ако свака колона у подацима има барем једну недостајућу опсервацију. Тада треба посегнути за неким другим алгоритмом. Када би постојао савршен механизам импутације, онда би постојао тачно један механизам импутације.

7.8 PMM - Провера на подацима

7.8.1 Генерисање недостајања

Генерисање недостајања по MCAR принципу не би требало да представља велики проблем. Сетимо се да је у MCAR случају расподела матрице R независна и од недостајућих података и од доступних података. Сходно томе, довољно је генерисати примерак матрице R из одговарајуће расподеле (која се, поново, бира емпиријски, на основу додатних сазнања) и онде где се реализовала нула уписати `NA`. За велику већину расподела програмски језик `R` већ има уграђене функције за генерисање.

Међутим, генерисање MAR података неће ићи тако лако. Основни проблем лежи у томе што у MAR случају расподела матрице R зависи не само од унутрашњих параметара, већ и од доступних нам података. Та зависност може се остваривати на бесконачно много начина, а ми ћемо овде изложити неке од њих. За детаљније разматрање генерисања недостајућих података вреди погледати [21].

7.8.1.1 Показни пример

Као и обично, најбоље се види на примеру. Кренућемо од најједноставнијег случаја. Нека се наша дизајн матрица $X = (X_1, X_2)$ састоји од 1 000 узорака из дводимензионе нормалне расподеле $\mathcal{N}\left([3\ 3], \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix}\right)$.

```
set.seed(123)
n <- 1000
covMat_X1X2 <- matrix(c(1, 0.6,
                        0.6, 1),
                      nrow = 2)
library(MASS)
X <- mvrnorm(n, mu = c(3, 3), Sigma = covMat_X1X2)
```

За почетак ћемо генерисати недостајања **само у колони** X_2 по MAR принципу, и даћемо им да зависе од вредности у колони X_1. То се може урадити на више начина, а ми ћемо представити три која су у пракси од значаја. Нека је R_2 колона-индикатор недостајања у X_2 (у аналогији с матрицом R коју смо на почетку помињали). Дајемо три модела:

1. **MARRIGHT**: $\text{logit}(\mathbf{P}\{R_2 = 0\}) = X_1 - 3$
2. **MARMID**: $\text{logit}(\mathbf{P}\{R_2 = 0\}) = 0.75 - |X_1 - 3|$
3. **MARTAIL**: $\text{logit}(\mathbf{P}\{R_2 = 0\}) = |X_1 - 3| - 0.75$

Напомена. Бројеви су одабрани да би неке особине касније биле боље уочљиве на сликама - нису од пресудне важности.

Имена која смо им доделили ускоро ће постати јасна, а сада их прво применимо на нашим подацима. У `r-base` пакету `stats` дате су функције за рад са логистичком расподелом. `plogis` представља функцију расподеле вероватноћа логистичке расподеле, а она је ништа друго до инверз logit функције. Радимо прво са MARRIGHT моделом.

```
prob_R2jeNula_MARRIGHT <- plogis(X[, 1] -3)
prob_R2jeJedan_MARRIGHT <- 1 - prob_R2jeNula_MARRIGHT
```

У вектору `prob_R2jeJedan_MARRIGHT` су вероватноће да је одговарајуће поље присутно, тј. да није недостајуће. То нам одговара јер ћемо у зависности од тих вероватноћа да генеришемо вектор R_2. Урадимо то.

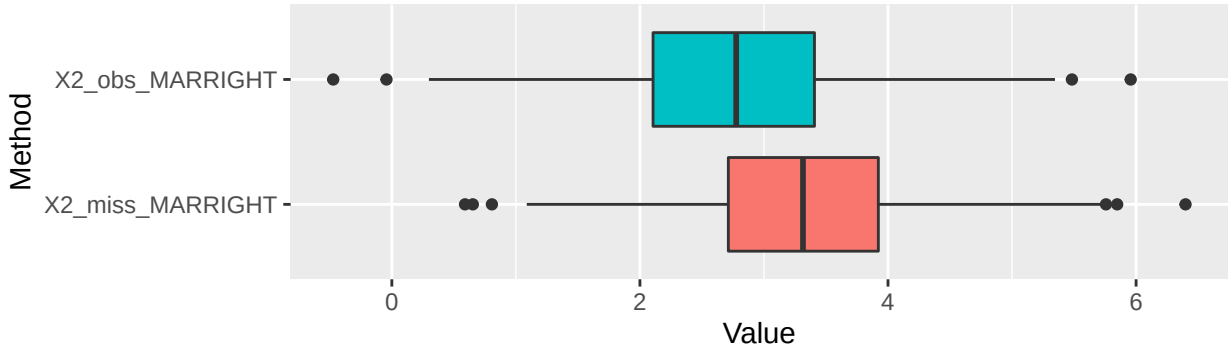
```
R2_MARRIGHT <- rbinom(n, 1, prob_R2jeJedan_MARRIGHT)
```

Сада ћемо раздвојити недостајуће и доступне податке.

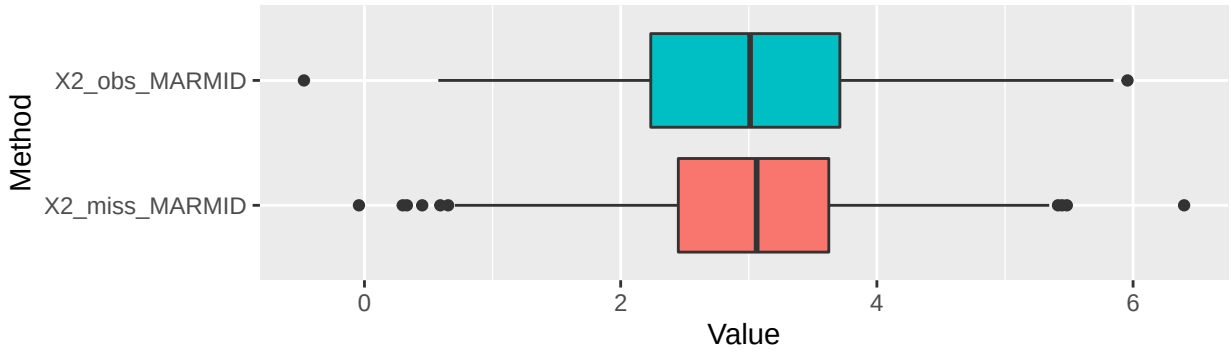
```
X2 <- X[, 2]
# kolona dostupnih vrednosti
X2_obs_MARRIGHT <- X2[R2_MARRIGHT == 1]
# kolona nedostajucih vrednosti
X2_miss_MARRIGHT <- X2[R2_MARRIGHT == 0]
```

На потпуно аналоган начин радимо и за MARMID и MARTAIL, разликује се само аргумент у првом реду.

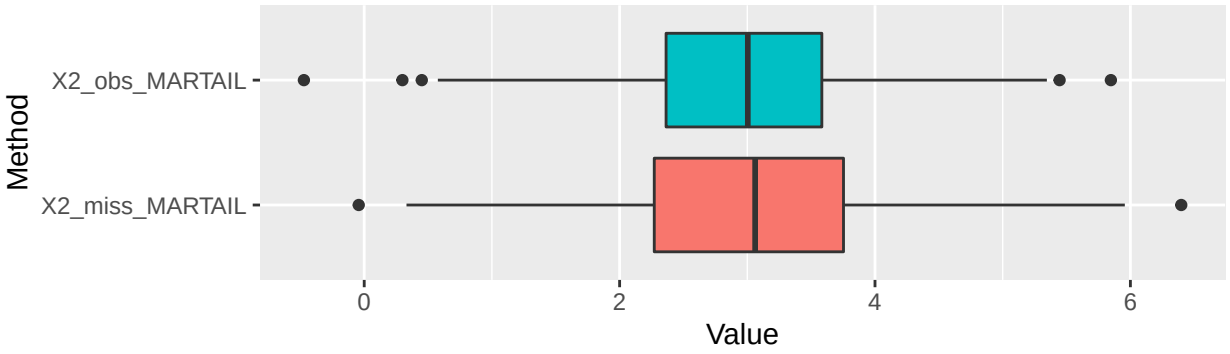
Сада ћемо нацртати боксплотове за сваки од претходних парова недостајућих и доступних података и постаће нам јасно откуд назив. На слици испод видимо боксплотове за доступне и недостајуће податке MARRIGHT модела. Јасно је да он „одузима'' више података за више вредности поља, тако да се вектор доступних вредности помера улево.



На следећој слици видимо боксплотове за MARMID модел. Он узима више података из околине средње вредности, тако да средња вредност остаје иста, али дисперзија доступних података расте.



Код MARTAIL модела видимо супротну ствар - одузима се више података далеких од средње вредности, тако да дисперзија оног што је остало опада.



Сваки од претходна три модела даје нам апроксимативно 50% недостајућих података, али сваки на свој начин. Ови модели недостајања су прилично екстремни и ретко се срећу на реалним подацима. Не само да имамо јаку везу између R_2 и X_1, него је и удео недостајућих података нереално висок за скоро па све реалне ситуације. Ипак, наравно да их нисмо узалуд наводили. Уколико се неки метод импутације добро покаже на овако екстремно оштећеним подацима, разумно је дати себи за право помислити да је тај метод квалитетан и да ће се понашати добро и у мање екстремним ситуацијама.

Са друге стране, прилично смо се ограничили. Имамо недостајање података само у једној колони, а уколико желимо недостајања и у другој, можемо то урадити само за оне индексе за које је у претходној податак доступан. Другим речима, за сада не умемо да генеришемо више недостајућих поља у врсти. Такође, не можемо директно да утичемо на проценат недостајућих података, а било би нам згодно када бисмо га могли проследити некој функцији као аргумент. Ствар се додатно компликује када у причу укључимо произвољан број колона. Срећом, и за то постоји решење, које ћемо дати у следећем пододељку.

7.8.1.2 Уопштење

Бранд је 1999. године у [13] је развио метод за генерисање MAR података за дизајн матрицу са p колона $X_1, X_2, \dots, X_p$ који ћемо ми сада изложити. Претпостављамо да је $X = (X_1, X_2, \dots, ..., X_p)$ на почетку комплетна. Овај метод захтева задавање следећих ствари:

- α, жељени удео недостајућих података;
- R_{pat}, бинарна матрица димензија $n_{pat} \times p$ која дефинише n_{pat} дозвољених шаблона по којима недостају подаци у врстама; дозвољени су сви осим свих нула и свих јединица;
- $f = (f_{(1)}, \dots, f_{(n_{pat})})$, вектор који садржи релативне френвенције појављивања сваког од шаблона;
- $\mathbf{P}\{R \mid X\} = \big(\mathbf{P}\{R_{(1)} \mid X_{(1)}\}, \dots, \mathbf{P}\{R_{(n_{pat})} \mid X_{(n_{pat})}\}\big)$, низ од n_{pat} модела недостајања, сваки за по један шаблон.

Напомена. У пакету `mice` постоји функција `ampute` која имплементира три горе наведена начина недостајања, као и MARLEFT у аналогији са MARRIGHT. Њен аргумент `type` може узети четири вредности: `"MID", "TAIL", "LEFT", "RIGHT"`. Подразумевана пропорција недостајања је 50%, а може се мењати аргументом `prop`.

7.8.2 Поређење са регресијом

Прави је моменат да направимо кратку рекапитулацију свега што смо до сада обрадили. Дефинисали смо типове недостајања података, детаљно смо прошли кроз идеју вишеструке импутације, те затим цело једно поглавље посветили PMM алгоритму. Објаснили смо зашто је он бољи од неких класичних метода, на пример регресије. Сада ћемо целу причу илустровати једним примером.

Треба нагласити да се од вишеструке импутације углавном не очекује да да најближе оцене недостајућих поља. То је најбоље радити регресијом, или неком другом параметарском методом, јер је познато да су те оцене, под претпоставком линеарног модела, најбоље у средњеквадратном смислу. Од вишеструке импутације се првенствено очекује да не наруши структуру података: варијабилност, предвидивост (тј. понављање шаблона унутар података) итд.

Генерисаћемо један вештачки пример, вишедимензиону нормалну расподелу димензије 6, где недостајања постоје у прве две колоне.

```
data <- mvrnorm(n = 1000,
                mu = c(0, 1, 2, 3, 4, 5),
                Sigma = diag(1, 6))
data <- data.frame(data)
colnames(data) <- c("V0", "V1", "V2", "V3", "V4", "V5")
```

Видимо да смо податке генерисали тако да је коваријациона матрица дијагонална, односно да су колоне независне. То смо урадили да би чињеница да регресија нарушава структуру података била још уочљивија. Генерисаћемо MAR недостајања пропорције 40%, по TAIL принципу. Слични резултати се добијају и за друге типове и пропорције недостајања, што је лако проверљиво, те нећемо оптерећивати запис.

```
data_Y <- data[, 1:2]

data_Y_amputed <- ampute(data_Y, prop = 0.4, mech = "MAR", type = "TAIL")$amp

data_incompl <- cbind(data_Y_amputed, data[, 3:6])
colnames(data_incompl) <- c("V0", "V1", "V2", "V3", "V4", "V5")

data_imput_pmm <- complete(mice(data_incompl)) # PMM je default
```

Сада ћемо исто урадити и за регресиону импутацију.

```
data_imput_reg <- complete(mice(data_incompl, method = "norm.predict"))
```

Сада ћемо цртати зависност колона `V0` и `V2` за оба метода. Бирамо једну колону која је имала недостајања и једну која није, јер је свеједно пошто су независне. Да бисмо импутиране вредности обојили различитом бојом од доступних, имаћемо мало физичког посла.

