732A99/732A68/ TDDE01 Machine Learning

Computer lab 2 block 1

Johannes Hedström, Mikael Montén & Siddhesh Sreedar

STIMA
Department of Computer and Information Science
Linköpings universitet

2024-01-10

Contents

1 Contribution of work

Mikael did assignment 1, Siddhesh did assignment 2, Johannes did assignment 3.

2 Assignment 1. Explicit regularization

The given CSV-file 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.

Divide data randomly into train and test (50/50) by using the codes from the lectures.

2.1 1.

Assume that Fat can be modeled as a linear regression in which absorbance characteristics (Channels) are used as features. Report the underlying probabilistic model, fit the linear regression to the training data and estimate the training and test errors. Comment on the quality of fit and prediction and therefore on the quality of model.

Table 1: Error estimates on train and test

Training error	Test error
0.0057091	722.4294

The underlying probabilistic model for this linear regression is . The training error is 0.0057 and the test error is 722.43. Comparing the errors we can see why the big glaring problem with this model, which is its overfitting. A test error of such larger magnitude means the model has too many parameters to consider making it too constrained to the specific data and less prone to optimal generalization which is required to perform well at test data. Both errors are calculated by the MSE. The test MSE being so high means that the euclidean distance between true value and predicted value is very high and therefore the prediction is misleading.

Table 2: Quality of Fit for linear regression

R^2	R^2_{adj}
0.9999637	0.9993582

The underlying probabilistic model is $Y_i = \beta_0 + \beta_k X_k + \epsilon, k = 1 - 100$.

Above is the quality of fit for the model. R^2 shows that the variables explain 99.99 of variance in the target variable Fat, and R^2_{adj} shows 99.93% variance explained but is penalized for an increase in number in variables. The fact that R^2_{adj} is very high as well is an indication that the addition of variables has increased complexity but also information gain regarding Fat. It's important to note that the quality of fit is measured on the training data. $R^2 = 1 - \frac{SSE}{SST}$ and $R^2_{adj} = 1 - \frac{SSE/(n-k)}{SST(n-1)}$, where k is the amount of parameters in the model.

They both measure the squared regression error divide by the total error but R^2_{adj} is penalized by having more variables which is seen in the denominator where higher k variables increase the residual.

It's easy to look at the R^2 and proclaim the model is very good despite its' complexity. However, evaluation of the training vs test errors tells us that the quality of the model is quite subpar for predictive modelling in its' current state.

2.2 2.

Assume now that Fat can be modeled as a LASSO regression in which all Channels are used as features. Report the cost function that should be optimized in this scenario.

The cost function to be minimzed is the L^1 regularised cost function which is $\hat{\theta} = \arg\min_\theta \frac{1}{n}||X\theta - y||_2^2 + \lambda||\theta||_1$

2.3 3.

Fit the LASSO regression model to the training data. Present a plot illustrating how the regression coefficients depend on the log of penalty factor (logλ) and interpret this plot. What value of the penalty factor can be chosen if we want to select a model with only three features?

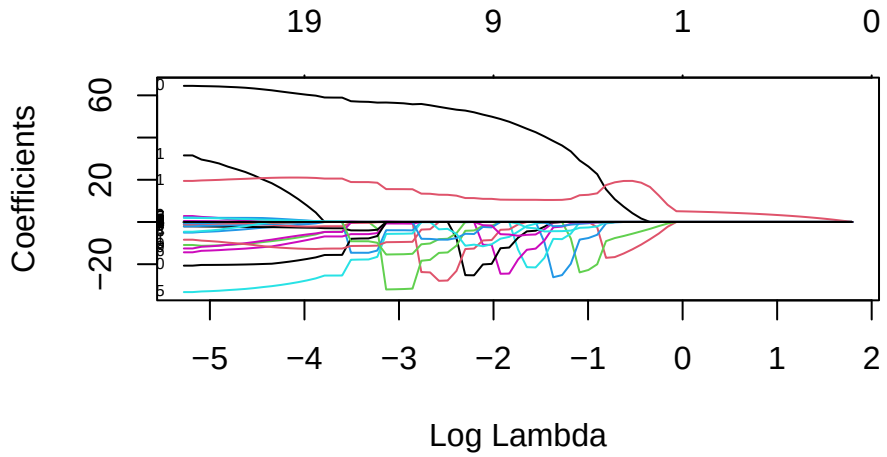


Figure 1: Coefficient values and amounts depending on penalty factor LASSO

Each line in this plot is an individual parameter coefficient. The Y-axis shows the value of the coefficient dependent on the iteratively increasing $log\lambda$ which is the penalty factor. The above X-axis is the amount of coefficients that are still non-zero. Many oefficients converge to 0 at a penalty factor value of around -3.8 and -3.2. Subsequently from this point a majority of coefficients converge to 0 and eventually all coefficients converge to 0 at a $log\lambda$ value of around 1.8. The last parameter to converge is alone for a large time before converging to the intercept model meaning it carries alot of information regarding the response. If we want to set a penalty factor to get a three feature model we would choose $log\lambda = -0.4 \implies \lambda = exp(-0.4) = 0.67032$

