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. Explicit regularization.

Task 1: Raw data description: the "tecator.csv" contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat. For each meat sample the data consists of a 100-channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry.

Columns containing levels of moisture, protein and sample number were removed from the dataset because they are not involved into analysis.

The raw data was divided by 50/50 to train and test data.

Assuming that Fat can be modeled as a linear regression in which Channels are used as features, we have the following probabilistic model:

$$Y = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \cdots .. + \theta_p * x_p + \epsilon$$

$$Y|X \sim N(\theta^T, \sigma^2)$$

The linear regression was fitted to the training data and training and test errors were estimated:

Mean squared error (train data)	Mean squared error (test data)
0.005709117	982.2478

The model fits the training data well, but as seen from the value of test error, the model is overfitted and too complex. It memorized the train dataset but performed poorly on test dataset.

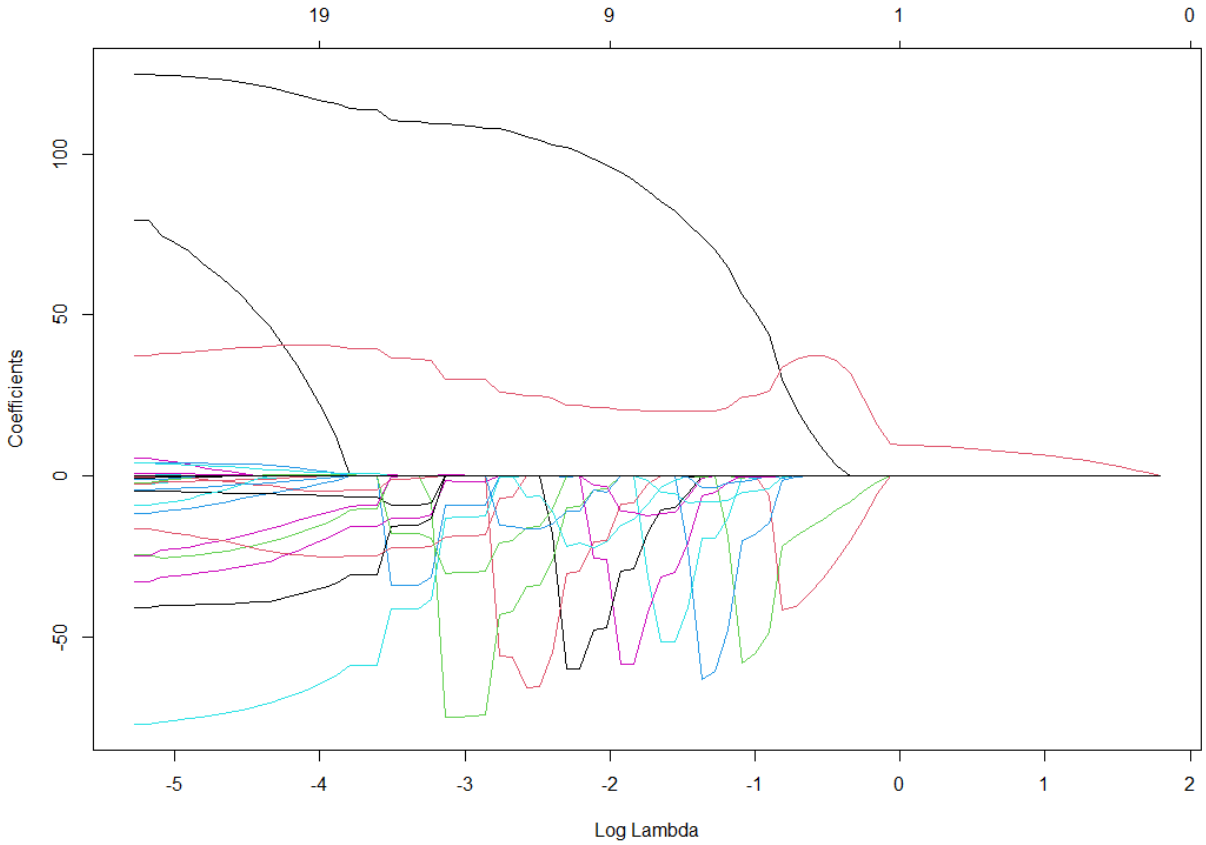
Task 2: Assuming that Fat can be modeled as a LASSO regression with Channels as features, we have the following cost function to optimize:

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \theta_0 - \theta_1 * x_{1j} - \cdots - \theta_p * x_{pj})^2 + \lambda \sum_{j=1}^{p} |\theta_i|$$

Where $\lambda > 0$ is penalty factor, y is target, x is features, θ is model parameters.

Task 3: The LASSO regression model was fitted to the training data (alpha=1).

Figure 1. Dependency of the regression coefficients on the log of penalty factor (log λ). LASSO.