Прво ћемо из колоне `V0` у оштећеним подацима извући индикатор недостајања.

```
ind_fali <- is.na(data_incompl$V0)
ind_ne_fali <- as.logical(rep(1, length(data_incompl$V0)) - ind_fali)
```

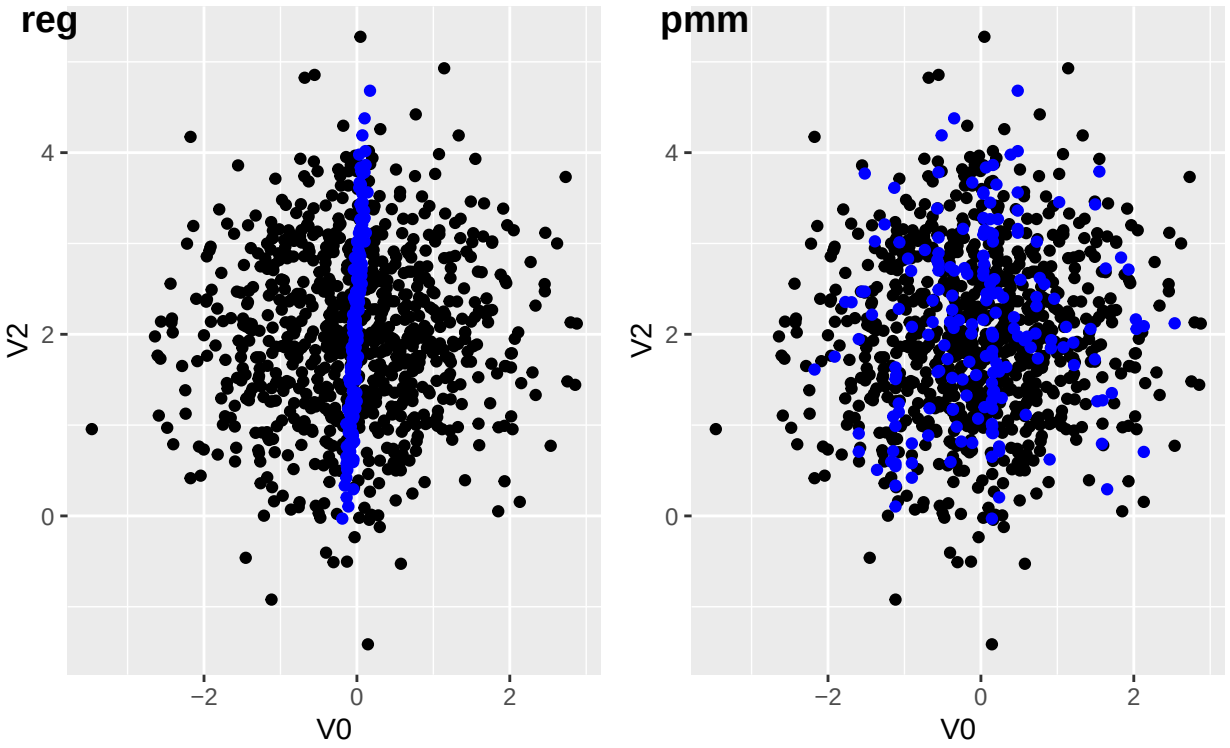
Сада из импутираних база издвајамо посебно врсте које су биле комплетне, а посебно оне које су импутиране.

```
V0_V2_reg_obs <- data_imput_reg[ind_ne_fali, c(1, 3)]
V0_V2_pmm_obs <- data_imput_pmm[ind_ne_fali, c(1, 3)]

V0_V2_reg_imp <- data_imput_reg[ind_fali, c(1, 3)]
V0_V2_pmm_imp <- data_imput_pmm[ind_fali, c(1, 3)]
```

Сада смо спремни да цртамо. Плавом бојом означавамо импутиране вредности, а црном доступне.

```
library(cowplot)

plot1 <- ggplot() +
```

```
  geom_point(data = V0_V2_reg_obs,
             mapping = aes(x = V0, y = V2)) +
  geom_point(data = V0_V2_reg_imp,
             mapping = aes(x = V0, y = V2),
             color = "blue")

plot2 <- ggplot() +
  geom_point(data = V0_V2_pmm_obs,
             mapping = aes(x = V0, y = V2)) +
  geom_point(data = V0_V2_pmm_imp,
             mapping = aes(x = V0, y = V2),
             color = "blue")

cowplot::plot_grid(plot1, plot2, labels = c("reg", "pmm"))
```



Више је него очигледно колико *Predictive mean matching* више очувава структуру података од класичне регресије (сама регресија и нема баш смисла за независне колоне, овде је узета чисто показно). Намерно смо одабрали независне колоне да би то било што уочљивије, а, наравно, исто важи и за зависне колоне. Ко жели и у то да се увери, може погледати [21], а можда и боље, поиграти се сам.

„Нарушавање структуре података'' може се мерити и разним информационим показатељима, од којих је у последње време популаран `ApEn` (approximate entropy), о којему се може прочитати у [22]. На српском језику о овом показатељу може се прочитати у [26], а и видети његова примена на анализу временских серија.

Вежбе 8

Обрада слика у R-у

Напомена. Делимично преузето из [6].

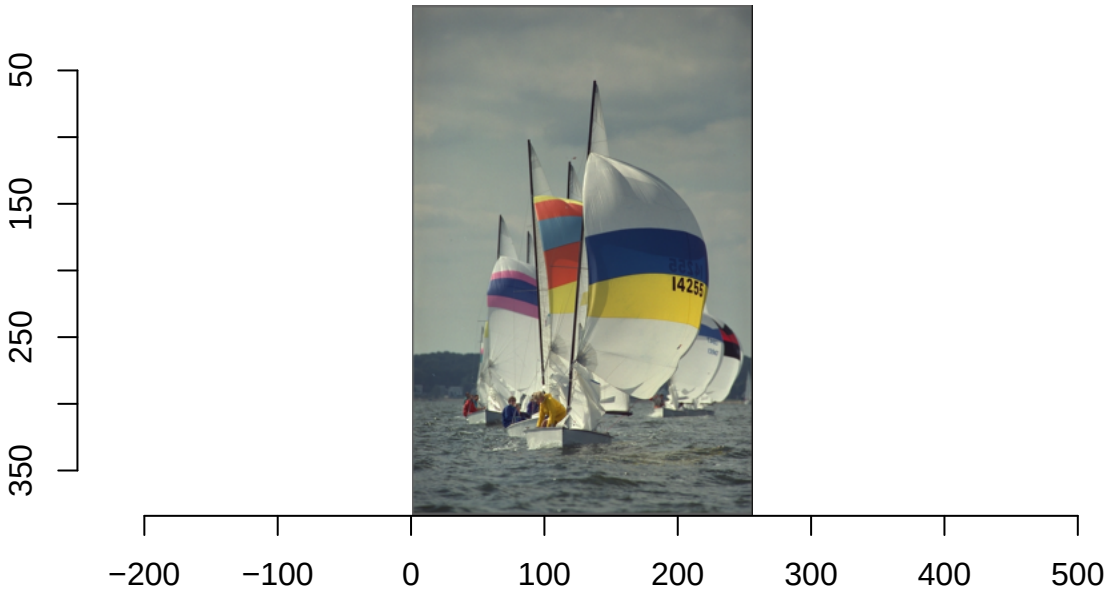
У програмском језику `R` постоји пакет `imager` који нуди мноштво корисних алата за обраду слика. Већина функција потиче из библиотеке `CImg` чији је аутор David Tschumperlé.

```
library(imager)
```

8.1 Учитавање и приказ слика

У овом пакету постоји уграђена слика чамаца. Погледајмо.

```
plot(boats)
```



Да смо позвали `display(boats)` слика би се отворила у посебном прозору. Уочимо и занимљиву чињеницу: ипсилон оса расте одозго надоле. То је традиционалан начин одабира координатног почетка у горњем левом углу слике и пакет `imager` га се постојано држи.

Свака слика је класе `cimg`.

```
class(boats)
```

```
## [1] "cimg"         "imager_array" "numeric"
```

Да бисмо добили нешто више информација о слици можемо укуцати

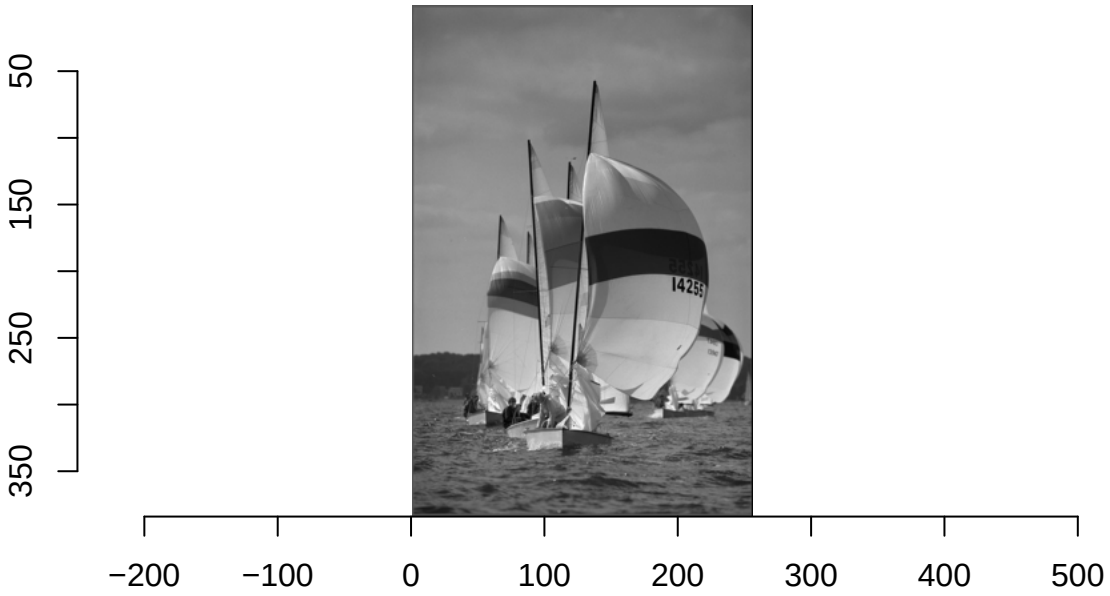
```
boats
```

```
## Image. Width: 256 pix Height: 384 pix Depth: 1 Colour channels: 3
```

Видимо да слика има ширину од 256 пиксена и висину од 384 пиксела. Видимо да је дубина слике једнака 1. Шта то значи? То значи да слика има само један фрејм. Да је дубина била број већи од 1, то би значило да је у питању заправо видео. Последњи број који нам је исписан јесте број канала боје. Ова слика дата је у RGB систему, те има три канала боје: црвена (Red), зелена (Green) и плава (Blue).

Слику врло лако можемо направити црно-белом.

```
plot(grayscale(boats))
```



Уколико је сада испишемо, видимо да је број канала једнак 1, јер имамо само интензитет црне боје.

```
grayscale(boats)
```

```
## Image. Width: 256 pix Height: 384 pix Depth: 1 Colour channels: 1
```

Објекат класе `cimg` је заправо само танак интерфејс за обичан низ дужине 4. То видимо позивом:

```
dim(boats)
```

```
## [1] 256 384   1   3
```

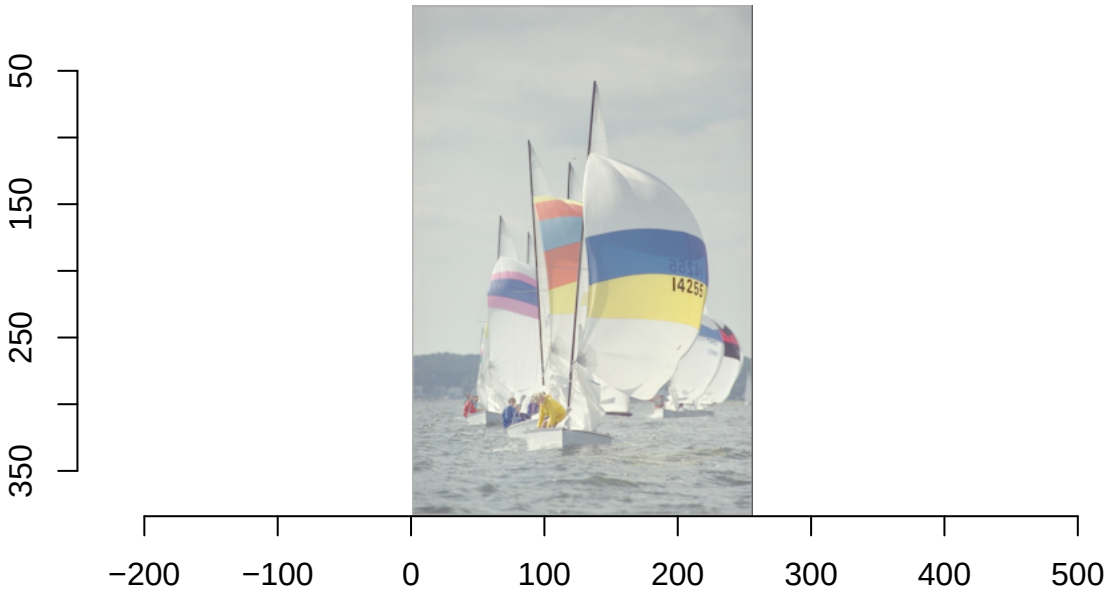
Још ћемо касније да видимо како се тачно слике чувају. За сада ћемо још мало њима да се играмо. Рецимо, коректан је израз

```
log(boats) + 3 * sqrt(boats)
```

```
## Image. Width: 256 pix Height: 384 pix Depth: 1 Colour channels: 3
```

Видимо да су се димензије очувале. Да видимо шта буде када их нацртамо.

```
plot(log(boats) + 3 * sqrt(boats))
```



Боје су се мало промениле. Можемо да урадимо и следеће

```
mean(boats)
```

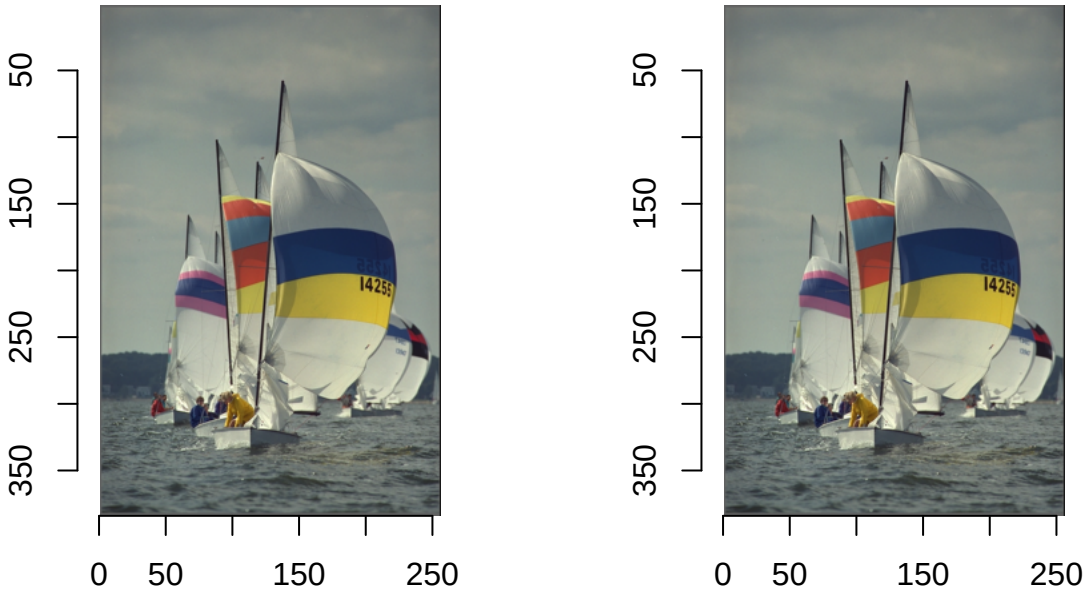
```
## [1] 0.5089061
```

```
sd(boats)
```

```
## [1] 0.144797
```