2.4 4.

Repeat step 3 but fit Ridge instead of the LASSO regression and compare the plots from steps 3 and 4. Conclusions?

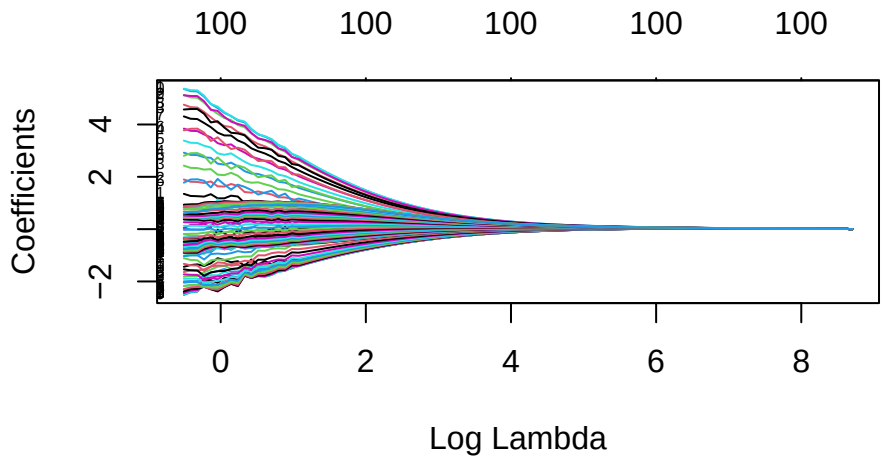


Figure 2: Coefficient values and amounts depending on penalty factor RIDGE

The two regularization techniques have different calculations regarding $log\lambda$ and it's easy too see the small difference replacing the square with absolute values in the penalizing term of the cost function. Ridge regression also converges to similar result as the LASSO, in that it eventually converges to approximately 0 and the last parameter is minimized to approximately 0 a great deal of iterations later than the second last. The difference here is important, where LASSO actually gives the parameters a value of 0, Ridge compresses them to a value close to zero but the parameter is still formally in the model - which is why the 100 number on the top of the graph does not decrease.

2.5 5.

Use cross-validation with default number of folds to compute the optimal LASSO model. Present a plot showing the dependence of the CV score on $log\lambda$ and comment how the CV score changes with $log\lambda$. Report the optimal $log\lambda$ and how many variables were chosen in this model. Does the information displayed in the plot suggests that the optimal $log\lambda$ value results in a statistically significantly better prediction than $log\lambda = -4$? Finally, create a scatter plot of the original test versus predicted test values for the model corresponding to optimal lambda and comment whether the model predictions are good.

The CV is measured by MSE. The plot shows MSE from 37 included covariates down to 0. This means the model heavily overfits for models above 37 covariates and those are immediately discarded. We can see that the model has chosen 9 covariates as the optimal, cause after that the MSE starts increasing again meaning the model seems prone to underfit.

The optimal $log\lambda$ is around -2.5, meaning $\lambda = exp(-2.5) = 0.082085$ which converges to 9 covariates. The confidence intervals are plotted in the graph as the gray areas, and comparing the point estimation for the