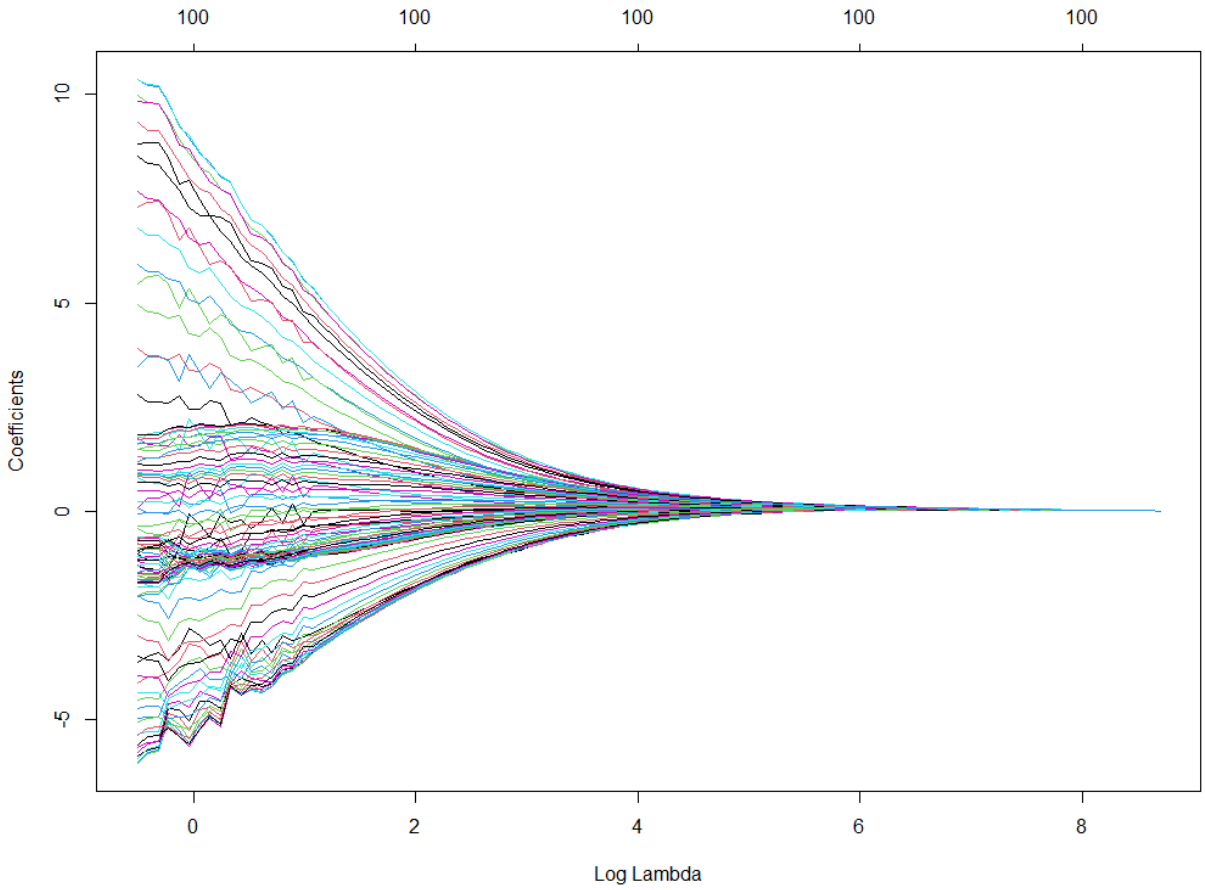


When increasing the penalty factor, we shrink the number of variables in the model. L1 regularization term forces parameters to zero and thus can be used as a feature selection method.

To select a model with only three features one can pick $\lambda =$ 0.853045182.

Task 4: After that we applied the Ridge regression to the data (alpha=0).

Figure 2. Dependency of the regression coefficients on the log of penalty factor (log λ). Ridge.

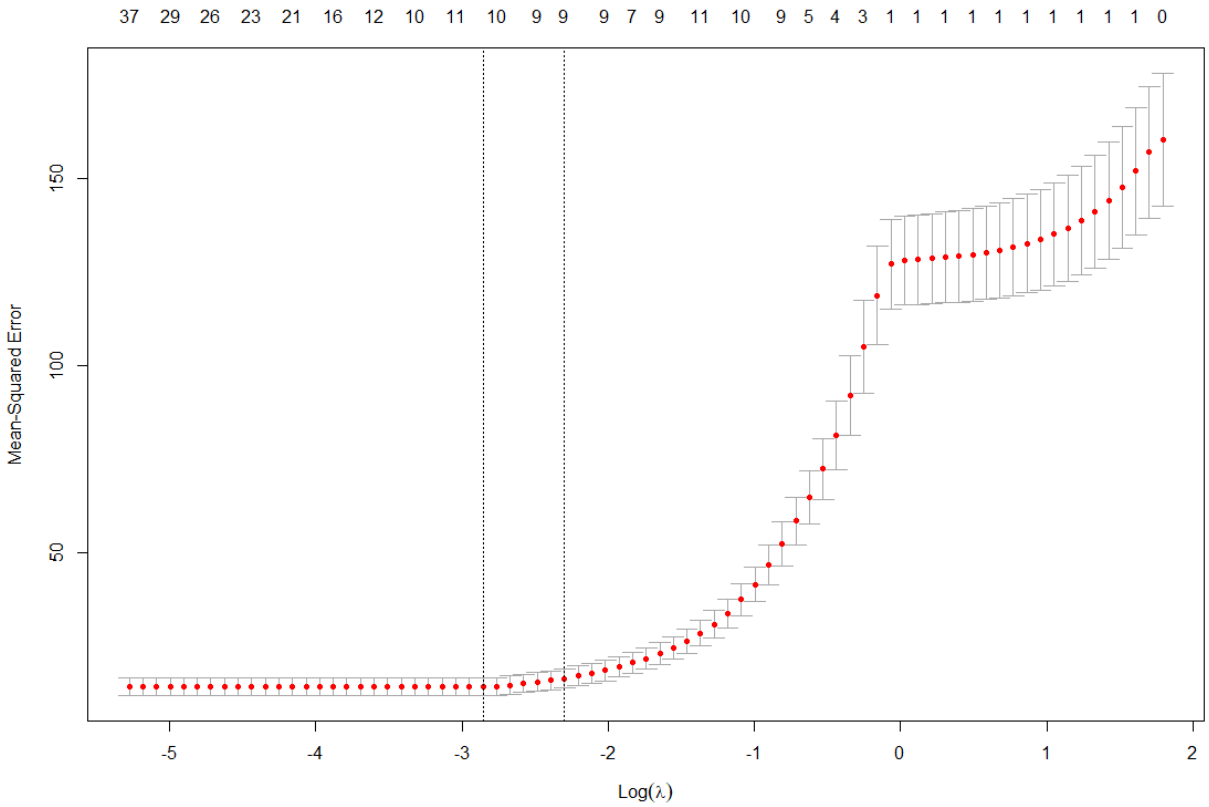


L2 regularization term pushes parameters to small values while λ increases. Unlike L1, it doesn't force them necessarily to zero. In contrast to Ridge, the LASSO regularization will actually set less-important variables to zero and thus helps to simplify the model and throw away less important parameters.

It may be beneficial to use L1 with a high-dimensionality data.

Task 5: Cross-validation process with the default number of folds was used to compute the optimal LASSO model

Figure 3. Cross-validation.



The dotted line on the left represents the optimal λ which minimizes cross-validation error. The dotted line on the right represents the point with one standard error of optimal λ. After that error grows.

As it can be seen from the cross-validation results, optimal $\lambda = 0.05745$ (with the log λ = -2.85684027469).

To understand the number of variables with optimal λ we built a LASSO model with alpha=1, lambda=0.05745. and figured out that 7 variables were used in the optimal model.

