TDDE01. Lab2. Group B24 report.

# **Statement of contribution**

Firstly, general analysis was performed teamwise. Approaches and strategies of solving the tasks were elaborated teamwise as well.

Student Elham was responsible for code writing and problem solving for the Assignment 3. Student Anton was responsible for code writing and problem solving for the Assignment 2. Student Elena was responsible for code writing and problem solving for the Assignment 1.

After completion of coding stage group analyzed the results together and peer reviewed the results of each other’s work. Finally, each student corrected their solutions according to the received reviews from groupmates.

# **Assignment 1.**

# **Assignment 2.**

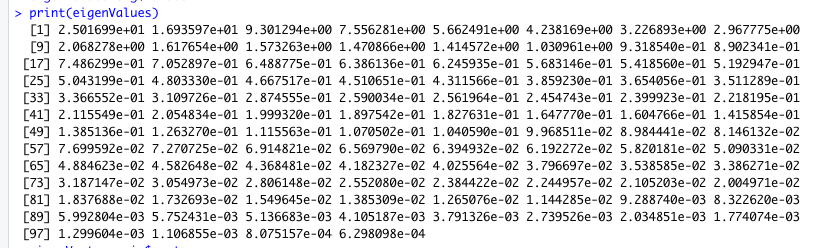
**Assignment 3. Principal components and implicit regularization**

## **Assignment3: Task1**

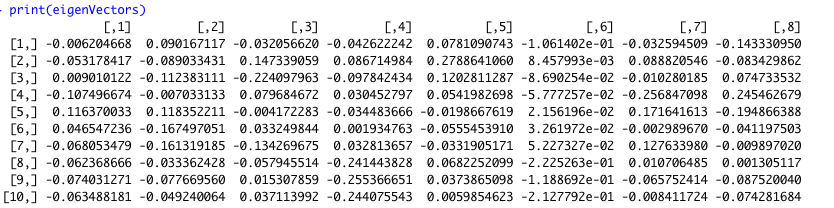
At first, we scaled all data without column number 101(ViolentCrimesPerPop) and then we add the column (101) to our data.

The next step was implementing PCA so we compute covariance matrix for our data without last column, then we used eigen function to have eigen values and eigen vectors.

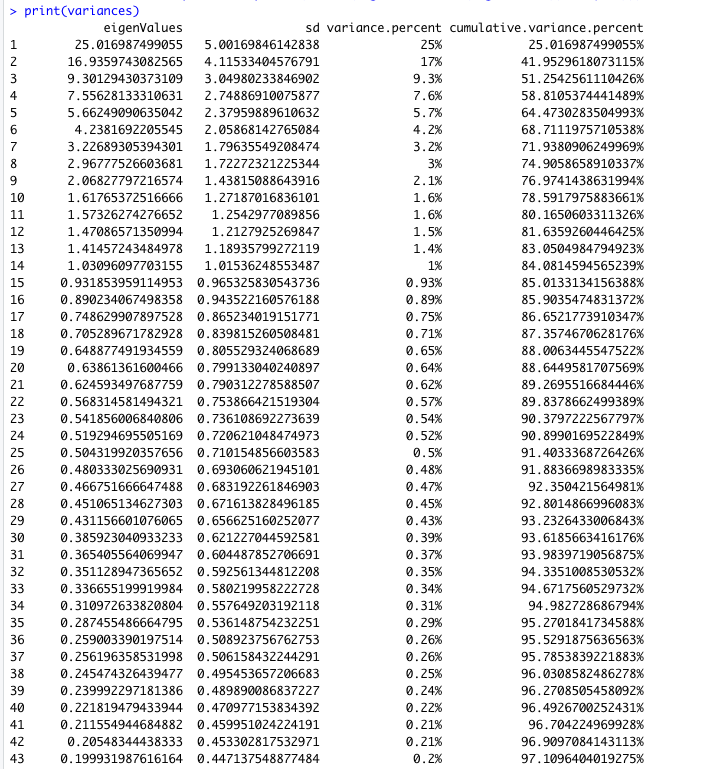
EigenValues:



EigenVectors:



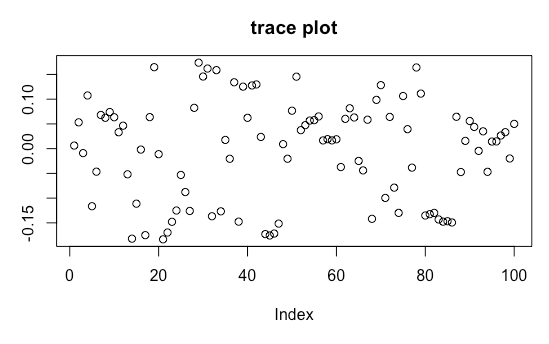
Based on our computing for obtaining 95% of variance in the data we need 35 components. To get this result we calculate proportion of variation then made a data frame and calculate cumulative variance percent and then count those that had the variance more than 95%. And the proportion of variation by two fist PC is 25.017 + 16.936 sums up to 41.95



## **Assignment3: Task2**

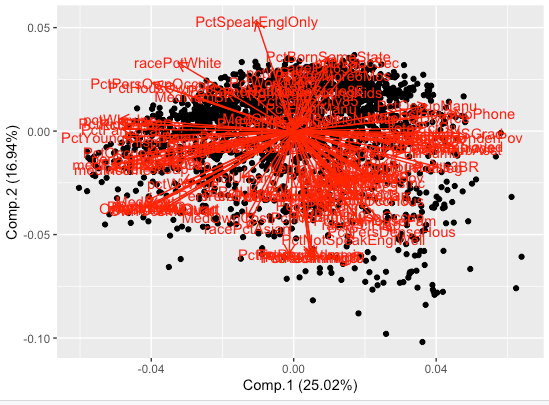
In this task we used princomp() function to have PCAs. We draw 3 different plots. The first one did not contain readable information

plot(xy$loadings[,1],main="Traceplot")



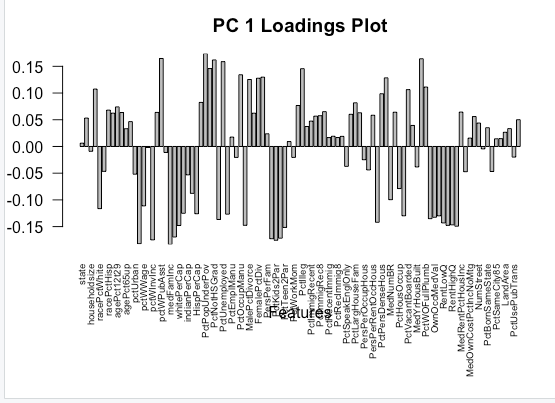
We used another plot it was a mess and not readable

autoplot(xy,data,loadings = TRUE,loadings.label = TRUE)



And a bar plot

barplot(xy$loadings[,1],main="PC1LoadingsPlot",las=2,xlab="Features",ylab="loadings",xpd=FALSE,cex.names=0.6)



and judging from a bar plot there might be around 10 important variables for PC1 that have notable contribution to PC1.

In next step we stored loadings for PC1 in a vector sort it and get 5 biggest values, these are most important variables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PctPopUnderPov | pctWInvInc | PctKids2Par | medIncome | medFamInc |
| 0.1737978 | -0.1748683 | -0.1755423 | -0.1819830 | -0.1833080 |

1. medFamInc: median family income (differs from household income for non-family households)

2. medIncome: median household income

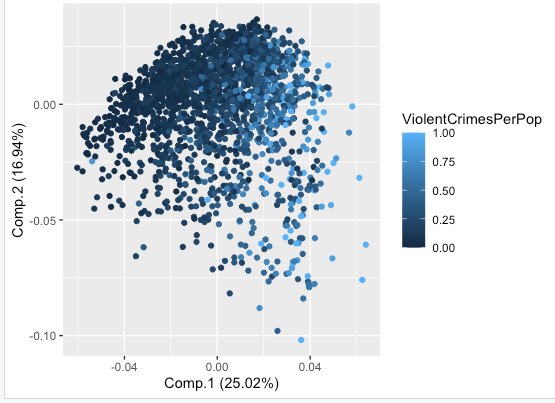
3. PctKids2Par: percentage of kids in family housing with two parents

4. pctWInvInc: percentage of households with investment / rent income in 1989

5. PctPopUnderPov: percentage of people under the poverty level

They all refer to wealth, well-being, profitability. First 4 contribute to crime level in a negative way, the last - in a positive way

Last step for this question was plotting PC scores (PC1, PC2), while the color of the points is shown by ViolentCrimesPerPop.



As you can see PC1 is related to crime increase and positive PC2 is associated with lower crime levels.

## **Assignment3: Task3**

We scaled, split (50/50) the data set and modeled it in a linear regression model while ViolentCrimesPerPop is the target then we estimated model by Mean Squared error for both test and train data:

MSEtest = 1.282006 MSEtrain = 0.2871696

We believed that the model is overfitted because test error is 6 times higher than train error, but it's hard to judge because MSE is not enough to interpret the model quality.