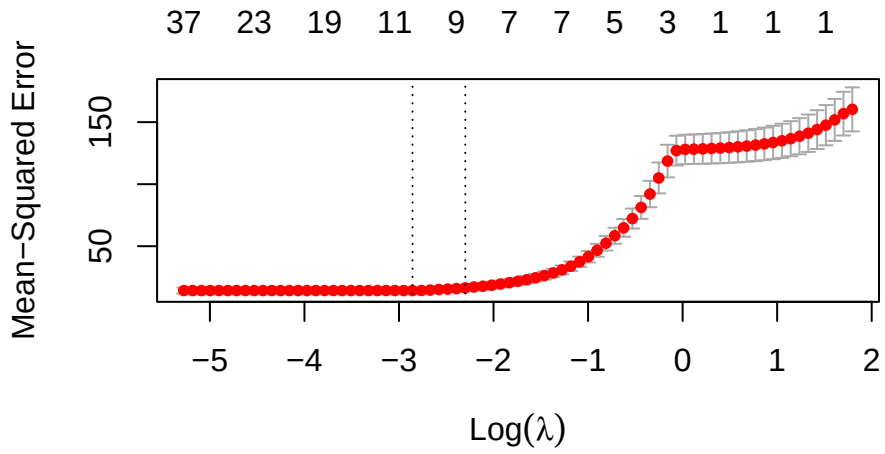


Figure 3: MSE for each log lambda

optimal log lambda with the point estimation for log lambda = -4, we can see that the confidence interval surrounding the points does overlap each other and therefore its not statistically significantly a better prediction with the optimal $log\lambda$-value compared to -4.

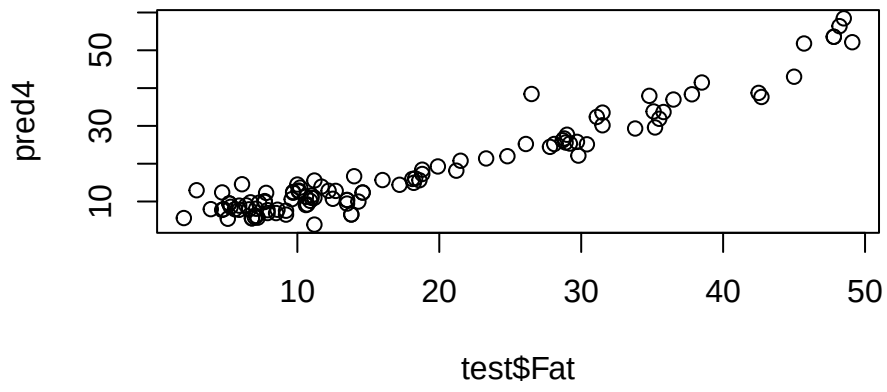


Figure 4: Fitted vs predicted values from new model

The prediction error for this model has been greatly improved compared to the first model. For this model we can see that y and $\hat{y}$ follow each other in a positive linear manner. There is however a slight increase in slope for higher values which is indicative of the model having a harder time predicting values for higher values of Fat.

The benefits from the LASSO regression is obvious though. By penalizing the regression model and going from 100 to only 9 features, the model does not overfit on data any more and is instead able to handle new data and making good predictions for the general case regarding these features and target.

3 Assignment 2. Decision trees and logisitc regression for bank marketing

3.1 1)

```
data_2<-data_1[,-12]

n=dim(data_2)[1]
set.seed(12345) #Set seed first
id = sample(1:n, floor(n*0.4))
train = data_2[id,]

id1 = setdiff(1:n,id)
set.seed(12345) #Set seed second
id2 = sample(id1, floor(n*0.3))
val = data_2[id2,]
```

```r
id3 = setdiff(id1,id2)
test = data_2[id3,]
```

3.2 2)

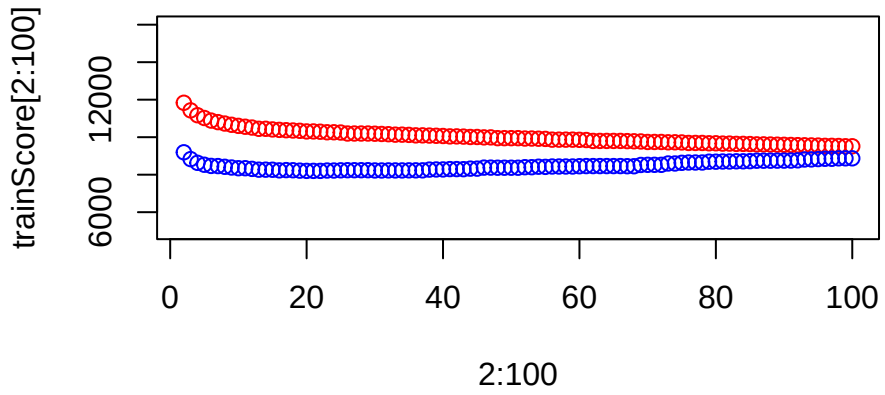
```
## $`Model1 train`
## [1] 0.1048441
##
## $`Model1 val`
## [1] 0.1092679

## $`Model2 train`
## [1] 0.1048441
##
## $`Model2 val`
## [1] 0.1092679

## $`Model3 train`
## [1] 0.09400575
##
## $`Model3 val`
## [1] 0.1119221
```

Based on the 3 models, the first 2 models have the same misclassifications rate while model3 has a higher missclassification rate for the validation data and relatively lower misclasssifcation rate for the training data. Based on the tree sturcture of model1 and model2, we choose model2 as its less complex.