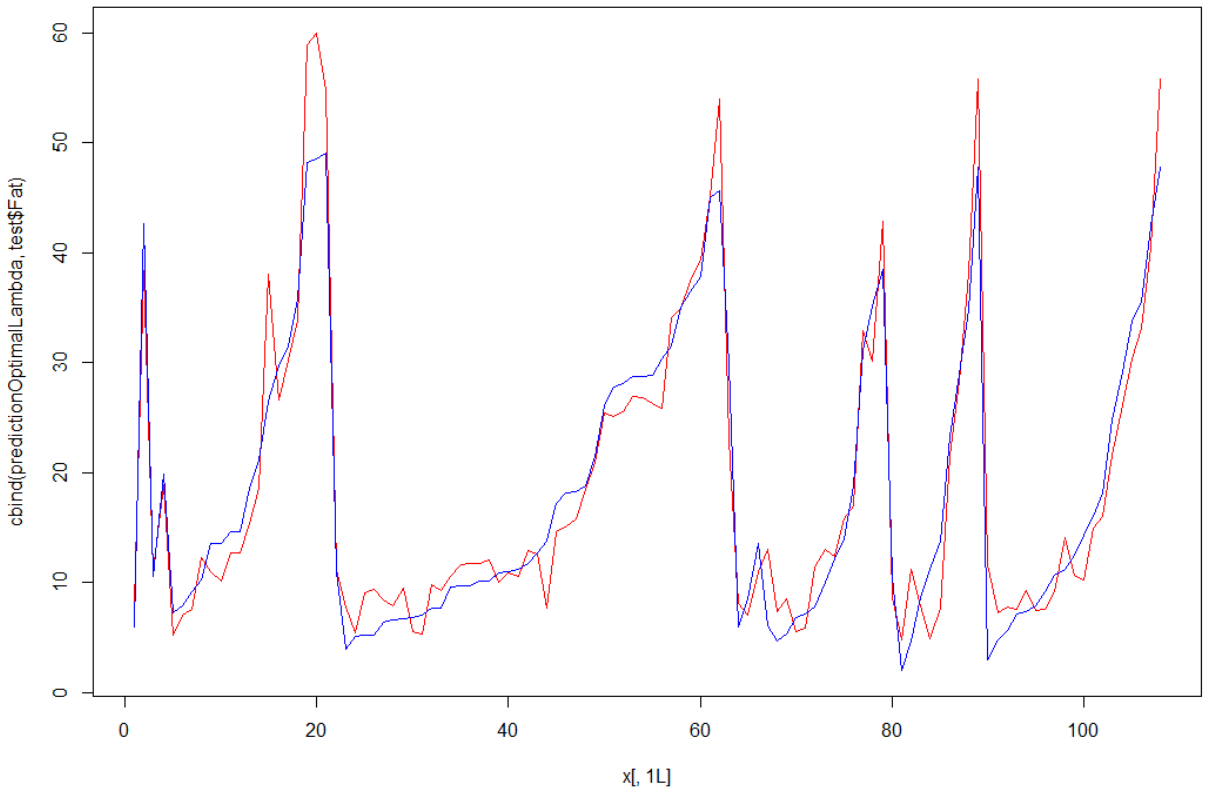
To understand if the optimal λ gives better prediction that λ with log λ = -4, we directly compared the error for those two λ values:

Error value for optimal λ	Error value for λ with log λ = -4
14.21175	14.22138

Although the value of the error for optimal λ is less than for log λ = -4, there is probably no significant statistical difference as the values are very close to each other.

Finally, we created a plot of the original test versus predicted test values for the model corresponding to optimal lambda.

Figure 4. Test data versus prediction with optimal λ.



It can be seen from the graph that the optimal model (red line) predicts the test data (blue line) quite well. If we check the MSE value for optimal model and compare it with the very first linear regression model, one can see significant improvement:

MSE (optimal model)	MSE (linear regression)
13.2998	982.2478

Assignment 2.

Task 1.

Bank.csv file is read, and the variable duration is removed as instructed. The data is portioned, with the code given by lecture 2a, into training, test and validation data sets. With the split 40/30/30.

The bank file contains information about bank clients which is to be used for a marketing campaign with the goal to get as many as possible to subscribe to the banks term deposits.

Task 2.

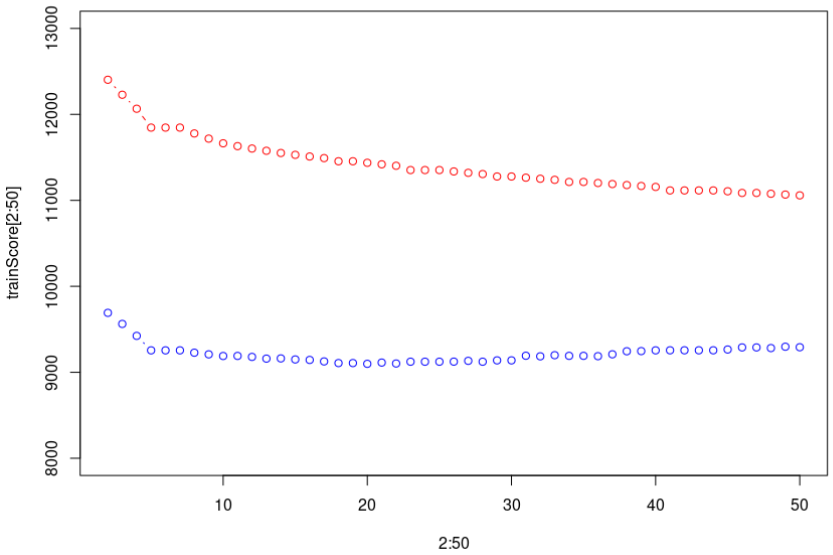
The R package tree was used for fitting the decision trees to the training data. Once that had been done confusion matrixes were created and used to calculate the misclassification rates for the three trees.

Decision Tree	**Misclassification Rate Train**	**Misclassification Rate Valid**
Default	0.1142446	0.1203274
Minimum Node Size 7000	0.1142446	0.1203274
Deviance Minimum = 0.0005	0.1045676	0.1448242

Deviance tree = 0,0005 overfitted more than the other two trees to the training data, so it is not the best model. The other two however have the same misclassification rate, so judging by that metric they are equally good.

Based on the misclassification rates we acquired it seems that the tree with higher deviance fit better to the training data but worse to the validation data set. While decision trees with the highest allowed node size at 7000 didn't make any difference compared to the default tree.

Task 3.



Dependence of deviances for the training and validation data on the number of leaves. Blue is validation and red is training.

First a little background, y-axis, trainscore shows deviance, which shows amounts or errors, high deviance equals high error rate. X-axis shows number of leaves on the decision tree which corresponds to how complex the decision tree model is. Because the complexity of a decision tree is determined by how deep it is. Depth is how many nodes there are from the root (start node) to leaf (end node). So, more leaves mean a longer path in a binary tree, where each node only have two paths.

In the graph we can see that the model learns the training data better and better so that the error goes down as complexity of the model increases. But the error in the validation set goes up, indicating increased overfitting and higher variance.

We can see high bias in the graph in the beginning, most likely because it is not a complex model at that point and we can see high error.

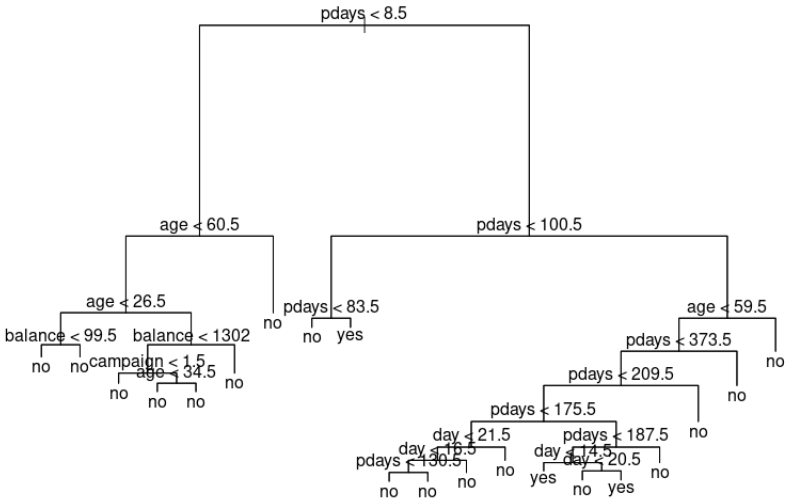
Bias-variance tradeoff means, as the name suggests, that as bias shrinks the variance in the model will increase and vice versa. Short description is that bias can be described as accuracy, how close we are to the actual values. While variance can be described as precision, how close each data point is to other data points. The goal with bias-variance tradeoff is to select a model where the total error is as small as possible. The model is on target and not to spread out.