Можемо се запитати: зашто наредне две слике изгледају потпуно идентично:

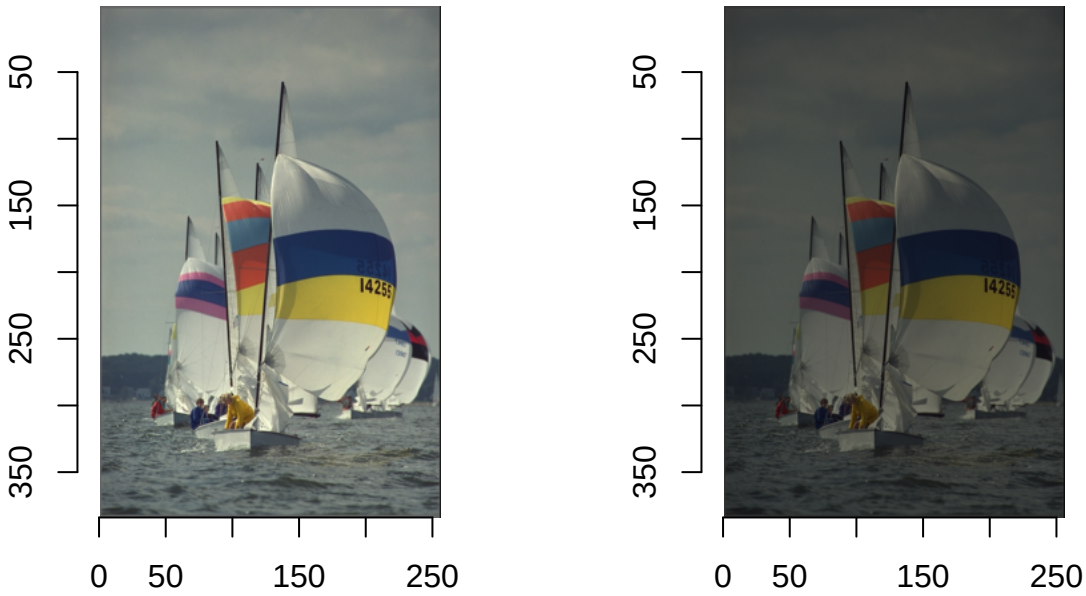
```
layout(t(1:2))
plot(boats)
plot(boats/2)
```



Разлог за то је тај што функција `plot` аутоматски рескалира слику тако да је цео распон боја искоришћен. Један од разлога за то је тај што не постоји универзалан начин за запис RGB вредности: `CImg` користи 0-255 (тамно ка светлом), док `R` функција `rgb` користи распон од 0 до 1. Други разлог је тај што је често згодно радити са сликом чија је средња вредност нула (шта год то за сад значило).

Уколико не желимо да до тога дође, мењамо аргумент `rescale`.

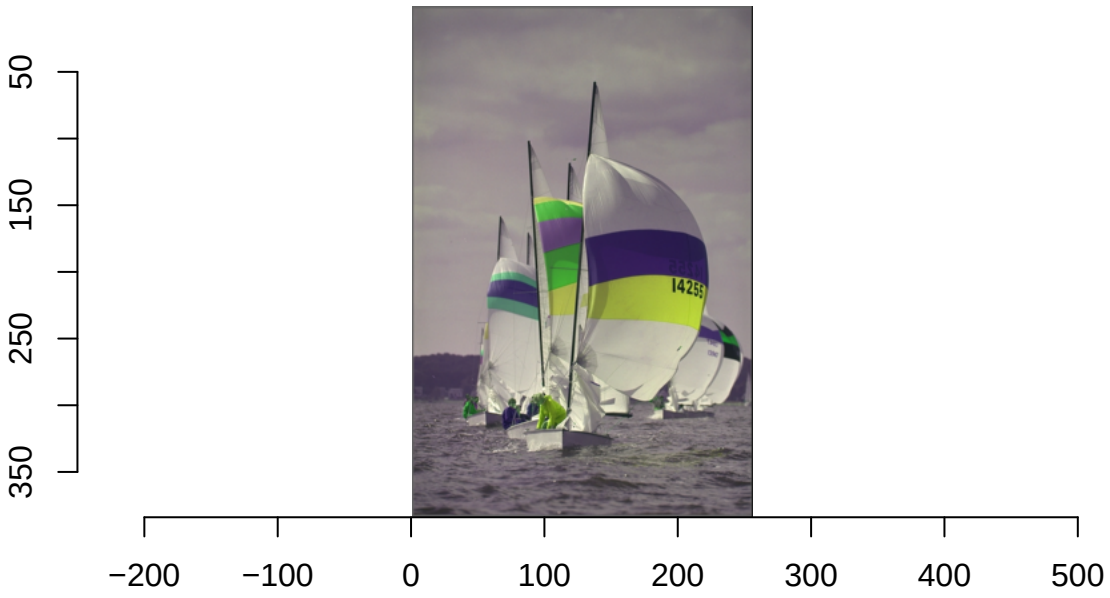
```
layout(t(1:2))
plot(boats,rescale=FALSE)
plot(boats/2,rescale=FALSE)
```



Сада видимо јасну разлику.

У `R`-у постоји функција `rgb` преко које можемо контролисати боје слике. Рецимо, можемо обрнути црвену и зелену.

```
cscale <- function(r,g,b) rgb(g,r,b)
plot(boats, colourscale = cscale, rescale = FALSE)
```



Видимо да су се црвена и зелена обрнуле. Слично се можемо играти и са црно-белим сликама. Наредни код grayscale приказује као bluescale.

```
library(plyr)
```

```
##
## Attaching package: 'plyr'
```

```
## The following object is masked from 'package:imager':
##
##     liply
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

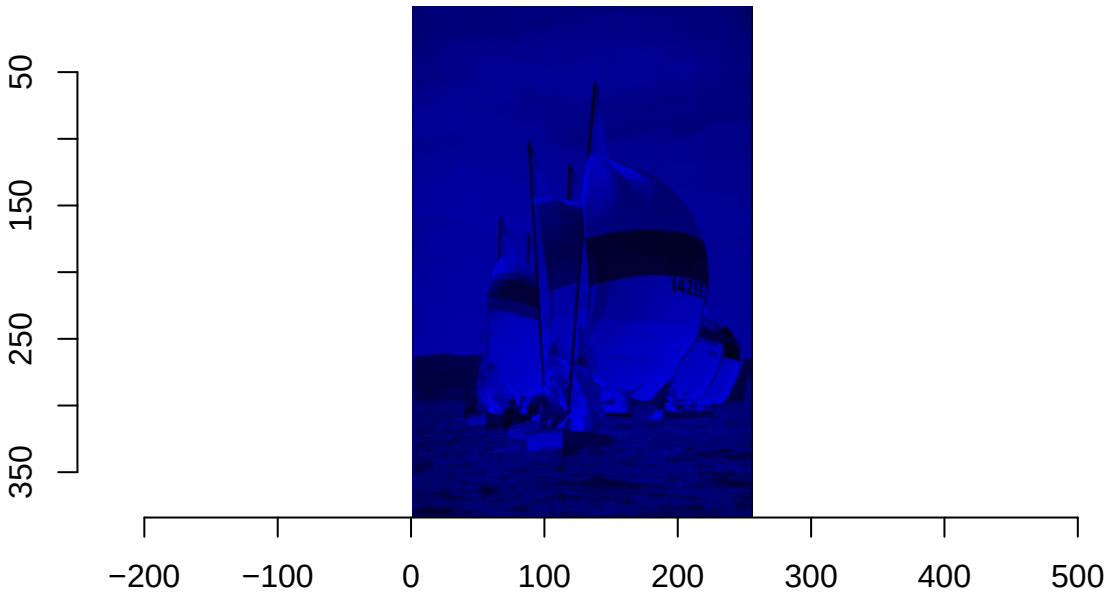
```
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## 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
```

```
cscale <- function(v) rgb(0,0,v)
grayscale(boats) %>% plot(colourscale=cscale,rescale=FALSE)
```



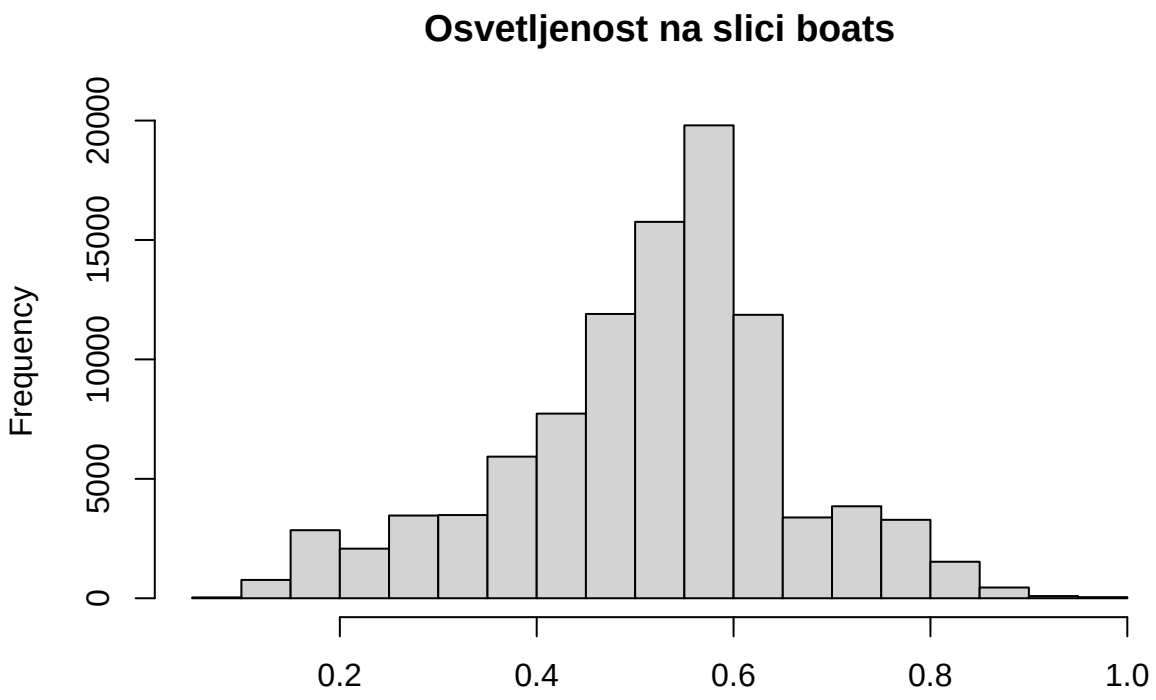
Овај пакет нативно подржава JPEG, PNG и BMP, а за остале формате треба инсталирати `ImageMagick`.

8.2 Пример 1: Хистограмско уједначење

Хистограмско уједначење је уџбенички пример филтера за побољшање контраста. Такође је и добар увод у то шта све може да се ради са `imager`-ом, па ћемо га обрадити.

Хистограм слике је ништа друго до хистограм вредности свих појединачних пиксела. Погледајмо.

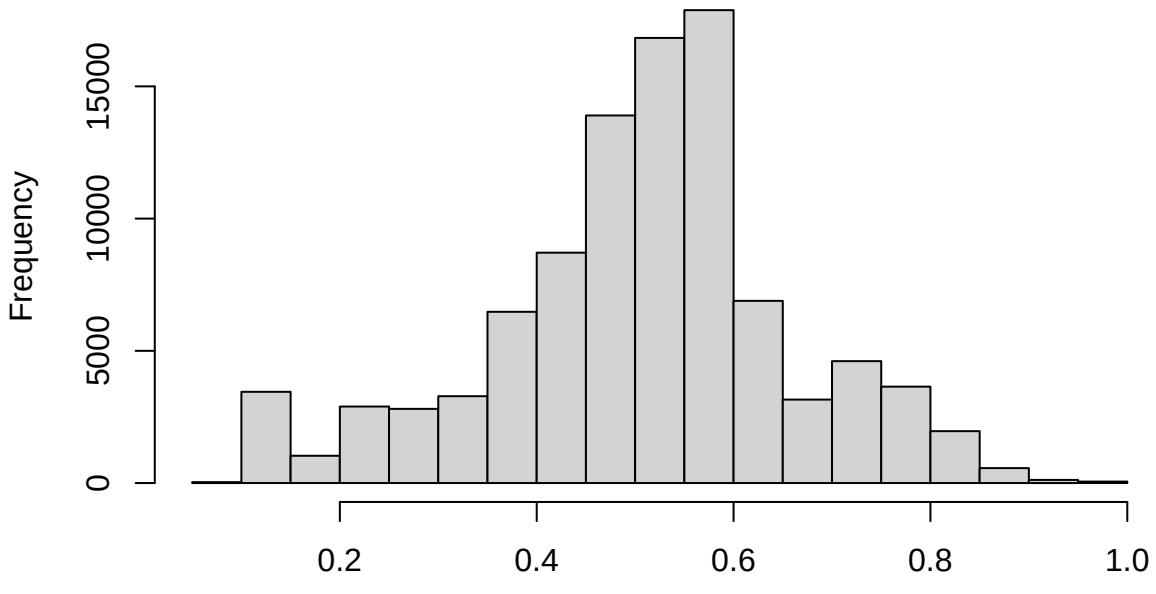
```
grayscale(boats) %>% hist(main="Osvetljenost na slici boats")
```



.

Дакле, R р слике чува као низове вредности пиксела. Када слика није црно-бела, тада је то заправо низ трију вредности пиксела. Ако бисмо желели хистограм, рецимо, само црвеног канала, позвали бисмо