Report how changing the deviance and node size affected the size of the trees and explain why: As the minimum node size value is increased from its default of 10, the size of the tree is decreased. The node size is the amount of data that is expected in each leaf of the tree. So a large node size would imply a shorter tree, while a smaller node size would result in more splits.

As the minimum deviance value is reduced from its default of 0.01, the size of the tree is increased. Deviance is an impurity measure, so we want to reduce it. So as we reducing this value, we expect more purity in each split which is one the reason why in this model, the size of the tree has increased.

3.3 3)



Interpret this graph in terms of bias-variance tradeoff: From the graph for the training data, we can see that as we increase the number of leaves, the error continues to reduce and this could eventually lead to overfitting (high bias) and we can see that from the plot of the validation data that after a point the error rises. But having a low bias (underfitting), could lead to high variance in the data.

Now checking which would be the optimal based on 50 leaves:

```
## [1] 22
```

After the pruning, the variables that are the most important in the tree are: “poutcome” “month” “day”

Interpret the information provided by the tree structure: We can see that the feature used in the root node is “poutcome”. The “month” and “day” feature is used in various places in the decision tree as a split.

3.4 4)

```
##      Ypredict4_test
##          no   yes
##   no  11872   107
##   yes  1371   214
```

```
## [1] 0.8910351
```

```
## [1] 0.224554
```

The F1_score is 0.224554

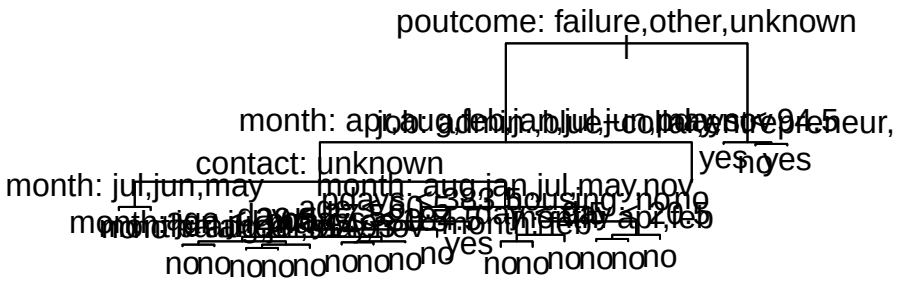


Figure 5: Tree structure. See Appendix 1 for better size

Comment whether the model has a good predictive power and which of the measures (accuracy or F1-score) should be preferred here: The model has a decent predictive power, more than 50% for both accuracy and F1-score so we can be relatively confident with this model but since its an imbalance class, we should look at the F1_score which is very low.
Since the class used to make the prediction is imbalanced, using the F1-score as a measure should be preferred.

3.5 5)

```
##      final
##          no   yes
##   no  11030   949
##   yes   771   814
```

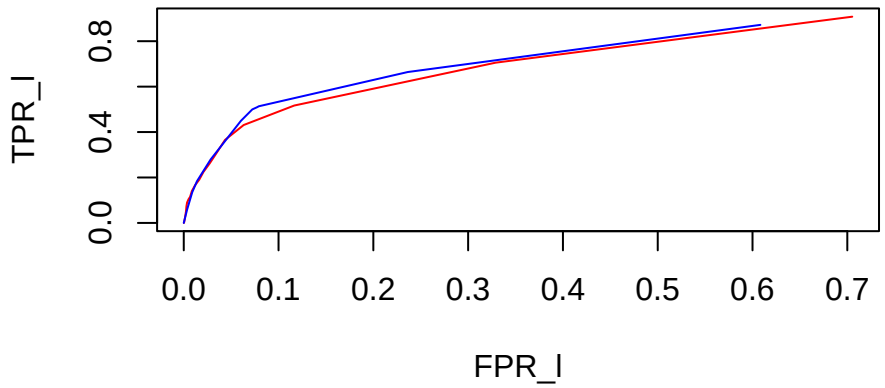
```
## [1] 0.8731937
```

```
## [1] 0.4862605
```

The F1_score is now 0.4862605.