This can be done by selecting a model that has a good enough amount of complexity, low complexity models give high bias, while high complexity models can overfit to the training data, meaning high variance.

The optimal amount of leaves seems to be from the graph 20 based on that for that amount of leaves the deviance, the errors are the lowest for the validation data.

Running summary on the tree with 50 leaves we get to know that the variables actually used in tree construction are:

pdays	age	balance	campaign	day



Tree structure for task 3.

Based on the structure as seen in the image above, there are some redundant branches, pdays<83.5 after pdays<8.5, same with a few other pdays. The model might be a bit to complex.

Task 4.

Optimal model was selected to the pruned model from the previous task. It was named bestTree.

The estimated confusion matrix for bestTree. Row no yes is prediction, column no yes is actual value.

predictionTest

	no	yes
no	11764	215
yes	1354	231

Accuracy	88.43262%
F1-score	22.74742%

Going by accuracy the model has good predictive power, while going by F1-score the model has poor. Worth mentioning is that there are roughly 16000 occurrences of class “yes” while there are 2000 of class “no” in the training data set, so the data set is imbalanced between its classes.

F1-score takes into account imbalances between classes while accuracy does not so the F1-score is the better metric for our model, so the model has poor predictive power.

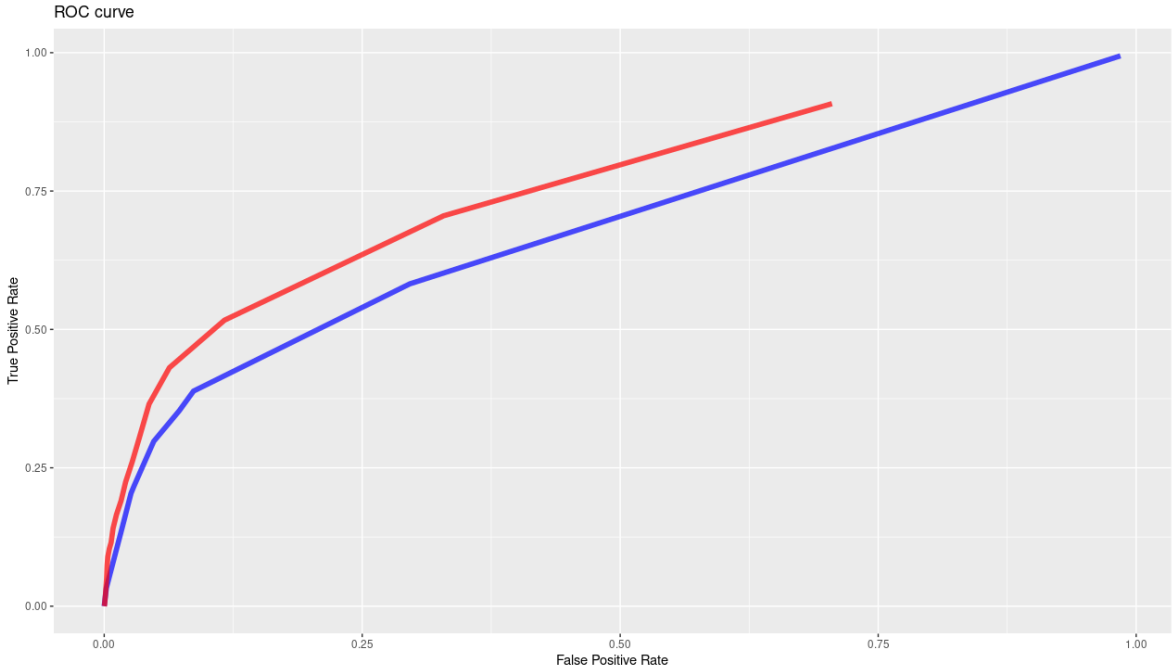
Task 5.

Accuracy	85.21085%
F1-score	38.04818%

Accuracy got slightly worse, it decreased with 3 percentage units, while F1-score increased by 16 percentage units. So, the models prediction power has gotten better.

What we did was that we weighted our probability table in a direction in order to make up for the existing imbalance between the two classes. So F1-score improved because we balanced our prediction by applying weights.

Task 6.



ROC curve. Red is the logistic regression model and blue is the decision tree.

True positive rate and false positive rate is calculated using a confusion matrix with the following formulas:

TPR = TP/P True Positive Rate = True Positive / Positives

FPR = FP/N False Positive Rate = False Positive / Negatives

The rates are used to evaluate a model, high TPR is good while low FPR is good, given the same FPR a higher TPR shows the better model.

These rates can be plotted in an receiver operating characteristics curve, ROC curve, which shows the performance for a classifier. Where the bigger the area under the graph the better.

In our ROC curve we can see that the logistic regression model is the better predictor since it has more area under the graph.

There exists an alternative to the ROC curve, the Precision Recall Curve, which is more informative for imbalanced problems, such as ours. While the Receiver Operating Characteristics, ROC curve is better for balanced problems.

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:

```
> print(eigenValues)
  [1] 2.501699e+01 1.693597e+01 9.301294e+00 7.556281e+00 5.662491e+00 4.238169e+00 3.226893e+00 2.967775e+00
  [9] 2.068278e+00 1.617654e+00 1.573263e+00 1.470866e+00 1.414572e+00 1.030961e+00 9.318540e-01 8.902341e-01
 [17] 7.486299e-01 7.052897e-01 6.488775e-01 6.386136e-01 6.245935e-01 5.683146e-01 5.418560e-01 5.192947e-01
 [25] 5.043199e-01 4.803330e-01 4.667517e-01 4.510651e-01 4.311566e-01 3.859230e-01 3.654056e-01 3.511289e-01
 [33] 3.366552e-01 3.109726e-01 2.874555e-01 2.590034e-01 2.561964e-01 2.454743e-01 2.399923e-01 2.218195e-01
 [41] 2.115549e-01 2.054834e-01 1.999320e-01 1.897542e-01 1.827631e-01 1.647770e-01 1.604766e-01 1.415854e-01
 [49] 1.385136e-01 1.263270e-01 1.115563e-01 1.070502e-01 1.040590e-01 9.968511e-02 8.984441e-02 8.146132e-02
 [57] 7.699592e-02 7.270725e-02 6.914821e-02 6.569790e-02 6.394932e-02 6.192272e-02 5.820181e-02 5.090331e-02
 [65] 4.884623e-02 4.582648e-02 4.368481e-02 4.182327e-02 4.025564e-02 3.796697e-02 3.538585e-02 3.386271e-02
 [73] 3.187147e-02 3.054973e-02 2.806148e-02 2.552080e-02 2.384422e-02 2.244957e-02 2.105203e-02 2.004971e-02
 [81] 1.837688e-02 1.732693e-02 1.549645e-02 1.385309e-02 1.265076e-02 1.144285e-02 9.288740e-03 8.322620e-03
 [89] 5.992804e-03 5.752431e-03 5.136683e-03 4.105187e-03 3.791326e-03 2.739526e-03 2.034851e-03 1.774074e-03
 [97] 1.299604e-03 1.106855e-03 8.075157e-04 6.298098e-04
```