```
R(boats) %>% hist(main="Crveni kanal na slici boats")
```



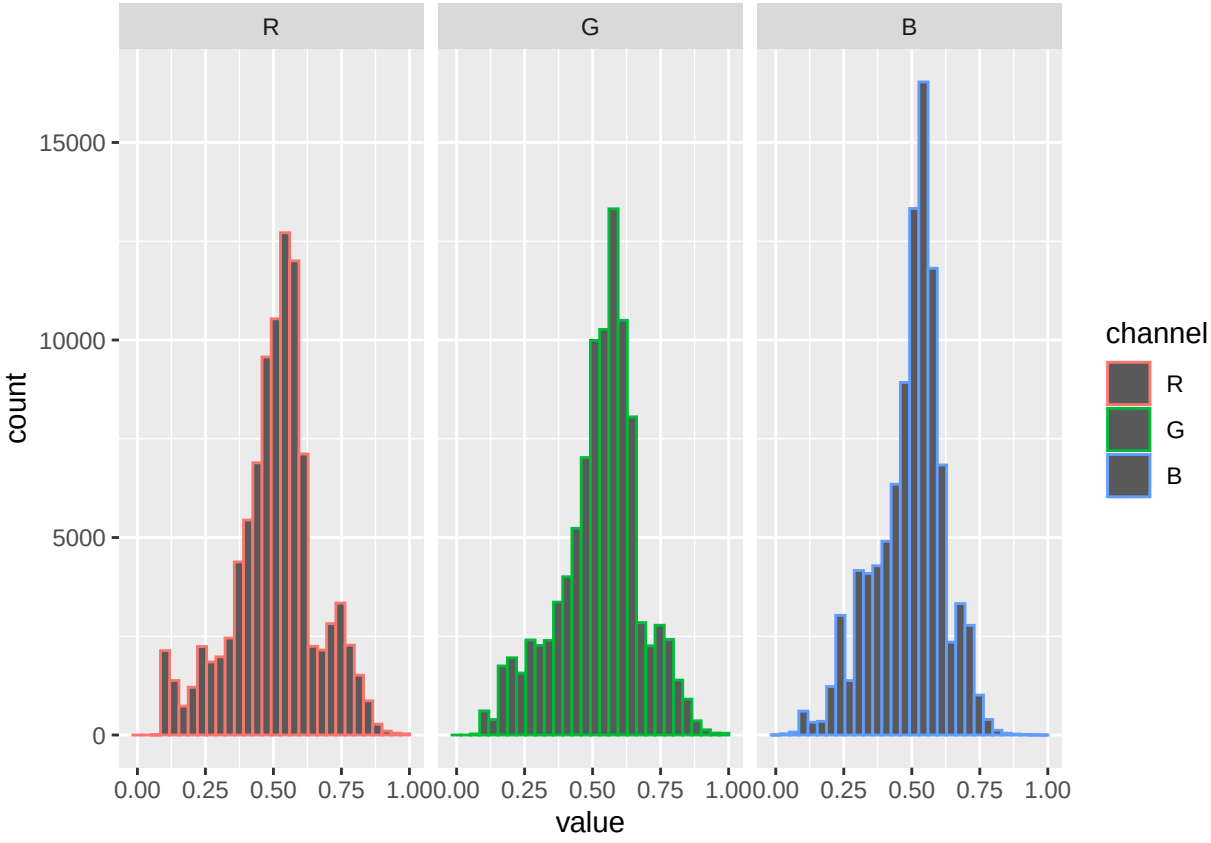
.

Други приступ би био да слику претворимо у `data.frame`, па да цртамо све канале одједном.

```
library(ggplot2)
bdf <- as.data.frame(boats)
head(bdf,3)
```

```
##   x y cc     value
## 1 1 1  1 0.3882353
## 2 2 1  1 0.3858633
## 3 3 1  1 0.3849406
```

```
bdf <- plyr::mutate(bdf,channel=factor(cc,labels=c('R','G','B')))
ggplot(bdf,aes(value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)
```

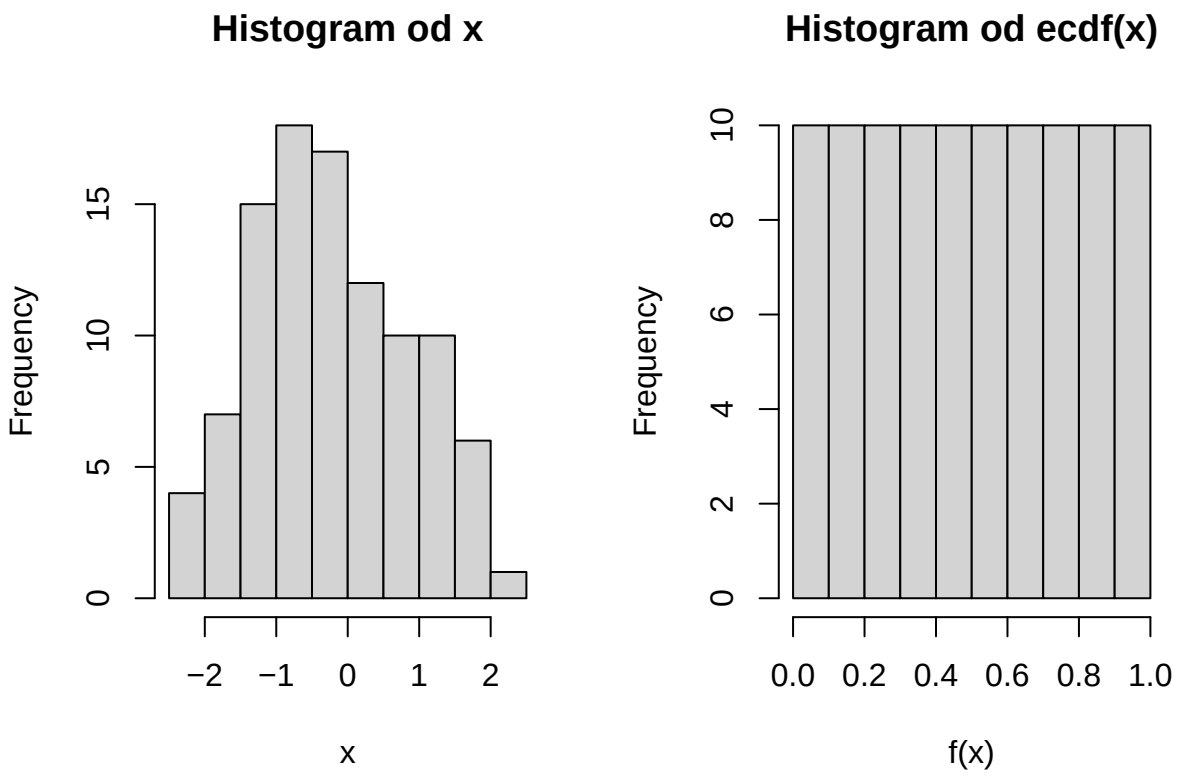


Оно што одмах видимо са ових хистограма јесу пренаглашене средње вредности, види се да су те вредности сваке од боја некако превише коришћене.

Хистограмско уједначење решава овај проблем. Вредност сваког пиксела мења се његовим РАНГОМ (редним бројем у узорку). То је суштински исто као када бисмо узорак пропустили кроз његову емпиријску функцију расподеле, јер она управо то ради.

Илустроваћемо прво на баналном примеру.

```
x <- rnorm(100)
layout(t(1:2))
hist(x,main="Histogram od x")
f <- ecdf(x)
hist(f(x),main="Histogram od ecdf(x)")
```



Потпуно је уравнало хистограм. Слично ћемо урадити и са црно-белом верзијом наше слике.

```
boats.g <- grayscale(boats)
f <- ecdf(boats.g)
```

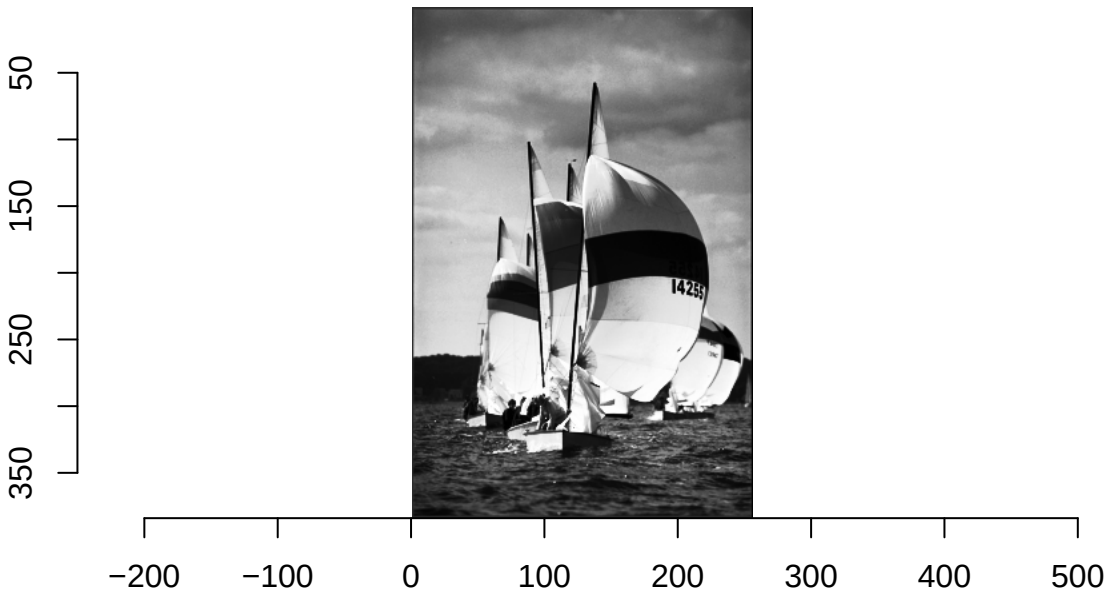
Проблем се јавља у томе што је `ecdf` из основног пакета, и не може да разликује да не ради баш са обичним вектором, већ са сликом.

```
glimpse(f(boats.g))
```

```
##  num [1:98304] 0.171 0.165 0.163 0.164 0.165 ...
```

Видимо да је вратило обичан нумерички вектор. Срећом, ово је лако превазићи, коришћењем `as.cimg`.

```
f(boats.g) %>% as.cimg(dim=dim(boats.g)) %>% plot(main="Sa histogramskim ujednacenjem")
```



Ок, за сада знамо (веома грубо) да изоштримо црно-белу слику. Изоштрење на слици у боји иде слично, с тим што сваки канал треба пропустити кроз хистограмско уједначење. Прво ћемо само уједначење упаковати као функцију.

```
hist.eq <- function(im) as.cimg(ecdf(im)(im),dim=dim(im))
```

Сада ћемо употребити функцију `imsplit` да поделимо слику по каналима.

```
cn <- imsplit(boats,"c")
cn
```

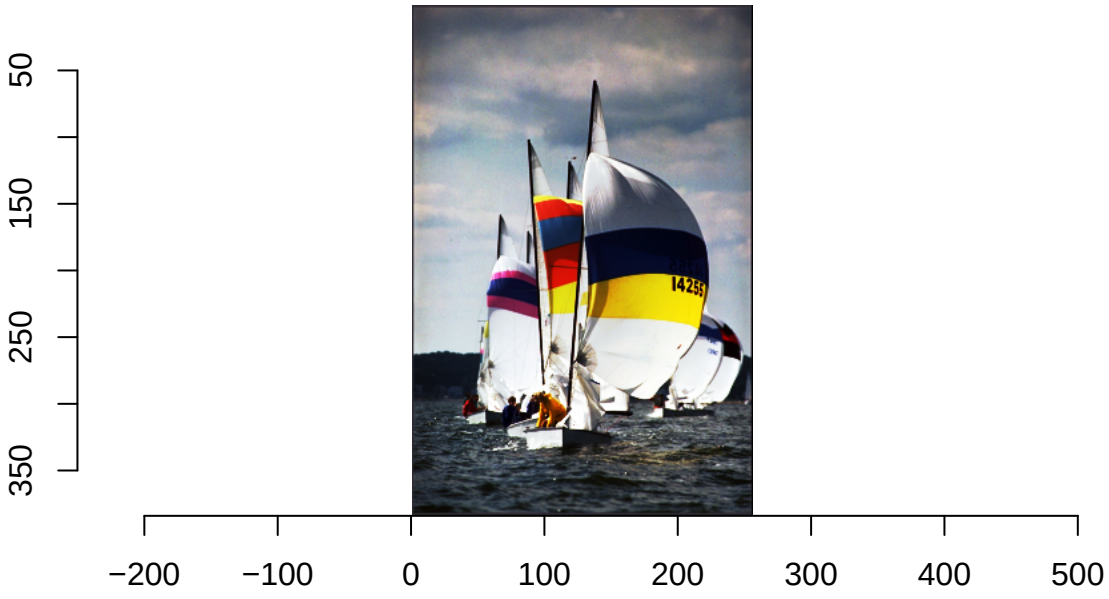
```
## Image list of size 3
```

Сада имамо листу од три слике, свака за одговарајући канал. Уједначавамо.

```
cn.eq <- lapply(cn,hist.eq)
```

Сада комбинујемо назад и приказујемо изоштрену слику са уједначеним свим каналима.

```
plot(imappend(cn.eq, "c"))
```



8.3 Пример 2: Детекција ивица

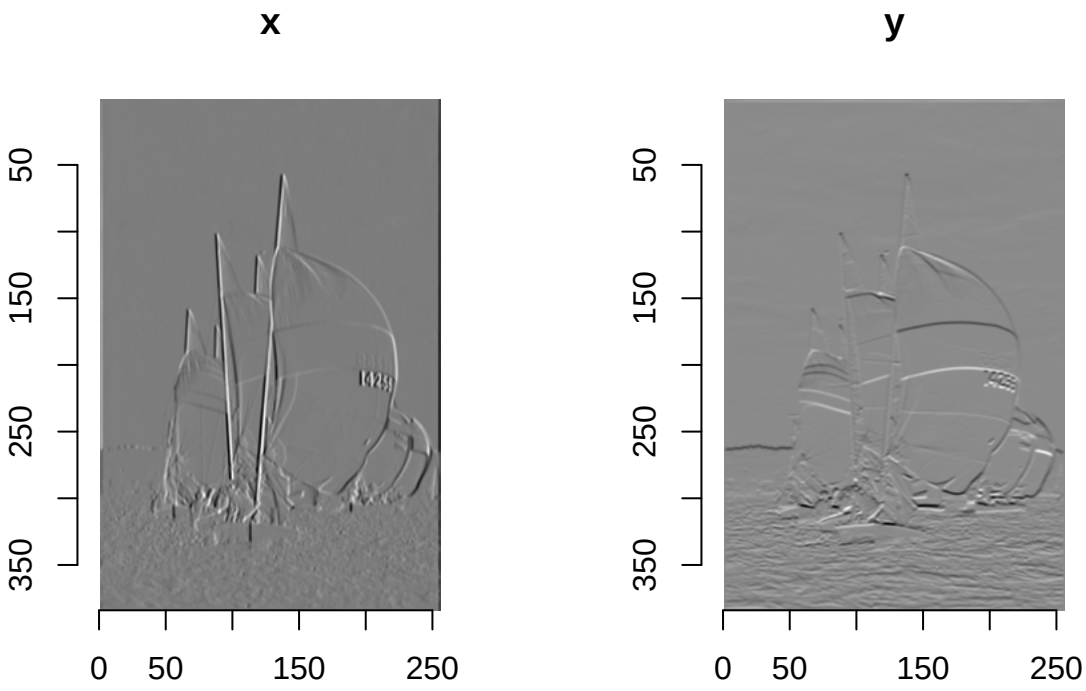
Хајде прво да одрадимо, па да објаснимо шта је шта.