Compare the results with the results from step 4 and discuss how the rates has changed and why: We can see that the accuracy has reduced. This is because as we have increased the threshold for classifying it as "no" is higher. So more will get classified as "yes". This is done to due to the imbalance in the dataset. But we can see that the F1score has increased which is good considering the imbalance in the data.

3.6 6)




```
##  [1] 0.907886435 0.705362776 0.516719243 0.430914826 0.365299685 0.303470032
##  [7] 0.261198738 0.224605678 0.190536278 0.165299685 0.140694006 0.114826498
## [13] 0.104100946 0.089589905 0.073817035 0.050473186 0.023343849 0.003154574
## [19] 0.000000000
```

```
##  [1] 0.87192429 0.66435331 0.51356467 0.51356467 0.50914826 0.49968454
##  [7] 0.44921136 0.27949527 0.18359621 0.13501577 0.13501577 0.13501577
## [13] 0.13501577 0.05804416 0.05804416 0.00000000 0.00000000 0.00000000
## [19] 0.00000000
```

```
##  [1] 7.052342e-01 3.289089e-01 1.165373e-01 6.319392e-02 4.349278e-02
##  [6] 3.389265e-02 2.721429e-02 2.078638e-02 1.636197e-02 1.185408e-02
## [11] 8.598380e-03 6.678354e-03 5.008765e-03 3.506136e-03 2.838300e-03
## [16] 2.337424e-03 1.335671e-03 4.173971e-04 8.347942e-05
```

```
##  [1] 0.608314550 0.236330245 0.079221972 0.079221972 0.077218466 0.072293180
##  [7] 0.060272143 0.028383004 0.013941064 0.008932298 0.008932298 0.008932298
## [13] 0.008932298 0.003506136 0.003506136 0.000000000 0.000000000 0.000000000
## [19] 0.000000000
```

From the plot, we can see that ROC curve for both the models are around the same but the tree model performs slightly better.

Yes, the precision-recall curve would be a better option due to the imbalance in the predictor class.

4 Assignment 3. Principal components and implicit regularization

The data file communities.csv contains the results of studies of the crime level in the united states based on various characteristics of the given location. The main variable that is studied is ViolentCrimesPerPop which represents the total number of violent crimes per 100K population. The meaning of other variables can be found at: https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime

4.1 1

Scale all variables except of ViolentCrimesPerPop and implement PCA by using function eigen(). Report how many components are needed to obtain at least 95% of variance in the data. What is the proportion of variation explained by each of the first two principal components?

We removed the state variable as its nominal and to scale it wouldn't make any sense, https://archive.ics.uci.edu/dataset/183/communities+and+crime also state that they doesn't count is as a predictive variable.

```
## To obtain 95% of the variance 34 components are needed.
```

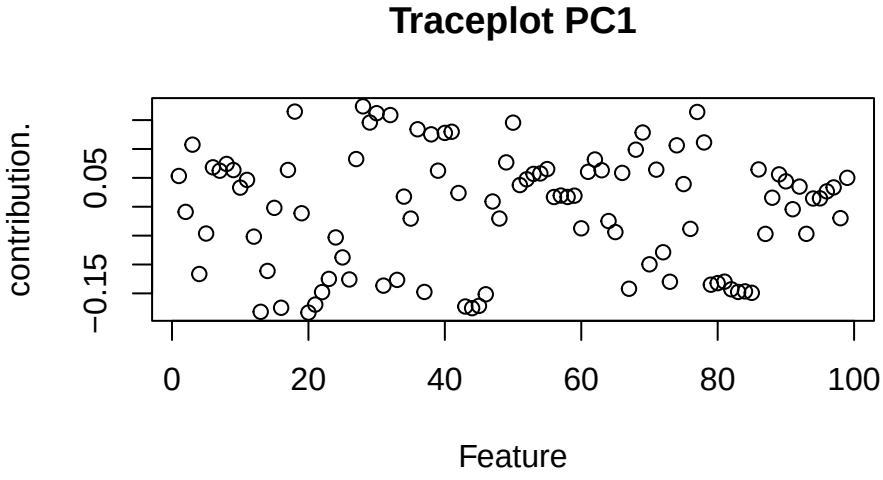
Table 3: The first two components

x
25.26876
16.97557

The first two components explain 25.27%(PC1) and 16.96%(PC2) of the variance, this is usually a bit low for the first components but our variables might have a low correlation between each other and therefore none of our components explain more of the variation.