EigenVectors:

```
> print(eigenVectors)
              [,1]         [,2]         [,3]         [,4]         [,5]         [,6]         [,7]         [,8]
 [1,] -0.006204668  0.090167117 -0.032056620 -0.042622242  0.0781090743 -1.061402e-01 -0.032594509 -0.143330950
 [2,] -0.053178417 -0.089033431  0.147339059  0.086714984  0.2788641060  8.457993e-03  0.088820546 -0.083429862
 [3,]  0.009010122 -0.112383111 -0.224097963 -0.097842434  0.1202811287 -8.690254e-02 -0.010280185  0.074733532
 [4,] -0.107496674 -0.007033133  0.079684672  0.030452797  0.0541982698 -5.777257e-02 -0.256847098  0.245462679
 [5,]  0.116370033  0.118352211 -0.004172283 -0.034483666 -0.0198667619  2.156196e-02  0.171641613 -0.194866388
 [6,]  0.046547236 -0.167497051  0.033249844  0.001934763 -0.0555453910  3.261972e-02 -0.002989670 -0.041197503
 [7,] -0.068053479 -0.161319185 -0.134269675  0.032813657 -0.0331905171  5.227327e-02  0.127633980 -0.009897020
 [8,] -0.062368666 -0.033362428 -0.057945514 -0.241443828  0.0682252099 -2.225263e-01  0.010706485  0.001305117
 [9,] -0.074031271 -0.077669560  0.015307859 -0.255366651  0.0373865098 -1.188692e-01 -0.065752414 -0.087520040
[10,] -0.063488181 -0.049240064  0.037113992 -0.244075543  0.0059854623 -2.127792e-01 -0.008411724 -0.074281684
```

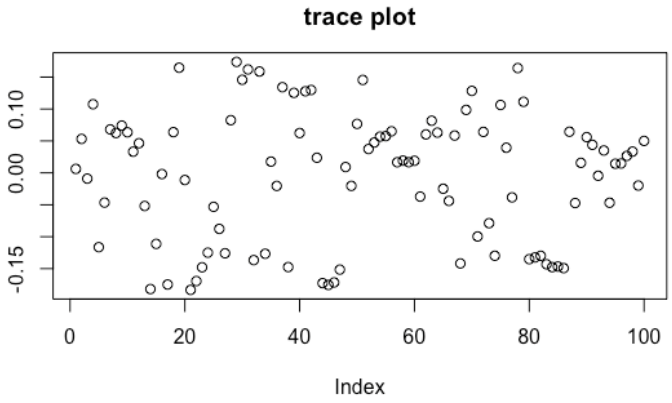
Based on our computing for obtaining 95% of variance in the data we need 35 components. To get this result we calculate proportion of variation then made a data frame and calculate cumulative variance percent and then count those that had the variance more than 95%. And the proportion of variation by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

```
> print(variances)
            eigenValues                sd variance.percent cumulative.variance.percent
1        25.016987499055   5.00169846142838             25%              25.016987499055%
2       16.9359743082565   4.11533404576791             17%             41.9529618073115%
3       9.30129430373109   3.04980233846902            9.3%             51.2542561110426%
4       7.55628133310631   2.74886910075877             7.6%            58.8105374441489%
5       5.66249090635042   2.37959889610632             5.7%            64.4730283504993%
6        4.2381692205545   2.05868142765084             4.2%            68.7111975710538%
7       3.22689305394301   1.79635549208474             3.2%            71.9380906249969%
8       2.96777526603681   1.72272321225344               3%            74.9058658910337%
9       2.06827797216574   1.43815088643916             2.1%            76.9741438631994%
10      1.61765372516666   1.27187016836101             1.6%            78.5917975883661%
11      1.57326274276652    1.2542977089856             1.6%            80.1650603311326%
12      1.47086571350994    1.2127925269847             1.5%            81.6359260446425%
13      1.41457243484978   1.18935799272119             1.4%            83.0504984794923%
14      1.03096097703155   1.01536248553487               1%            84.0814594565239%
15     0.931853959114953  0.965325830543736            0.93%            85.0133134156388%
16     0.890234067498358  0.943522160576188            0.89%            85.9035474831372%
17     0.748629907897528  0.865234019151771            0.75%            86.6521773910347%
18     0.705289671782928  0.839815260508481            0.71%            87.3574670628176%
19     0.648877491934559  0.805529324068689            0.65%            88.0063445547522%
20      0.63861361600466  0.799133040240897            0.64%            88.6449581707569%
21     0.624593497687759  0.790312278588507            0.62%            89.2695516684446%
22     0.568314581494321  0.753866421519304            0.57%            89.8378662499389%
23     0.541856006840806  0.736108692273639            0.54%            90.3797222567797%
24     0.519294695505169  0.720621048474973            0.52%            90.8990169522849%
25     0.504319920357656  0.710154856603583             0.5%            91.4033368726426%
26     0.480333025690931  0.693060621945101            0.48%            91.8836698983335%
27     0.466751666647488  0.683192261846903            0.47%             92.350421564981%
28     0.451065134627303  0.671613828496185            0.45%            92.8014866996083%
29     0.431156601076065  0.656625160252077            0.43%            93.2326433006843%
30     0.385923040933233  0.621227044592581            0.39%            93.6185663416176%
31     0.365405564069947  0.604487852706691            0.37%            93.9839719056875%
32     0.351128947365652  0.592561344812208            0.35%            94.3351008530532%
33     0.336655199919984  0.580219958222728            0.34%            94.6717560529732%
34     0.310972633820804  0.557649203192118            0.31%             94.982728686794%
35     0.287455486664795  0.536148754232251            0.29%            95.2701841734588%
36     0.259003390197514  0.508923756762753            0.26%            95.5291875636563%
37     0.256196358531998  0.506158432244291            0.26%            95.7853839221883%
38     0.245474326439477  0.495453657206683            0.25%            96.0308582486278%
39     0.239992297181386  0.489890086837227            0.24%            96.2708505458092%
40     0.221819479433944  0.470977153834392            0.22%            96.4926700252431%
41     0.211554944684882  0.459951024224191            0.21%             96.704224969928%
42      0.20548344438333  0.453302817532971            0.21%            96.9097084143113%
43     0.199931987616164  0.447137548877484             0.2%            97.1096404019275%
```