```
gr <- imgradient(boats.g, "xy")
gr
```

```
## Image list of size 2
```

Добили смо листу од две слике.

```
plot(gr, layout = "row")
```

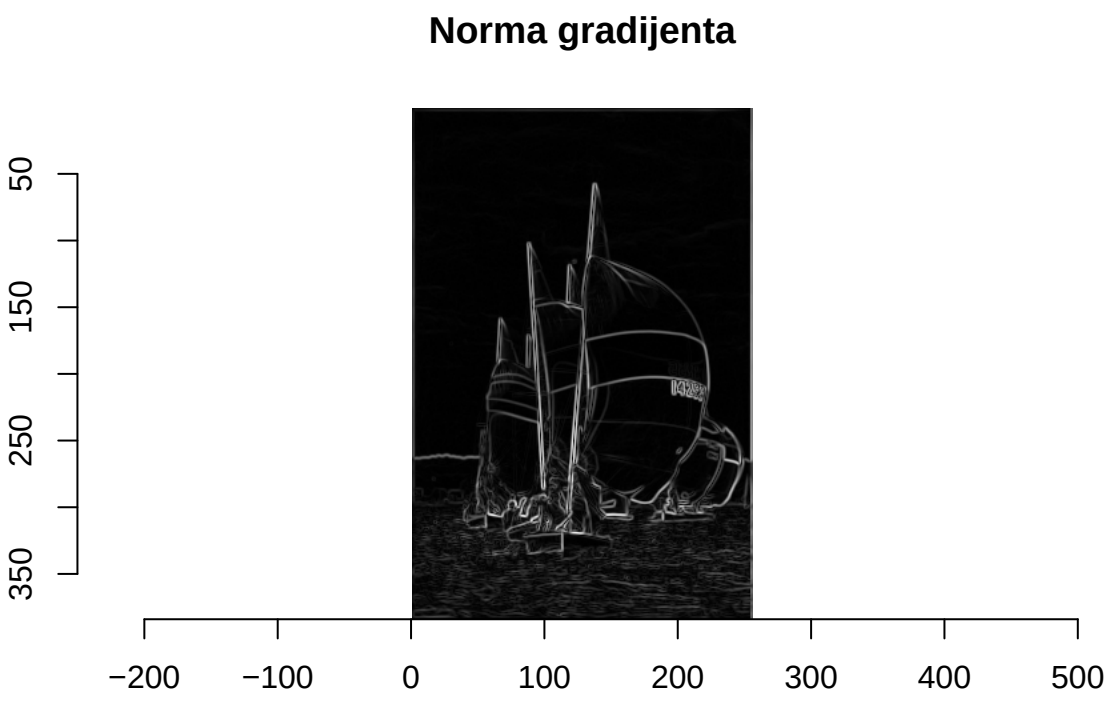


Уколико са $I(x,y)$ означимо интензитет слике на позицији (x,y), на првој слици су заправо приказани $\frac{\partial}{\partial x}I(x,y)$, а на другој $\frac{\partial}{\partial y}I(x,y)$. Градијент у некој тачки показује нам колико се слика брзо мења у околини неке тачке. Стога ивице објеката на слици одговарају областима високог градијента, те је смислено тражити их тамо где је норма градијента

$$\sqrt{\left(\frac{\partial}{\partial x}I(x,y)\right)^2+\left(\frac{\partial}{\partial y}I(x,y)\right)^2}$$

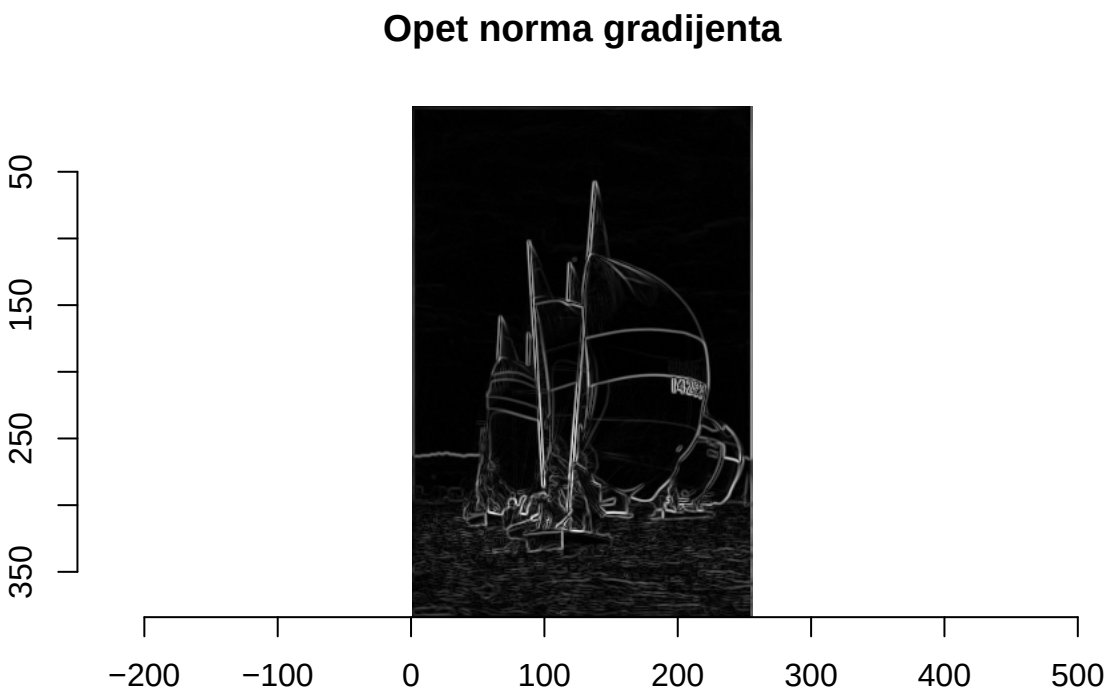
максимална. Сада ћемо исцртати норму градијента.

```
dx <- imgradient(boats.g,"x")
dy <- imgradient(boats.g,"y")
grad.mag <- sqrt(dx^2+dy^2)
plot(grad.mag,main="Norma gradijenta")
```



Видимо да је изузетно добро детектовао ивице. За ово постоји и згодна пречица.

```
imgradient(boats.g,"xy") %>% enorm %>% plot(main="Opet norma gradijenta")
```



Користили смо функцију `enorm`, која узима листу слика и рачуна повратну вредност по формули

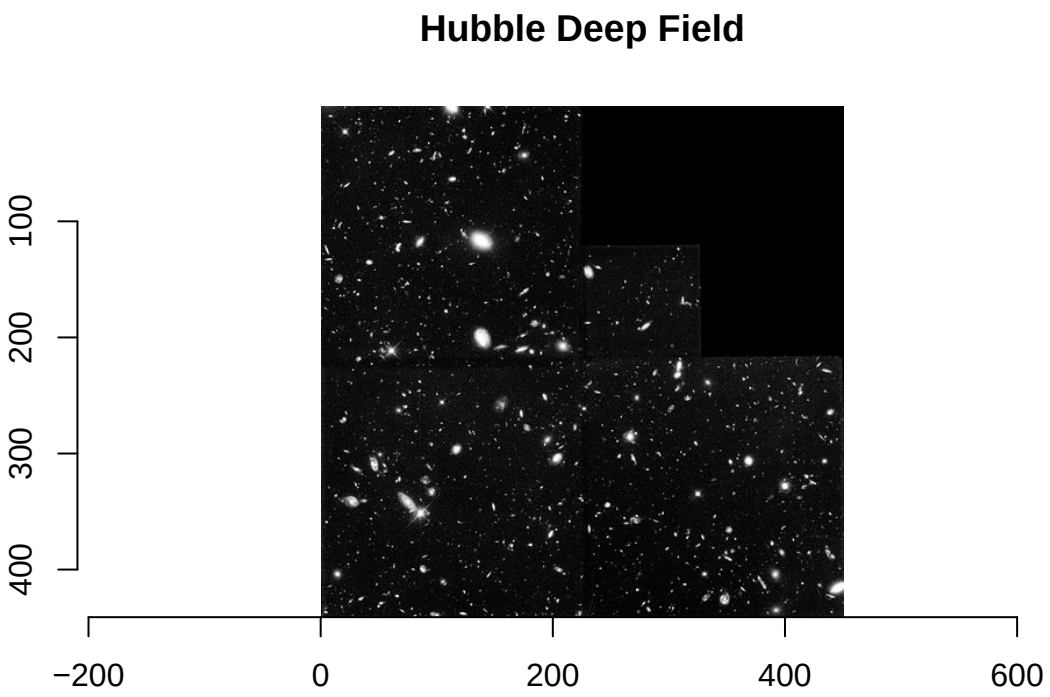
$$u_i = \sqrt{\sum_j x_{ij}^2},$$

где је x_{ij} вредност пиксела i на j-тој слици у листи. Ова функција јесте пример тзв. редукционих функција, то јест функција које комбинују вредности пиксела са више слика.

8.4 Пример: Детекција области (blob detection)

Коначни циљ овог одељка, до којег ћемо полако доћи, биће да детектујемо галаксије на наредној слици, која је усликана телескопом Хабл.

```
hub <- load.example("hubble") %>% grayscale
plot(hub,main="Hubble Deep Field")
```

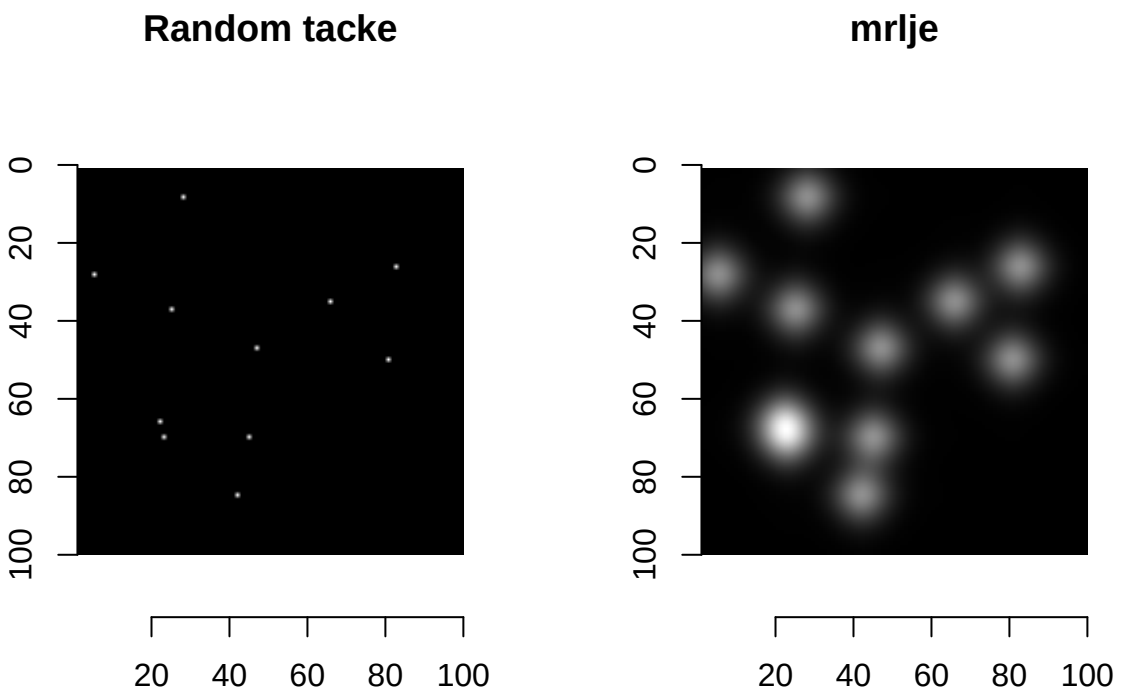


Почећемо прво са вештачким подацима. Ево примера како да се генерише слика на којој су на случајан начин постављене мрље (blob ћемо звати мрља у одсуству бољег термина).

```
layout(t(1:2))
set.seed(2)
points <- rbinom(100*100,1,.001) %>% as.cimg
```

```
## Warning in as.cimg.vector(obj, ...): Guessing input is a square 2D image
```

```
blobs <- isoblur(points,5) # zamucivanje
plot(points,main="Random tacke")
plot(blobs,main="mrlje")
```



Избацује упозорење јер није сигуран да ли сме да претпостави да су димензије 100×100 (можда су, рецимо, 10×1000). Ако хоћемо да будемо експлицитни, можемо да наведемо димензије:

```
rbinom(100*100,1,.001) %>% as.cimg(x=100,y=100)
```

```
## Image. Width: 100 pix Height: 100 pix Depth: 1 Colour channels: 1
```

Претпоставимо да је наш задатак да нађемо центре ових мрља. Постоји више начина да се то уради, а један од њих је посматрањем Хесијана слике у свакој тачки. Мрље су, у смислу осветљености, локални максимуми слике, а локалне максимуме карактерише негативно дефинитан Хесијан. Матрица која је негативно дефинитна има негативну детерминанту, а њу рачунамо стандардно:

$$\det(H) = I_{xx}I_{yy} - I_{xy}^2.$$

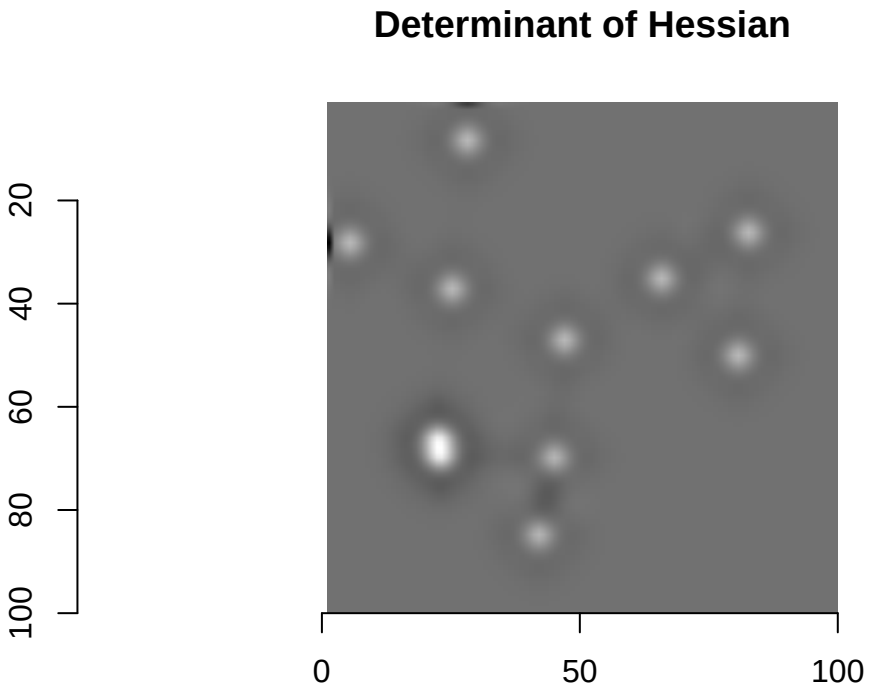
Може се показати да је довољно тражити максимум ове детерминанте, а он се налази нумерички. Кога занима више, нек погледа [2] (на часу ћу испричати мало детаљније, није од велике важности). Ми за то имамо уграђену функцију.