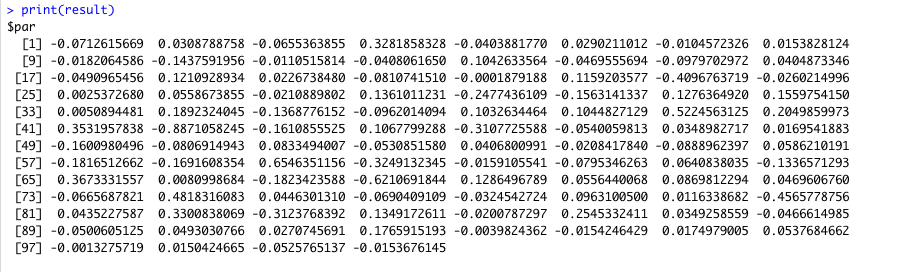
## **Assignment3: Task4**

We implemented a cost function and use the below formulas from lecture 1d, slide 7 to include theta. We also compute mean square error for both train and test inside the function.

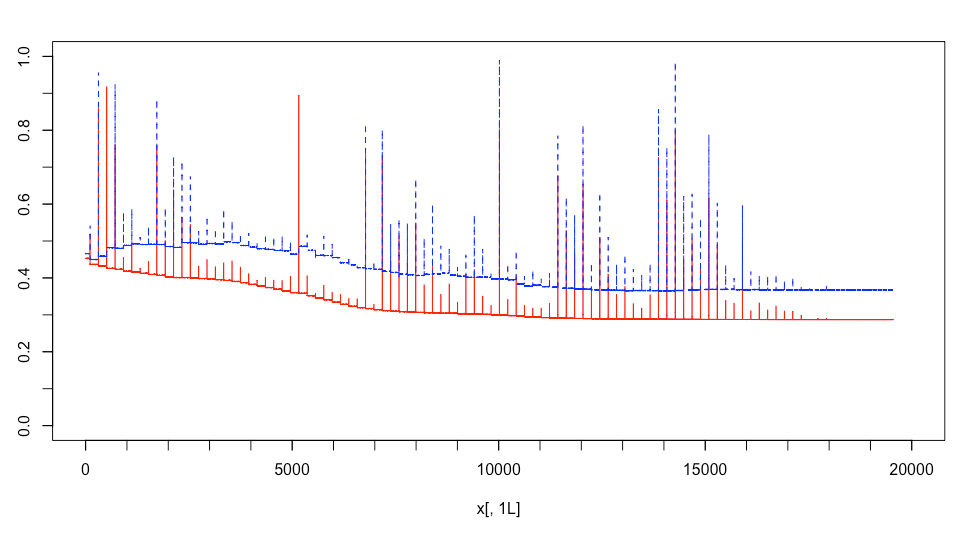
**Objective:**

**Usual choice**: Squared loss

For computing training and test errors for every iteration number we added global variables that were called inside the function and then used the BFGS methos in optim function.



We got huge numbers in MSEs. The outliers were 5% of data and they have changed the plot significantly so we removed them from vectors of MSEs, then plot them.



Blue test data Red train data

We interpret an early stopping criterion as a stop at an arbitrary iteration that is good enough in terms of size of error. In this case we could pick for example 2300 because after that point there seems to be relatively no difference in error size

No significant improvement of the error for 10000 iterations.

Train error from optimal model by early stopping criteria = 0.3343754

Test error from optimal model by early stopping criteria= 0.4550813

conclusion based on comparison, train MSE was higher in the optimal model, but the error on the test data was less than half of the error for the default linear regression model. The optimal model overfits less, much less than the default model.

**Assignment3: Codes**

**Assignment3-Task1:**

communities=read.csv("communities.csv")

#scaling

library(caret)

scaler=preProcess(communities[1:100])

communitiesScaled=predict(scaler,communities[1:100])

communitiesScaled$ViolentCrimesPerPop=communities$ViolentCrimesPerPop

data=communitiesScaled

data$ViolentCrimesPerPop=c()

#Covariance matrix and eigenvalues/vectors

covarianceMatrix <- cov(data)

eig=eigen(covarianceMatrix)

eigenValues=eig$values

print(eigenValues)

eigenVectors=eig$vectors

print(eigenVectors)

#calculate proportion of variation

sprintf("%2.3f",eigenValues/sum(eigenValues)\*100)