Assignment3: Task2

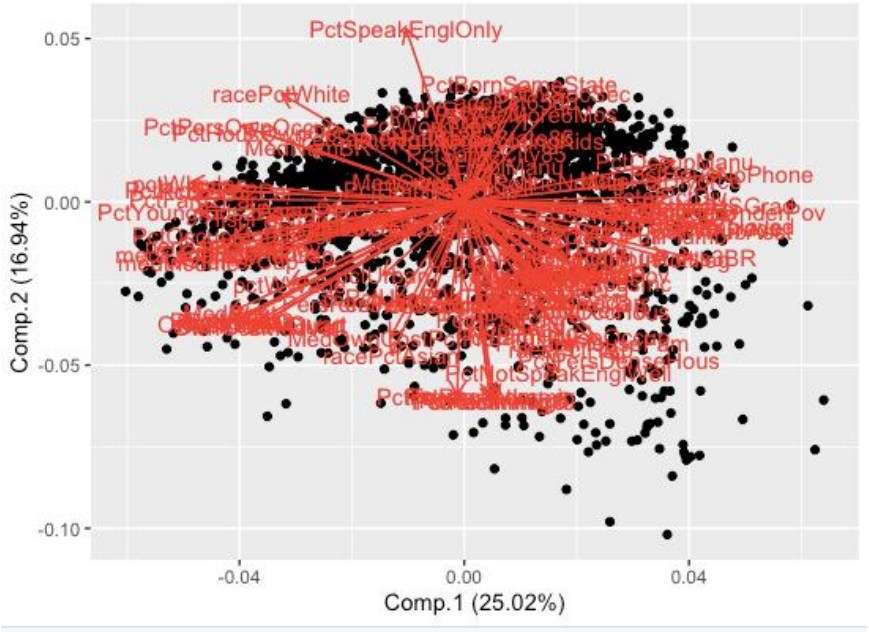
In this task we used princomp() function to have PCAs. We draw 3 different plots. The first one did not contain readable information

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



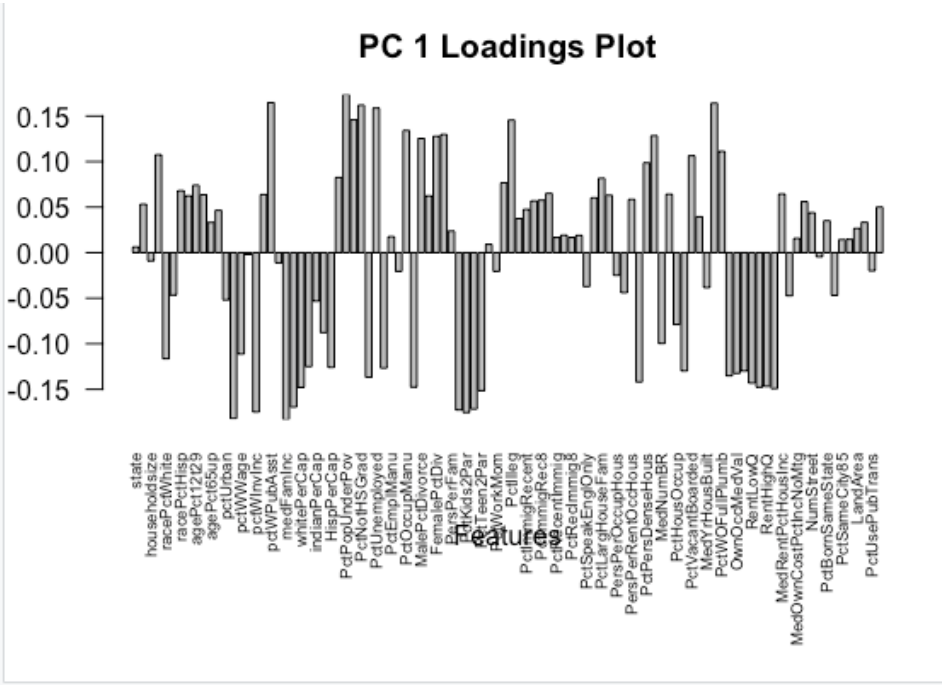
We used another plot it was a mess and not readable

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

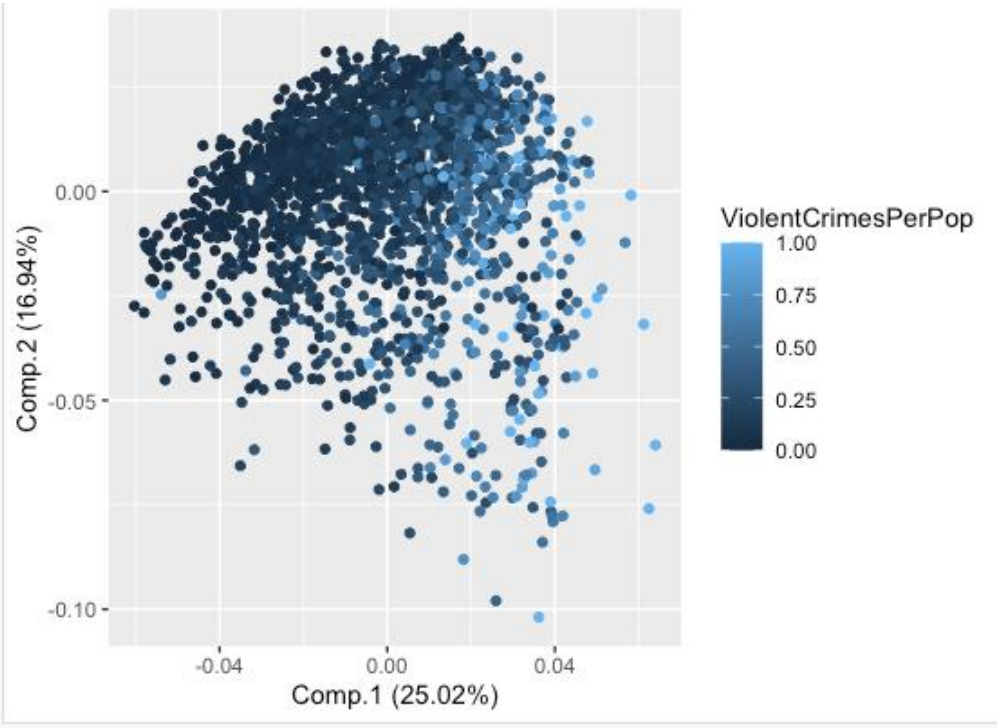
3. PctKids2Par: percentage of kids in family housing with two parents

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: $\min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i\boldsymbol{\theta})^2 = \min_{\boldsymbol{\theta}} (y - \boldsymbol{X\theta})^T(y - \boldsymbol{X\theta})$