```
imhessian(blobs)
```

```
## Image list of size 3
```

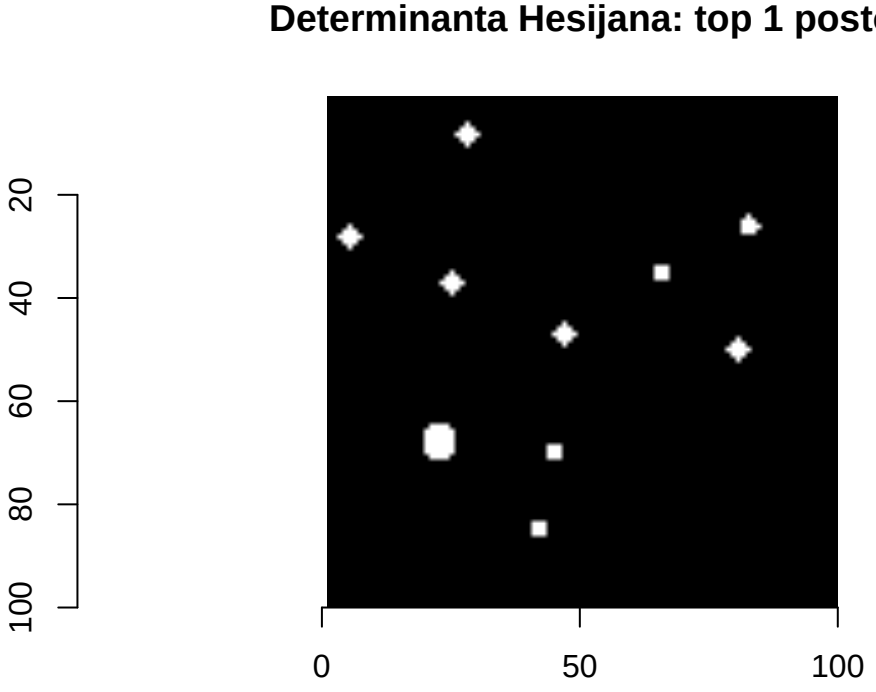
Имамо три слике у листи, свака за одговарајући извод. Сада ћемо да исцртамо детерминанту.

```
Hdet <- with(imhessian(blobs),(xx*yy - xy^2))
plot(Hdet,main="Determinant of Hessian")
```



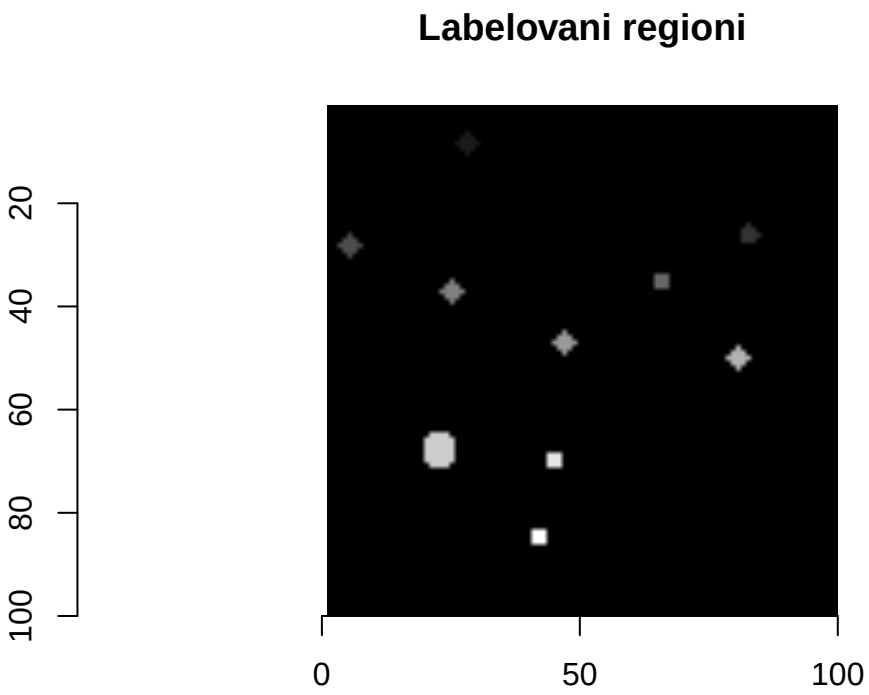
Да бисмо издвојили пикселе са највећом вредношћу, поставићемо неки трешхолд. То се ради овако:

```
threshold(Hdet,"99%") %>% plot(main="Determinanta Hesijana: top 1 posto")
```



Сада имамо дискретне регионе, и ако успемо да издвојимо њихове центре успећемо да нађемо и центре полазних мрља. Сада ћемо да употребимо функцију `label` која ће сваки бели регон да попуни пикселима одређене вредности, а позадину ће да остави на нули. Алгоритам којим попуњава нећемо радити јер је прекомпликован, рад се може наћи у `?label`.

```
lab <- threshold(Hdet,"99%") %>% label
plot(lab,main="Labelovani regioni")
```



Поента је да је свака мрља попуњена различитом нијансом, па могу да се разликују. То можемо видети ако кастујемо у dataframe.

```
df <- as.data.frame(lab) %>% subset(value>0)
unique(df$value) #10 regiona
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
head(df, 3)
```

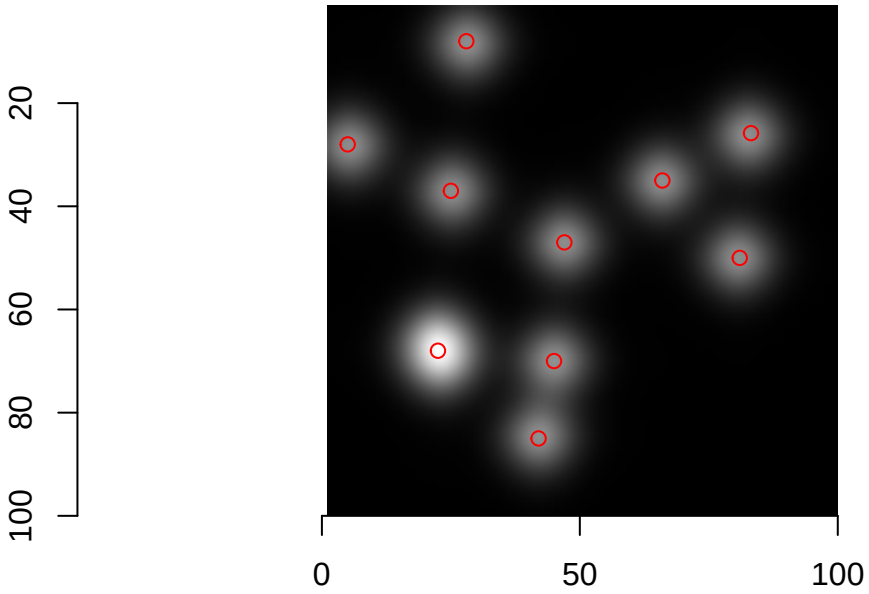
```
##      x y value
## 528 28 6     1
## 627 27 7     1
## 628 28 7     1
```

Видимо да су нам дате координате тачака и категорија у којој су. Сада делимо ручно на катеогрије и рачунамо средину за сваку.

```
# ovo je kraci nacin, moglo je bas rucno
centers <- dplyr::group_by(df,value) %>% dplyr::summarise(mx=mean(x),my=mean(y))
```

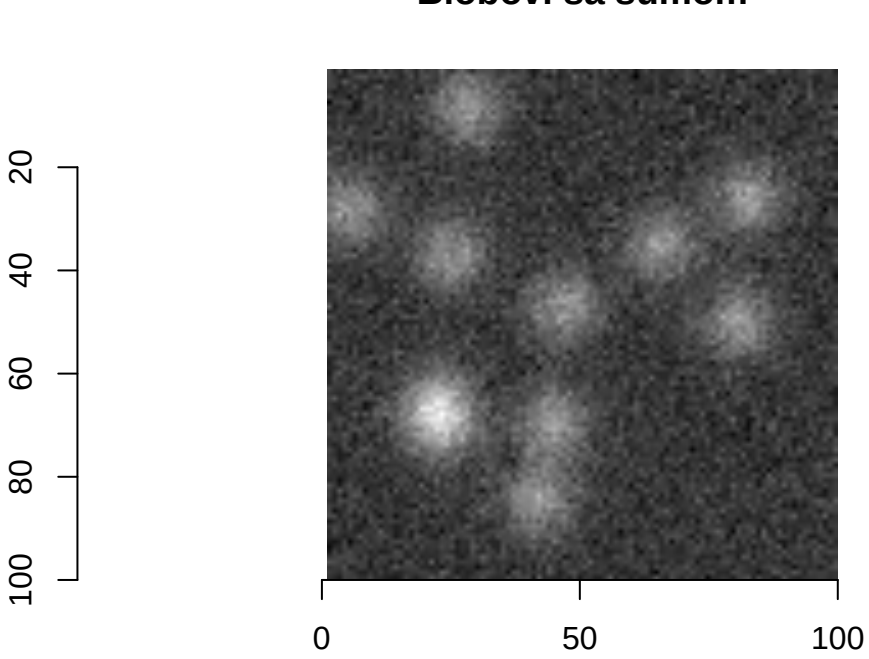
Сад цртамо блобове и центре заједно.

```
plot(blobs)
with(centers,points(mx,my,col="red"))
```



Видимо да их је пронашло веома добро. Да закомпликујемо, додаћемо шум.

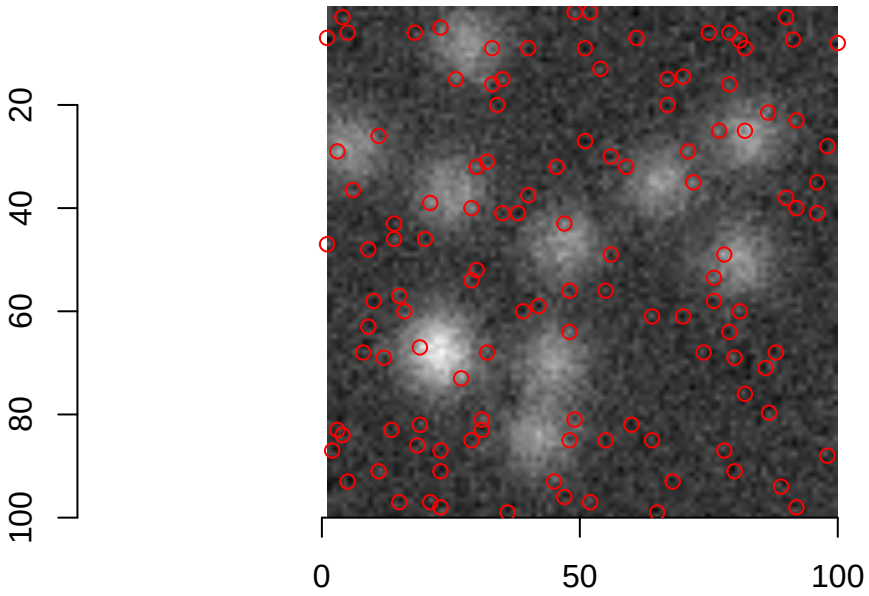
```
nblobs <- blobs+.001*imnoise(dim=dim(blobs))
plot(nblobs,main="Blobovi sa sumom")
```



Ако поново покушамо исто, театрално пропадамо.

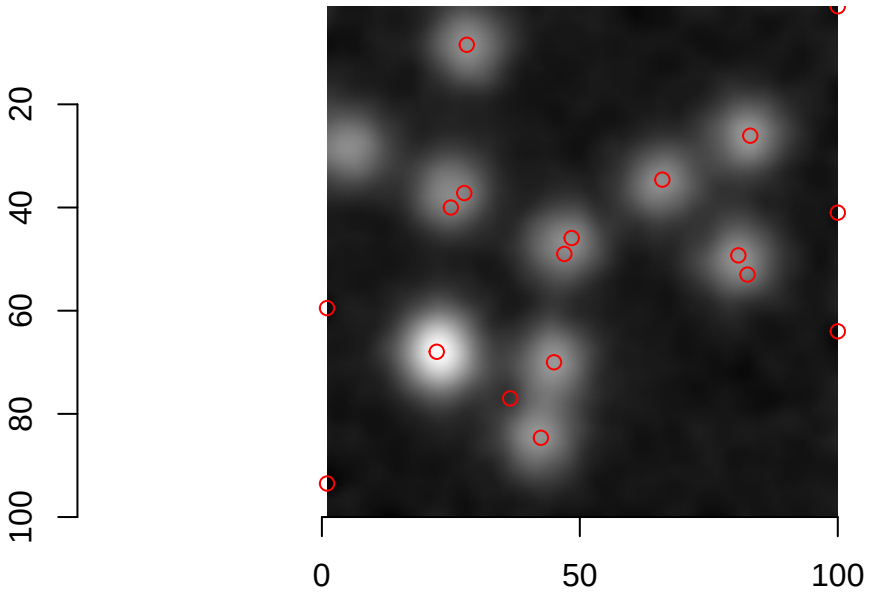
```
get.centers <- function(im,thr="99%")
{
    dt <- imhessian(im) %$% { xx*yy - xy^2 } %>% threshold(thr) %>% label
    as.data.frame(dt) %>% subset(value>0) %>% dplyr::group_by(value) %>% dplyr::summarise(mx=mean(x),my
}
```