4.2 2

Repeat PCA analysis by using princomp() function and make the trace plot of the first principle component. Do many features have a notable contribution to this component? Report which 5 features contribute mostly (by the absolute value) to the first principle component. Comment whether these features have anything in common and whether they may have a logical relationship to the crime level. Also provide a plot of the PC scores in the coordinates (PC1, PC2) in which the color of the points is given by ViolentCrimesPerPop. Analyse this plot (hint: use ggplot2 package).



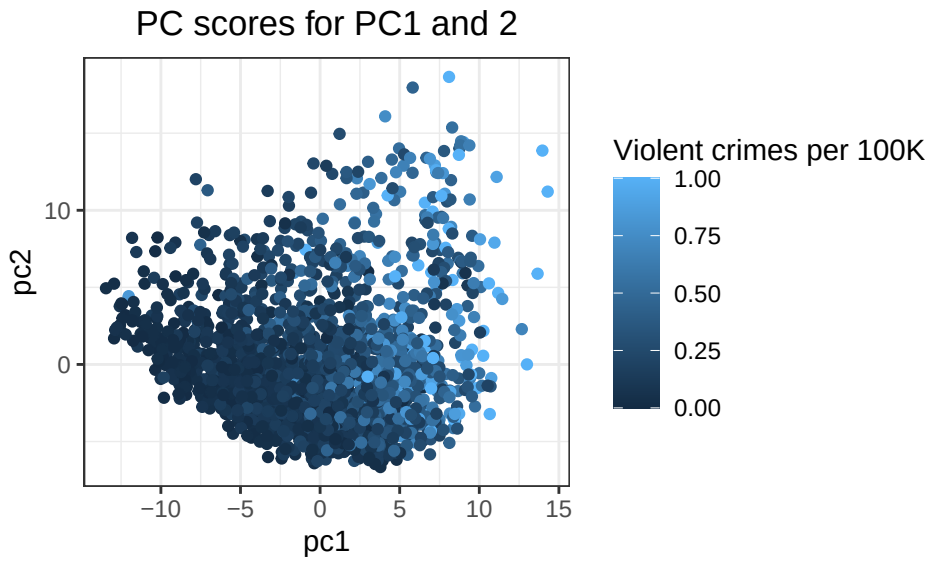
There are some features that contributes to the first component, we can can see that for each dot in the plot with values close to -0.15 or 0.15 on the y-axis are components that have a notable contribution.

	x
medFamInc	-0.1832453
medIncome	-0.1819115
PctKids2Par	-0.1755956
pctWInvInc	-0.1749060
PctPopUnderPov	0.1738039

medFamInc: median family income (differs from household income for non-family households) (numeric - decimal) medIncome: median household income (numeric - decimal) PctKids2Par: percentage of kids in family housing with two parents (numeric - decimal) pctWInvInc: percentage of households with investment / rent income in 1989 (numeric - decimal) PctPopUnderPov: percentage of people under the poverty level (numeric - decimal)

All of these 5 features have something to do with the amount of money the household have. 4 of the variables have a negative contribution which mean that a higher median income or higher percentage of kids living with 2 parents returns lower values for the Z_1. And the more percentage of people living under poverty returns higher values for z_1.

And this might be logical, because if theft and robbery might lead to violent crimes then we think that if there could be a correlation between percentage of people under poverty, the overall wealth of the households and ViolentCrimesPerPop.



Higher values of PC1 seems to have a higher higher violent crimes per 100k population as the color is lighter for higher values. Its harder to see any relationship between PC2 and violent crimes per 100 k population, as the color mostly seems to be correlated with PC1 and not PC2, so the variables that contribute the most to PC2 most likely doesn't correlate that strongly with violent crimes .

4.3 3

Split the original data into training and test (50/50) and scale both features and response appropriately, and estimate a linear regression model from training data in which ViolentCrimesPerPop is target and all other data columns are features. Compute training and test errors for these data and comment on the quality of model.

Train	Test
0.2773133	0.4249906

The model performs okay, its hard to say with just looking at one model and not comparing it against others but the test MSE is quite higher than the train MSE so the model probably overfits. Linear regression with 100 variables is probably not the best method for this problem, most parameters aren't significant and regularization is needed to reduce the overfit.