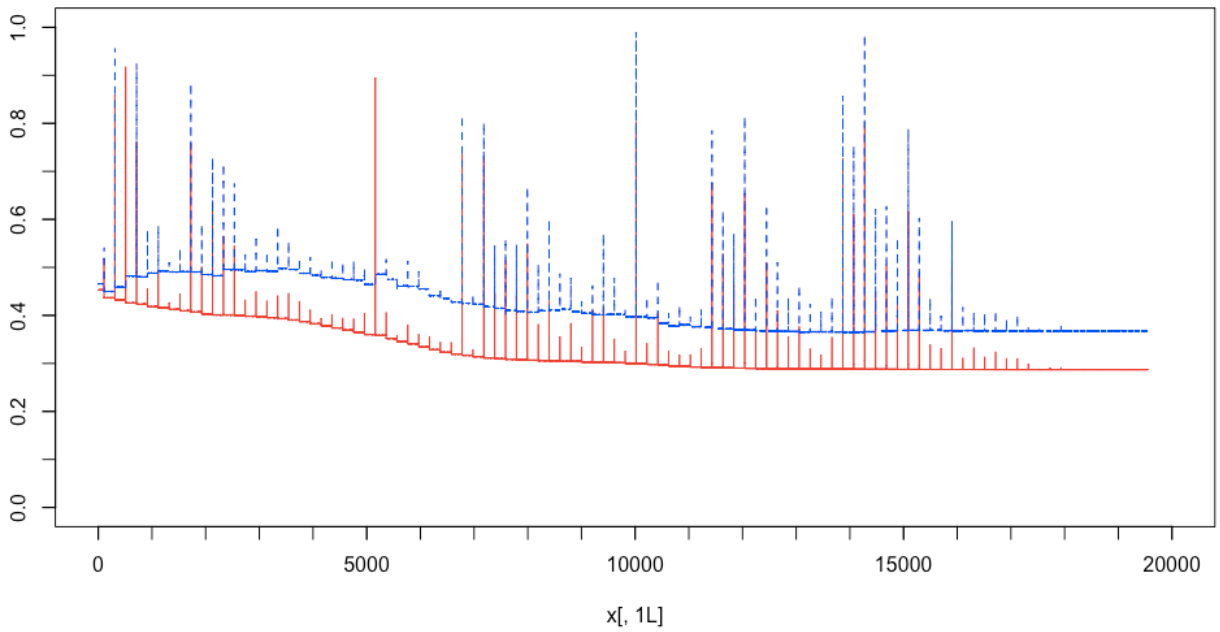
Usual choice: Squared loss $L(\hat{y}(x_i, \boldsymbol{\theta}), y_i) = (\hat{y}(x_i, \theta) - y_i)^2$

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.

```
> print(result)
$par
  [1] -0.0712615669  0.0308788758 -0.0655363855  0.3281858328 -0.0403881770  0.0290211012 -0.0104572326  0.0153828124
  [9] -0.0182064586 -0.1437591956 -0.0110515814 -0.0408061650  0.1042633564 -0.0469555694 -0.0979702972  0.0404873346
 [17] -0.0490965456  0.1210928934  0.0226738480 -0.0810741510 -0.0001879188  0.1159203577 -0.4096763719 -0.0260214996
 [25]  0.0025372680  0.0558673855 -0.0210889802  0.1361011231 -0.2477436109 -0.1563141337  0.1276364920  0.1559754150
 [33]  0.0050894481  0.1892324045 -0.1368776152 -0.0962014094  0.1032634464  0.1044827129  0.5224563125  0.2049859973
 [41]  0.3531957838 -0.8871058245 -0.1610855525  0.1067799288 -0.3107725588 -0.0540059813  0.0348982717  0.0169541883
 [49] -0.1600980496 -0.0806914943  0.0833494007 -0.0530851580  0.0406800991 -0.0208417840 -0.0888962397  0.0586210191
 [57] -0.1816512662 -0.1691608354  0.6546351156 -0.3249132345 -0.0159105541 -0.0795346263  0.0640838035 -0.1336571293
 [65]  0.3673331557  0.0080998684 -0.1823423588 -0.6210691844  0.1286496789  0.0556440068  0.0869812294  0.0469606760
 [73] -0.0665687821  0.4818316083  0.0446301310 -0.0690409109 -0.0324542724  0.0963100500  0.0116338682 -0.4565778756
 [81]  0.0435227587  0.3300838069 -0.3123768392  0.1349172611 -0.0200787297  0.2545332411  0.0349258559 -0.0466614985
 [89] -0.0500605125  0.0493030766  0.0270745691  0.1765915193 -0.0039824362 -0.0154246429  0.0174979005  0.0537684662
 [97] -0.0013275719  0.0150424665 -0.0525765137 -0.0153676145
```

The picture above shows the optimal theta from the 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 6000 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.

Appendix. Code. Task 1.

Assignment 1.

```
##############################################Preparations##############################################
tecator=read.csv("tecator.csv")

#REMOVE UNNECESSARY COLUMNS FIRST
tecator$Protein=c()
tecator$Moisture=c()
tecator$Sample=c()

#dividing the train and test
set.seed(12345)
n=nrow(tecator)
id=sample(1:n, floor(n*0.5))
train=tecator[id,]
test=tecator[-id,]

#model based on training data
linearModel=lm(Fat~., data=train)
sum=summary(linearModel)



##############################################Task 1.##############################################
#MSE for train and test
MSEtrain=mean(sum$residuals^2)

#calculate MSE for test data set
predictionTest=predict(linearModel,newdata=test,interval = "prediction")
MSEtest=mean((test$Fat-predictionTest)^2)
```

```
##############################################Task 2.##############################################

#cost function - see the report

##############################################Task 3.##############################################
library(glmnet)
library(dplyr)
#creating lasso regression model
x=as.matrix(train%>%select(-Fat))
y=as.matrix(train%>%select(Fat))
LASSOmodel=glmnet(x, y, alpha=1)
plot(LASSOmodel, xvar = "lambda")



##############################################Task 4.##############################################
#Ridge regression and  plot of lambda
ridgeRegressionModel=glmnet(x, y, alpha=0)
plot(ridgeRegressionModel, xvar="lambda")

##########################################Task 5.##############################################
#cross validation
modelCrossValidation <- cv.glmnet(x=x, y=y, alpha=1, family="gaussian")
plot(modelCrossValidation)
#lambda min is 0.05745
print(modelCrossValidation$lambda.min)

#build a model with minLambda and see how many variables are there
LASSOmodelOptimalLambda=glmnet(x, y, alpha=1,lambda=modelCrossValidation$lambda.min)
print(LASSOmodelOptimalLambda$beta)
#there are 7 variables for the model with optimal lambda



# 0.01831563888 is lambda for  log λ = −4, it is located somewhere between 63 and 64 position in lambda array of cross-validation dataset
print(modelCrossValidation$cvm[63])
```

```
print(modelCrossValidation$cvm[64])
print(modelCrossValidation$cvm[51])
print(modelCrossValidation$cvsd[63])
print(modelCrossValidation$cvsd[64])
print(modelCrossValidation$cvsd[51])

#plot of the original test versus predicted test values for the model corresponding to optimal lambda
x1=as.matrix(test%>%select(-Fat))
predictionOptimalLambda=predict(object = LASSOmodelOptimalLambda,newx=x1,s="lambda.min")
matplot(y=cbind(predictionOptimalLambda,test$Fat),type="l",col=c("red","blue"),lty=c(1,1))


#check mse for linear regression model in task 1 and compare with mse for optimal model

MSEOptimalLambda=mean((predictionOptimalLambda-test$Fat)^2)
print(MSEOptimalLambda)
```