#create handy dataframe to analyze variance and answer questions

variances=data.frame(cbind( eigenValues,sd = sqrt(eigenValues),

variance.percent = paste0(signif((eigenValues/sum(eigenValues)),2)\*100,"%"),

cumulative.variance.percent = paste0(cumsum(eigenValues/sum(eigenValues))\*100,"%")))

**Assignment3-Task2:**

#Traceplot for PC1

xy=princomp(data)

#this trace plot is not easy to interpret ( barplot is easier to interpret):

plot(xy$loadings[,1],main="traceplot")

# this one is also not easy to interpret

autoplot(xy,data,loadings = TRUE,loadings.label = TRUE)

# this is more nice but not a trace plot:

par(mar=c(8, 3, 3, 1))

barplot(xy$loadings[,1], main="PC 1 Loadings Plot", las=2,xlab="Features",ylab="loadings",xpd=FALSE,cex.names=0.6)

# Report which 5 features contribute mostly to the first principle component.

vector1 = xy$loadings[,1] #storing loadings for PC1 in a vector

vector2 = vector1[order(abs(vector1))] #sort this vector by abs

tail(vector2,5) # getting 5 biggest values, those are most important variables:

# a plot of the PC scores (PC1, PC2) in which the color of the points is given by ViolentCrimesPerPop.

autoplot(xy,communitiesScaled,colour = 'ViolentCrimesPerPop')

**Assignment3-Task3:**

#scaling

library(caret)

scaler1=preProcess(communities)

communitiesScaled1=predict(scaler1,communities)

#dividing the train and test

set.seed(12345)

n=nrow(communitiesScaled1)

id=sample(1:n, floor(n\*0.5))

train=communitiesScaled1[id,]

test=communitiesScaled1[-id,]

#model based on training data and prediction on test

linearModelCrimes=lm(ViolentCrimesPerPop~.-1, data=train)

predictionTestCrime=predict(linearModelCrimes,newdata=test,interval = "prediction")

MSEtest=mean((test$ViolentCrimesPerPop-predictionTestCrime)^2)

sum=summary(linearModelCrimes)

MSEtrain=mean(sum$residuals^2)

**Assignment3-Task4:**

#Implement a function that depends on parameter vector 𝜃 and represents the

#cost function for linear regression without intercept on the training data set.

k=0

ListMSETrainErrors=list()

ListMSETestErrors=list()

costfunction<-function(theta,datax,datatarget){

result=(t(as.matrix(datatarget)-as.matrix(datax)%\*%as.matrix(theta))%\*%(as.matrix(datatarget)-as.matrix(datax)%\*%as.matrix(theta)))

.GlobalEnv$k= .GlobalEnv$k+1

.GlobalEnv$ListMSETrainErrors[[k]]=mean((as.matrix(train[,-101])%\*%as.matrix(theta)-as.matrix(train[,101]))^2)

.GlobalEnv$ListMSETestErrors[[k]]=mean((as.matrix(test[,-101])%\*%as.matrix(theta)-as.matrix(test[,101]))^2)

return (result) }

#use BFGS method (optim() function without gradient specified) # compute training and test errors for every iteration number. #adding global variables that were called inside the function.

par1 <- rep(c(0),each=100)

result<- optim(par1,fn=costfunction,datax=input,datatarget=output,method="BFGS")

print(result)

#remove some huge numbers in MSEs appear from vectors of MSEs

newlist = as.vector(ListMSETrainErrors)

newlist=newlist[newlist <1]

newlist1 = as.vector(ListMSETestErrors)

newlist1=newlist1[newlist1 <1]

#The outliers are 5% of data and they change the plot significantly.

#newlist and newlist1 have different lengths so we fit their lengths to each other to be able to plot them

diff = length(newlist) - length (newlist1)

if (diff<0) {

newlist = newlist[-c(1:500)]

newlist1 = newlist1[-c(1:500)]

diff = -1\*diff

newlist1 = newlist1[-c(1:diff)]

} else

{newlist = newlist[-c(1:500)]

newlist = newlist[-c(1:diff)]

newlist1 = newlist1[-c(1:500)] }

#iterationNumber=length(newlist)

matplot(y=cbind(newlist,newlist1),type="l",col=c("red","blue"),xlim=c(0,20000), ylim=c(0,1))

minor.tick(nx = 5, tick.ratio = 1)

#No significant improvement of the error for 10000 iterations.

print("train from optimal model by early stopping criteria")

print(newlist[6000])

print("test error from optimal model by early stopping criteria")

print(newlist1[6000])