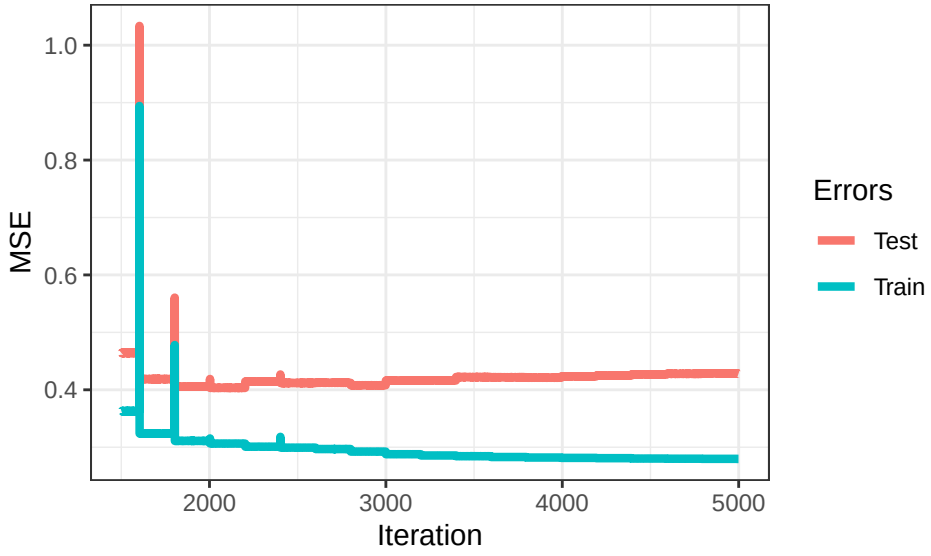
4.4 4

Implement a function that depends on parameter vector θ and represents the cost function for linear regression without intercept on the training data set. Afterwards, use BFGS method (optim() function without gradient specified) to optimize this cost with starting point $\theta^0 = 0$ and compute training and test errors for every iteration number. Present a plot showing dependence of both errors on the iteration number and comment which iteration number is optimal according to the early stopping criterion. Compute the training and test error in the optimal model, compare them with results in step 3 and make conclusions.

- a. Hint 1: don't store parameters from each iteration (otherwise it will take a lot of memory), instead compute and store test errors directly.
- b. Hint 2: discard some amount of initial iterations, like 500, in your plot to make the dependences visible.

```
# creating the function
linear_cost <- function(theta){
  # checking parameter vector theta
  stopifnot(is.vector(theta))
  # the predicted y with parameter vector theta
  y_hat <- as.matrix(trainS[,-ncol(trainS)]) %*% as.matrix(theta)

   # Saving outputs in environment
  .GlobalEnv$k <- .GlobalEnv$k+1
  if(.GlobalEnv$k > 500){ # not saving the first 1000 iterations
  .GlobalEnv$mse_train[[.GlobalEnv$k]] <- mean((trainS$ViolentCrimesPerPop - y_hat)^2)
  .GlobalEnv$mse_test[[.GlobalEnv$k]] <- mean((testS$ViolentCrimesPerPop - as.matrix(testS[,-ncol(testS)
  }
  # saving optimal parameters from the plot(added this after first run)
  if(k ==2160){
  .GlobalEnv$optimal <- theta
  }
  # cost function(MSE)
  return(mean((trainS$ViolentCrimesPerPop - y_hat)^2))

}

# starting point
theta <- c(rep(0,ncol(train)-2))

k=0
mse_train <- list()
mse_test <- list()
optimal <- c()

opt <- optim(theta, fn=linear_cost,gr= NULL, method = "BFGS")
```

```
## Optimal itiration for test MSE 1160
```