Code Appendix Assignment 2

```
#####Task 1####

bank=read.csv("bank-full.csv",sep=";")


#REMOVE UNNECESSARY COLUMNS FIRST

bank$duration=c()


#dividing the train, valid and test

n=dim(bank)[1]

set.seed(12345)

id=sample(1:n, floor(n*0.4))

train=bank[id,]

id1=setdiff(1:n, id)

set.seed(12345)

id2=sample(id1, floor(n*0.3))

valid=bank[id2,]

id3=setdiff(id1,id2)

test=bank[id3,]


#####Task2 decision trees#####

#default tree

#install.packages("tree")

library("tree")

defaultTree=tree(as.factor(y)~.,data=train)

plot(defaultTree)

print(defaultTree)

#tree with the smallest node size of 7000

Tree7000=tree(as.factor(y)~.,data=train,minsize=7000)

plot(Tree7000)

print(Tree7000)


#tree with the deviance to 0.0005
```

```
TreeDeviance00005=tree(as.factor(y)~.,data=train,mindev=0.0005)
plot(TreeDeviance00005)
print(TreeDeviance00005)

#confusion matrixes for the trees
#default
TreeFit=predict(defaultTree,newdata=train, type="class")
table(train$y,TreeFit)
TreeFitValid=predict(defaultTree,newdata=valid, type="class")
table(valid$y,TreeFitValid)

#nodesize7000
TreeFit2=predict(Tree7000,newdata=train, type="class")
table(train$y,TreeFit2)
TreeFit2Valid=predict(Tree7000,newdata=valid, type="class")
table(valid$y,TreeFit2Valid)

#deviance minimum0.0005
TreeFit3=predict(TreeDeviance00005,newdata=train, type="class")
table(train$y,TreeFit3)
TreeFit3Valid=predict(TreeDeviance00005,newdata=valid, type="class")
table(valid$y,TreeFit3Valid)

#missclassifications rate training and test
MCETrainDefaultTree=mean(TreeFit != train$y)
MCETrainTree7000=mean(TreeFit2 != train$y)
MCETrainDeviance00005=mean(TreeFit3 != train$y)
MCEValid=mean(TreeFit != valid$y)
MCEValidTree7000=mean(TreeFit2 != valid$y)
MCEValidDeviance00005=mean(TreeFit3 != valid$y)

#####Task 3#####
```

```
#creating emptyty vectors
trainScore=rep(0,50)
validScore=rep(0,50)

#Filling empty vectors with deviance scores from the pruned tree
for(i in 2:50) {
  prunedTree=prune.tree(TreeDeviance00005,best=i)
  pred=predict(prunedTree, newdata=valid,type="tree")
  trainScore[i]=deviance(prunedTree)
  validScore[i]=deviance(pred)
}

#graph of the dependence of deviances for the training and the validation data on the number of leaves
plot(2:50, trainScore[2:50], type="b", col="red", ylim=c(8000,13000))
points(2:50, validScore[2:50], type="b", col="blue")
min(validScore[2:50])

#fit for tree with optimal amount of leaves
bestTree = prune.tree(TreeDeviance00005,best=20)
#info for besttree such as varaiables actualy used
summary(bestTree)

#Plotting bestTree to see structure
plot(bestTree)
#adding text for readability
text(bestTree)
#more info
print(bestTree)

#####Task4######
#Creating predicted values for test data using decision tree bestTree.
predictionTest=predict(object=bestTree, newdata=test,type="class")
```

```
#Creating confusion matrix for the prediction
CMBestTree = table(test$y,predictionTest)
print(CMBestTree)

#calculating accuracy for the model.
accuracy <- sum(diag(CMBestTree)) / sum(CMBestTree)
cat(accuracy*100,"%")

#Precision: Correct positive predictions relative to total positive predictions
precision = CMBestTree[2,2]/(CMBestTree[2,2]+CMBestTree[2,1])
#Recall: Correct positive predictions relative to total actual positives
recall=CMBestTree[2,2]/(CMBestTree[2,2]+CMBestTree[1,2])
F1score=2*recall*precision/(recall+precision)
cat(F1score*100,"%")

#imablanced dataset?
print(length(train$y[train$y=="no"]))
print(length(train$y[train$y=="yes"]))

#####Task5#####
#Predicted loss matrix
LossMatricePredict = predict(bestTree, newdata = test)
#applying lossmatrix to predition table
weightedLosspred = ifelse(LossMatricePredict[,1]/LossMatricePredict[,2] > 5, "no", "yes")
#building confusion matrix between prediction with loss matrix and real test data
CMLossMatriceTree = table(yreal=test$y,ypred=weightedLosspred)
print(CMLossMatriceTree)

#calculating accuracy
accuracy1 <- sum(diag(CMLossMatriceTree)) / sum(CMLossMatriceTree)
cat(accuracy1*100,"%")
```

```
#Precision: Correct positive predictions relative to total positive predictions
precision1 = CMLossMatriceTree[2,2]/(CMLossMatriceTree[2,2]+CMLossMatriceTree[2,1])
#Recall: Correct positive predictions relative to total actual positives
recall1=CMLossMatriceTree[2,2]/(CMLossMatriceTree[2,2]+CMLossMatriceTree[1,2])
#calculating F1-score
F1score1=2*recall1*precision1/(recall1+precision1)
cat(F1score1*100,"%")

#####Task6#####
#fit for logistic regression model
logisticRegressionModel=glm(as.factor(train$y)~.,data=train,family=binomial())
#creating a vector of the threshholds
vectorOfπ=seq(0.05, .95, by=0.05)

#empty lists
TPRforTree = c()
FPRforTree = c()

#loop for calculating TPR and FPR for each threshold value of pi for decision tree
for(i in 1:19) {
 #predicting y for the decision tree
 predictionTree2 = predict(object=bestTree, newdata = test)
 # we check if predictionTree2 for "Yes" > Each π
 YHatclassifiedTree= ifelse (predictionTree2[,2] > vectorOfπ[i] , "yes", "no")
 print(vectorOfπ[i])

 CMforTree= table(ytraintrue=test$y ,ytrainpred=factor(YHatclassifiedTree,levels=c("no","yes")))
 print(CMforTree)
 #now we can compute TPR and FPR
 TPRforTree[i] = CMforTree[2,2]/sum(CMforTree[2,])
 FPRforTree[i] = CMforTree[1,2]/sum(CMforTree[1,])
}
```

```
#empty lists
TPRforLM = c()
FPRforLM = c()


#the same for logistic regression
for(i in 1:19) {

  predictionLM = predict(object=logisticRegressionModel, newdata = test,type="response")
  # we check if prediction for "Yes" > Each π
  YHatLM= ifelse (predictionLM > vectorOfπ[i] , "yes", "no")
  print(vectorOfπ[i])

  CMforLM= table(ytraintrue=test$y ,ytrainpred=YHatLM)
  print(CMforLM)
  #now we can compute TPR and FPR
  TPRforLM[i] = CMforLM[2,2]/sum(CMforLM[2,]) #TP/P True positive divided with all positive
  FPRforLM[i] = CMforLM[1,2]/sum(CMforLM[1,]) #FP/N False positive divided by all negatives
}


library(ggplot2)

df <- data.frame(tprtree=TPRforTree,fprtree=FPRforTree,tprlm=TPRforLM,fprlm=FPRforLM)
#ROC plot for the tree
ggplot(data=df)+
  geom_line(mapping=aes(FPRforTree,TPRforTree), color = "blue",  size = 2, alpha = 0.7)+
  geom_line(mapping=aes(FPRforLM, TPRforLM), color = "red", size = 2, alpha = 0.7)+
  labs(title= "ROC curve", x = "False Positive Rate", y = "True Positive Rate")+
  scale_color_manual(labels = c("Tree", "LM"), values = c("blue", "red"))
```

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])
```