```
plot(nblobs)
get.centers(nblobs,"99%") %$% points(mx,my,col="red")
```



Дакле, фали нам уметнути корак у коме уклањамо шум. Класично замућивање овде ће да одради посао.

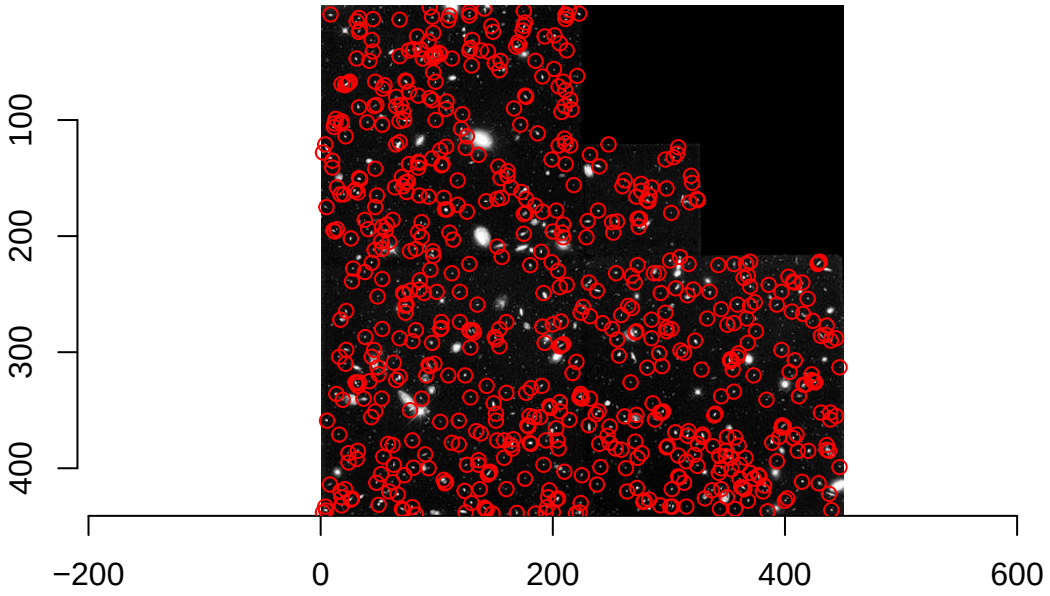
```
nblobs.denoised <- isoblur(nblobs,2)
plot(nblobs.denoised)
get.centers(nblobs.denoised,"99%") %$% points(mx,my,col="red")
```



Мало омашује али океј је. Сада смо спремни да пређемо на слику са почетка одељка.

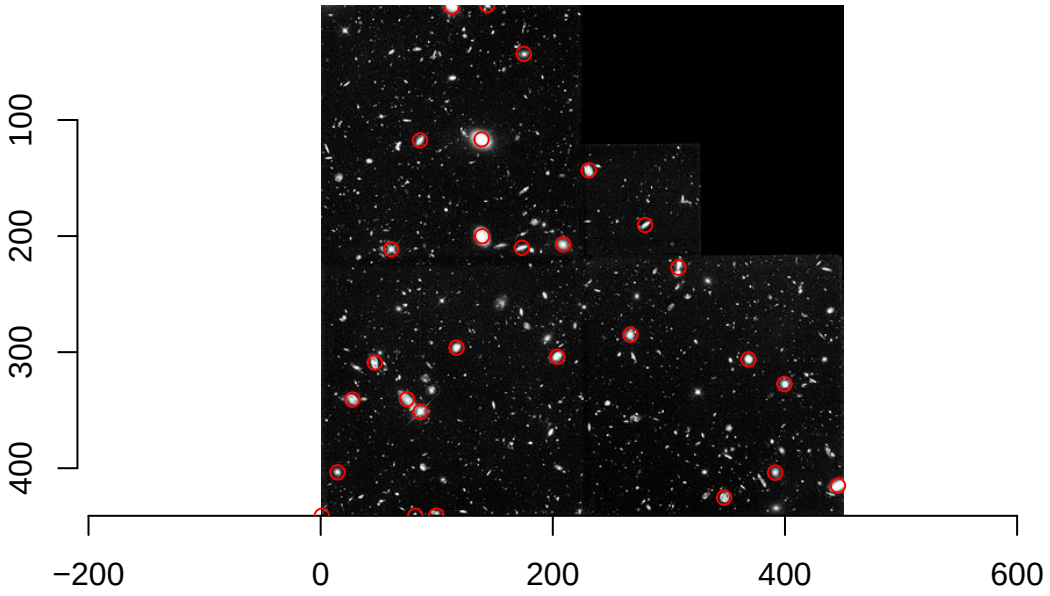
Први наивни покушај јесте да поновимо оно што смо раније радили.

```
plot(hub)
get.centers(hub,"99.8%") %$% points(mx,my,col="red")
```



Детектор нам купи и мале објекте који нису галаксије. Пробајмо класично замућивање.

```
plot(hub)
isoblur(hub,5) %>% get.centers("99.8%") %$% points(mx,my,col="red")
```



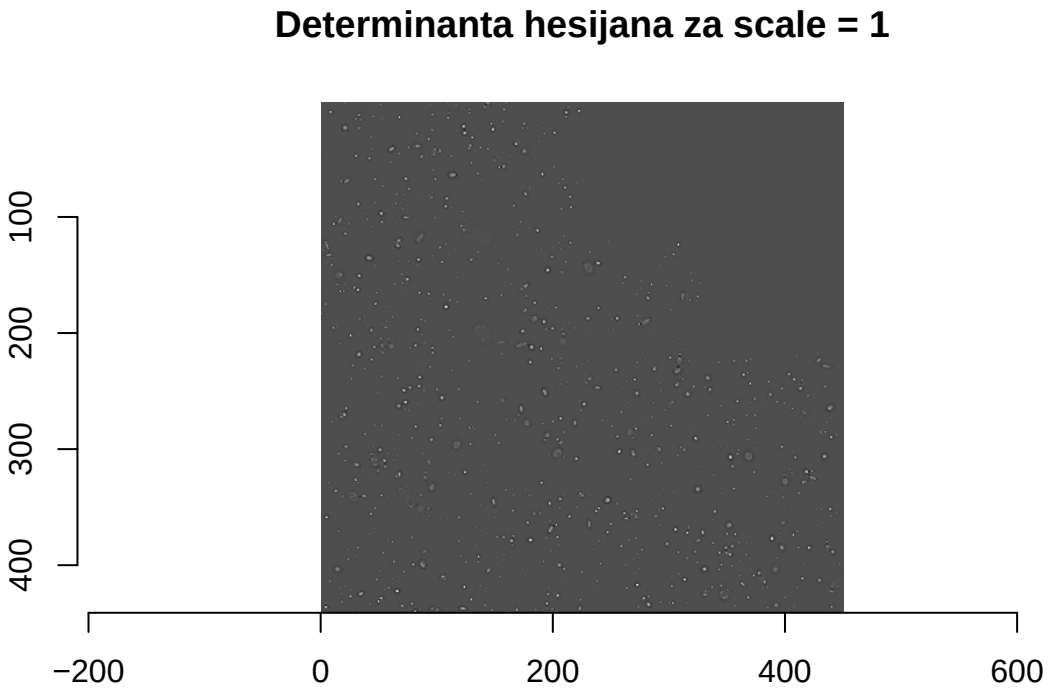
Сад пропушта неке галаксије. Дакле, треба нам да некако детектујемо објекте различитих редова величина димензија. То ћемо наставити на следећим вежбама.

Овде су почеле девете вежбе, али по природи ствари спојио сам све са осмим, јер представља једну целину.

Сетимо се, имали смо проблем детекције галаксија на слици која је снимак телескопом Хабл. Проблем се појавио када је требало детектовати објекте различитих редова величина. Сада ћемо то да решимо.

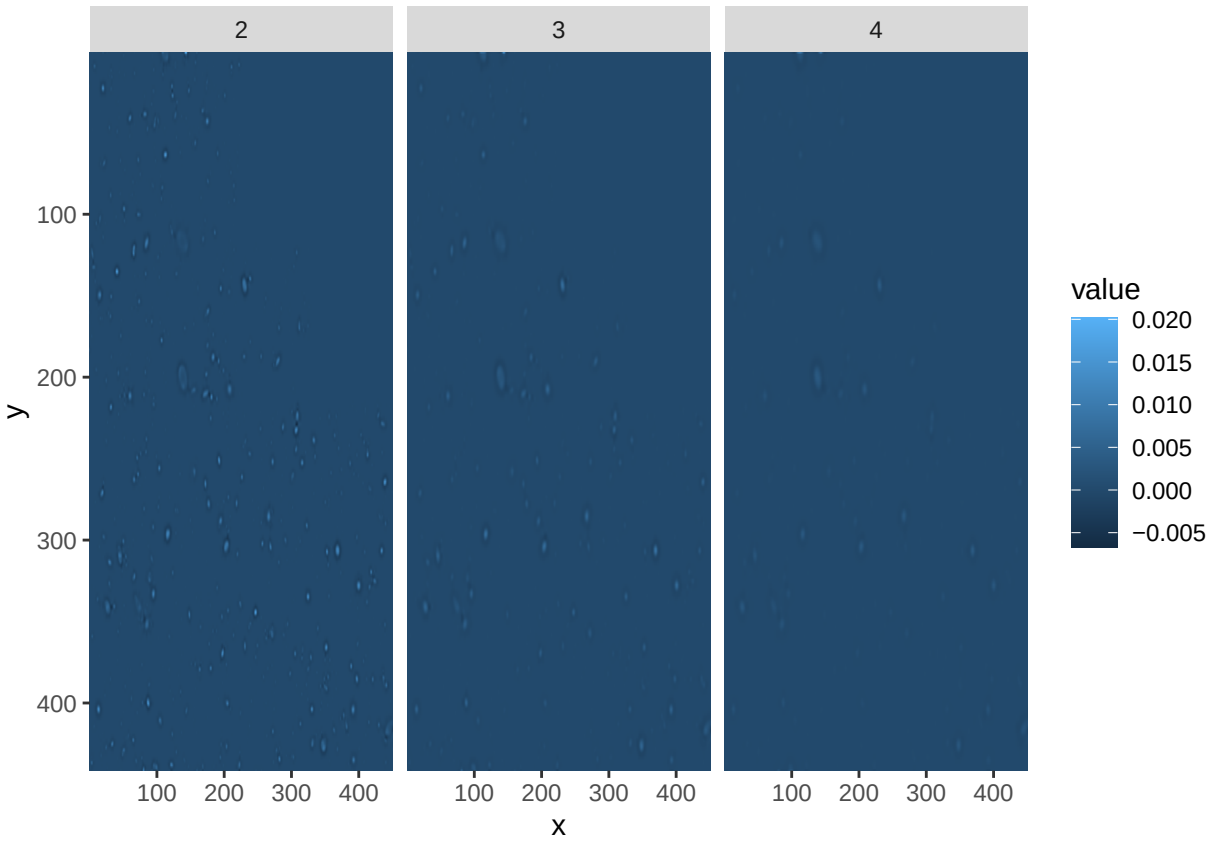
Штос је у томе што смо ми центре „мрља'' (блобова) тражили тамо где нам је највећа детерминанта хесијана. Сада ћемо детерминанту хесијана множити бројем, да јој повећамо магнитуду, и онда на таквој помноженој детерминанти тражити максимум. Синтаксу не треба памтити, не памтим је ни ја, већ сам је преузео са GitHub-а. Треба памтити поенте: овде је поента да се детерминанта хесијана множи неким бројем, да би се „боље видела''.

```
hessdet <- function(im,scale=1) isoblur(im,scale) %>% imhessian %$% { scale^2*(xx*yy - xy^2) }
plot(hessdet(hub,1),main="Determinanta hesijana za scale = 1")
```



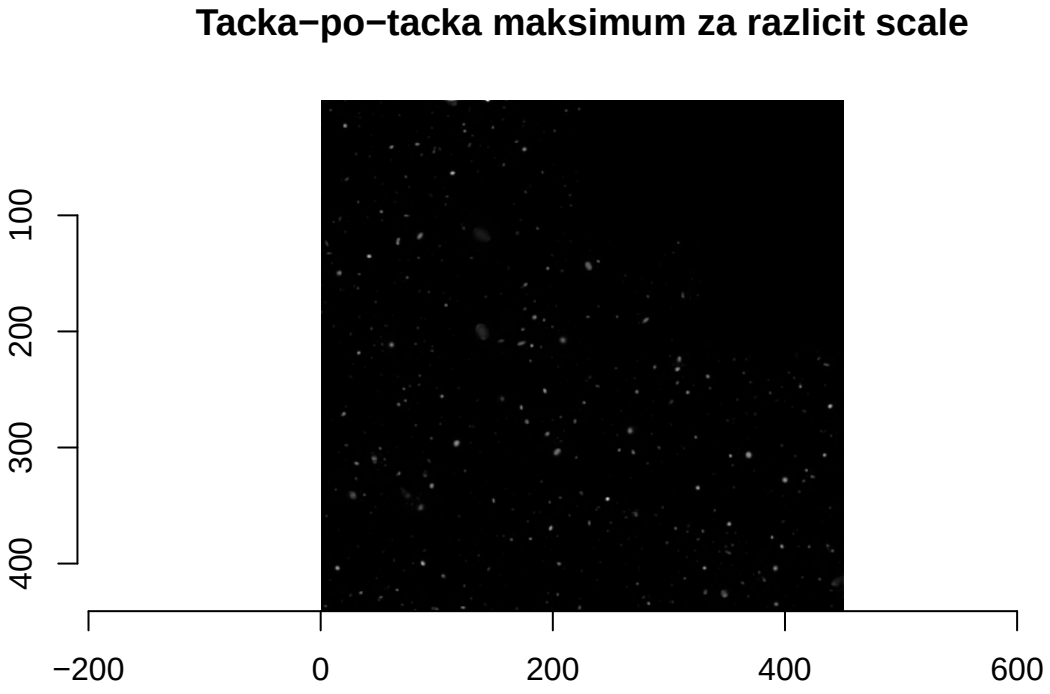
Можемо се поиграти па погледати како изгледа детерминанта хесијана за различите вредности параметра `scale`.

```
dat <- ldply(c(2,3,4),function(scale) hessdet(hub,scale) %>% as.data.frame %>% mutate(scale=scale))
p <- ggplot(dat,aes(x,y))+geom_raster(aes(fill=value))+facet_wrap(~ scale)
p+scale_x_continuous(expand=c(0,0))+scale_y_continuous(expand=c(0,0),trans=scales::reverse_trans())
```



Сада ћемо тражити максимуме за различите вредности параметра `scale`.

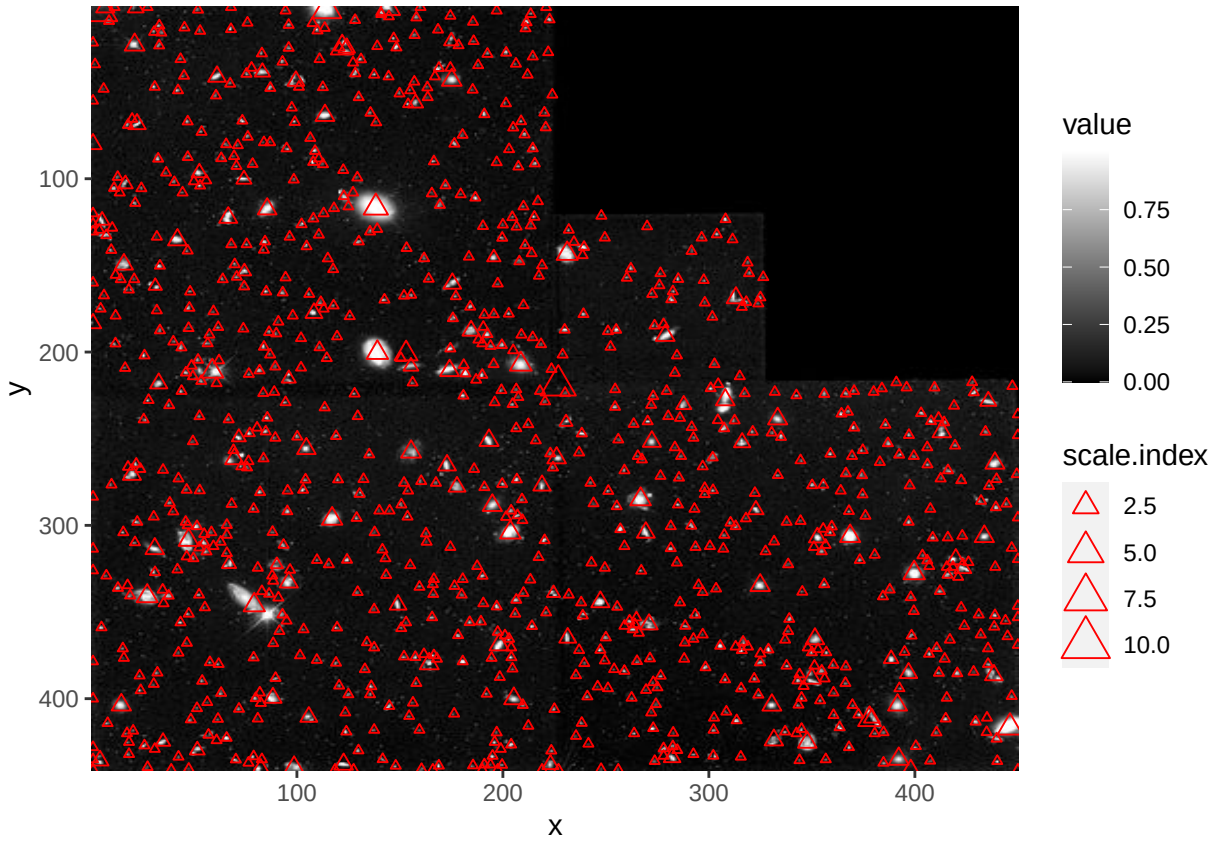
```
scales <- seq(2,20,l=10)

d.max <- llply(scales,function(scale) hessdet(hub,scale)) %>% parmax
plot(d.max,main="Tacka-po-tacka maksimum za razlicit scale")
```



У горњем коду једина функција која се истиче јесте функција `parmax`, која је још један пример редукционе функције. Она узима максимуме за различите вредности `scale`, и бира највећи.

Након поновног лабеловања, као и малопре, добија се слика са центрима. И ова синтакса је гуглабилна и не треба је памтити.

```
# Gledamo koja vrednost "scale" je uzeta za najvecu
i.max <- llply(scales,function(scale) hessdet(hub,scale)) %>% which.parmax
#Uzimamo data.frame labelovanih regiona
labs <- d.max %>% threshold("96%") %>% label %>% as.data.frame
#Dodajemo indekse za scale
labs <- mutate(labs,index=as.data.frame(i.max)$value)
regs <- dplyr::group_by(labs,value) %>% dplyr::summarise(mx=mean(x),my=mean(y),scale.index=mean(index))
p <- ggplot(as.data.frame(hub),aes(x,y))+geom_raster(aes(fill=value))+geom_point(data=regs,aes(mx,my,si
p+scale_fill_gradient(low="black",high="white")+scale_x_continuous(expand=c(0,0))+scale_y_continuous(ex
```



Сада је добро.

Вежбе 9

Паралелно израчунавање

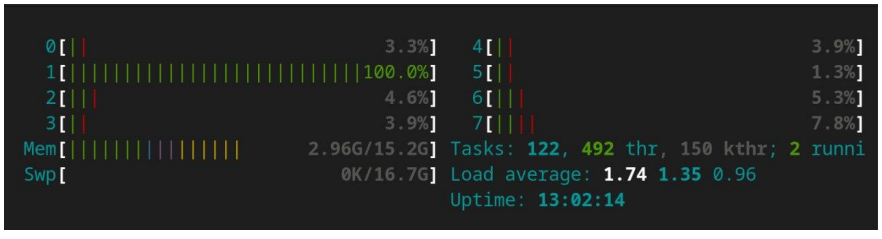
Напомена. Делимично преузето из [1].

Већина модерних рачунара располаже са више од једног процесорског језгра, што значи да сам процесор има више јединица у којима може извршавати основне процесорске инструкције. Рецимо, уколико имамо пуштен YouTube видео, он ће користити једно процесорско језгро. Уколико поред тога имамо отворен процес у терминалу, рецимо ажурирање система, он ће користити још једно процесорско језгро, и та два процеса ће се извршавати паралелно.

Већина модерних процесора омогућава и такозвани *hyperthreading* (вишенитност, ваљда; thread = нит), што значи да се једно процесорско језгро процесима на рачунару представља као два, тзв. логичка језгра. То с једне стране доноси бенефите да уколико је неки програм оптимизован да ради на 4 процесорска језгра, он ће моћи адекватно да ради и на рачунару који има само 2 физичка, али 4 логичка језгра. Наравно радиће спорије, али неће постојати озбиљнији проблеми, прекиди, багови.

Суштина је та да што више физичких језгара рачунар има, то ће више процеса моћи паралелно да извршава.

Програмски језик `R` направљен је тако да у сваком тренутку времена користи само једно ЛОГИЧКО језгро. Рецимо, рачунар на којем је ово писано има 4 физичка и 8 логичких језгара. У тренутку компилације овог `RMarkdown` документа, искоришћеност језгара је следећа:



Дакле, програмски језик `R` пати од проблема недостатка паралелног израчунавања. Од идентичних проблема пати и `Python`. Срећом, постоји начин да се ово реши, иако није претерано елегантан.

Наравно, погрешно би било рећи да постоје два начина извршавања процеса: серијски и паралелно. Постоје, наиме, и они између: када се процеси извршавају делимично серијски, а делимично паралелно. Ми ћемо се овде бавити „што паралелнијим'' начином извршавања.

9.1 Пакет`parallel`

Постоји неколико пакета у `R`-у који адаптирају паралелно израчунавање. Први који су се појавили и били јако квалитетни јесу `multicore` и `snow`. Међутим, они су већ одавно спојени у један пакет који долази у базној `R` инсталацији, а који се зове `parallel`.

```
library(parallel)
```

На следећи начин можемо проверити број (логичких) језгара које `R` види:

```
detectCores()
```

```
## [1] 8
```

Као што смо напоменули, рачунар на којем су ови материјали писани има 4 физичка и 8 логичких језгара.

9.2 Начини паралелизације

Постоје два основна начина паралелизације: `socket` и `forking`:

- Код `socket` приступа покреће се нова верзија `R`-а на сваком логичком језгру. Може да избаци упозорење и да пита да ли да дозволи повезивање тих инстанци; треба одговорити потврдно.
- Код `forking` приступа постојећа инстанца `R`-а копира се на ново језгро.

Постоје предности и мане и за један и за други.

`SOCKET` - предности:

- Ради на сваком оперативном систему, па и на Windows-у.[1]
- Сви процеси су независни један од другог, нема укрштања па не може доћи до проблема при спајању (cross-contamination).

`SOCKET` - мане:

- Спорији је.
- Ствари попут учитавања пакета морају да се одраде за сваки процес посебно. Слично је са променљивим.
- Много је компликованије за имплементирати.

`FORKING` - предности:

- Бржи.
- Исти workspace постоји у свим верзијама `R`-а.
- Тривијалан за имплементирати.

`FORKING` - мане:

- Ради само на POSIX (shell) компатибилним системима (MAC[2], GNU/Linux, BSD).
- Како су процеси дупликовани, проблем настаје у генерисању случајних бројева, има понављања.

Биће обрађен `forking` као бољи приступ. Ко је на `Windows`-у, може погледати `socket` у [1].

[1] Ако сте догурали до четврте године Математичког факултета В смера, а и даље користите Windows као главни оперативни систем, образовни систем је негде озбиљно заказао у вашем случају.

[2] Ако сте догурали до четврте године Математичког факултета В смера, а и даље користите MAC OS / OS X као главни оперативни систем, образовни систем је негде озбиљно заказао у вашем случају.

Напомена. Људи који користе GNU/Linux, а у коју групу спадам и сам, деле се на 2 групе: Они који мрзе Windows, али БАШ мрзе MAC и они који мрзе MAC, али БАШ мрзе Windows.[3] Ја (Данијел Гојков Алексић) дефинитивно спадам у другу групу.

9.3 `forking` коришћењем `mcapply`

Просто, у R-у постоји позната функција `lapply` која примењује функцију на листу велики број пута. Уместо ње, постоји функција `mclapply`, скраћено од *multi-core apply*, која процес „форкује'' (развиљушкава?) на више језгара. Хајде да упоредимо брзине.

```
# Pravimo neku bzvz funkciju
library(MASS)
f <- function(i) { lm(medv ~., data = Boston)  }
set.seed(300)
system.time(modeli <- lapply(1:1000, f))
```

```
##    user  system elapsed
##   2.113   0.000   2.118
```

```
# Sada isto sa mcapply
set.seed(300)
system.time(modeli <- mclapply(1:1000, f))
```

```
##    user  system elapsed
##   0.760   0.246   3.194
```

Циљ је да збир user + system буде мањи (гуглати).

[3] Luke Smith, са PeerTube/Odysee/YouTube канала Luke Smith. Погледати lukesmith.xyz.

Библиографија

[1] https://dept.stat.lsa.umich.edu/~jerrick/courses/stat701/notes/parallel.html

[2] https://en.wikipedia.org/wiki/Blob_detection#The_determinant_of_the_Hessian

[3] https://www.datanovia.com/en/lessons/dbscan-density-based-clustering-essentials/#algorithm

[4] https://stat.ethz.ch/pipermail/r-help/2005-December/083823.html

[5] https://youtu.be/MnRskV3NY1k

[6] https://dahtah.github.io/imager/gettingstarted.html

[7] Donald B. Rubin (1986). *Multiple Imputation for Nonresponse in Surveys*. JOHN WILEY & SONS, New York.

[8] Donald B. Rubin (1987a). *Multiple Imputation for Nonresponse in Surveys*. JOHN WILEY & SONS, New York (Друго издање).

[9] Roderick J. A. Little, Donald B. Rubin (1987). *Statistical Analysis with Missing Data*. John Wiley & Sons, New York (Прво издање).

[10] Roderick J. A. Little (1988). *Missing-Data Adjustment in Large Surveys*. Journal of Business & Economic Statistics , Jul., 1988, Vol. 6, No. 3 (Jul., 1988), pp. 287-296. Taylor & Francis, Ltd. on behalf of American Statistical Association.

[11] Donald B. Rubin (1996). *Multiple imputation after 18+ years.* Journal of the American Statistical Association, 91(434):473–489.

[12] John B. Barnard, Donald B. Rubin (1999). *Small-sample degrees of freedom with multiple imputation.* Biometrika, 86(4):948-955.

[13] Jaap P. L. Brand. *Development, Implementation and Evaluation of Multiple Imputation Strategies for the Statistical Analysis of Incomplete Data Sets*. PhD thesis, Erasmus University, Rotterdam, 1999.

[14] Roderick J. A. Little, Donald B. Rubin (2002). *Statistical Analysis with Missing Data*. John Wiley & Sons, New York (Друго издање).

[15] G. Hinton, S.T. Roweis (2002). *Stochastic Neighbor Embedding*. NIPS.

[16] L. Van der Maaten, G. Hinton (2008). *Visualising data using t-SNE*. Journal of machine learning research.

[17] Stanislav Paluch (2008). *Imformation Theory*. Europsky socialny fond, Žilina.

[18] Мато Пижурица, Митар Пешикан, Јован Јерковић (2010). *Правопис српскога језика*. Матица српска, Нови Сад.

[19] Craig K. Enders (2010). *Applied Missing Data Analysis*. Guilford Press, New York.

[20] Rebecca R. Andridge, Roderick J. A. Little (2010). *A review of hot deck imputation for survey non-response.* International Statistical Review, 78(1):40–64.

[21] Stef van Buuren (2018). *Flexible Imputatation of Missing Data.* Second edition, Chapman & Hall/CRC.

[22] Alfonso Delgado-Bonal, Alexander Marshak (2019). *Approximate Entropy and Sample Entropy: A Comprehensive Tutorial.* Entropy, 2019. DOI: https://doi.org/10.3390/e21060541

[23] Младен Николић, Анђелка Зечевић (2019). *Машинско учење (скрипта).* Математички факултет, Београд.

[24] Милан Јовановић (2020). *Предавања из Теорије узорака.* Математички факултет, Београд.

[25] Бојана Милошевић (2020). *Линеарни статистички модели* (скрипта). Математички факултет, Београд.

[26] Данијел Алексић (2021). *Импутација недостајућих података у једнодимензионим временским серијама* (семинарски рад). Математички факултет, Београд.
Рад захтевати од аутора, на danijel.aleksic98@gmail.com, а тренутно (јул 2021) је доступан на линку https://drive.google.com/file/d/1FXpR2yHPzrka1yeDLOHq8-QNErY0WQK1/view?usp=sharing.

[27] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2021). *An Introduction to Statistical Learning.* Free online edition, August 4.