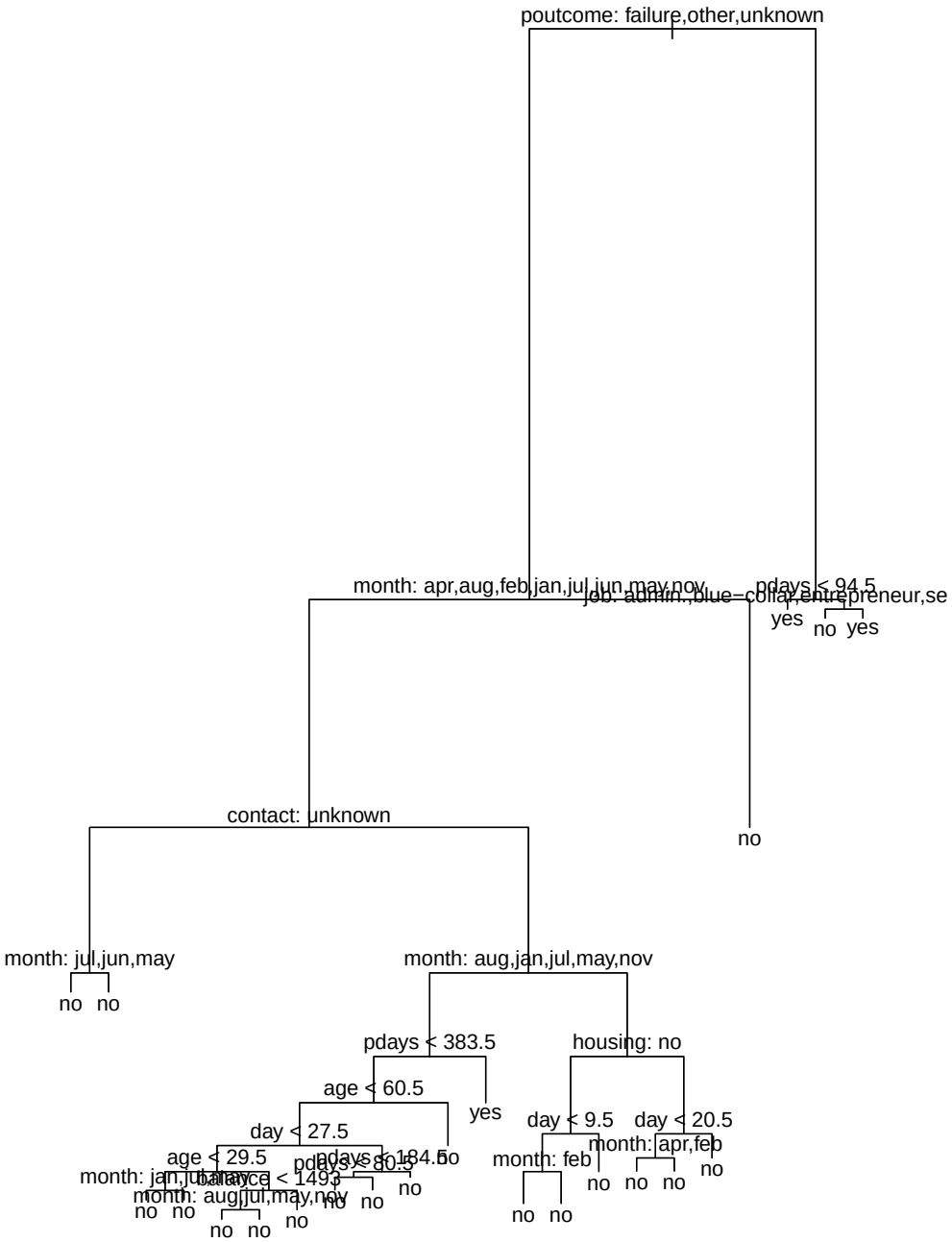
The optimal iteration according to early stopping should be around 2160 as the test error increases after that iteration, so the iterations could've stopped after some chosen amount of steps when the test MSE started to increase to save time.

Train	Test
0.306377	0.4031648

We now get a lower test MSE and a higher train MSE, and as we get lower for the Test data, the model can be considered better than the one in 3.3. Its expected to get a lower MSE for the test data as we choose the parameter vector after the MSE on the test data even though its optimized on the train data. We have reduced(regularized the model) some of the complexity which created the overfit from 3.3.

5 Appendix

5.0.1 Figures



5.0.2 Code

```
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning=FALSE, fig.width = 5, fig.height = 3, fig.a
set.seed(12345)
# packages
library(glmnet)
library(caret)
library(dplyr)
library(ggplot2)

df1 <- read.csv("tecator.csv")
rownames(df1) <- df1$Sample
df1 <- df1 %>% select(.,-Sample)
df1 <- read.csv("tecator.csv")
rownames(df1) <- df1$Sample
df1 <- df1 %>% select(.,-Sample)

# split data into train/test
set.seed(12345)
n <- nrow(df1)
id <- sample(1:n, floor(n*0.50))
train <- df1[id,]
test <- df1[-id,]
mod1 <- lm(Fat ~ .-Protein-Moisture, data = train) # fit linear regression
errors <- data.frame(mean(mod1$residuals^2), # training error
                         mean((test$Fat - predict(mod1, newdata = test))^2) # test error
                         )
colnames(errors) <- c("Training error", "Test error")
knitr::kable(errors, caption = "Error estimates on train and test")
qual <- data.frame(summary(mod1)[8], # quality of fit
                       summary(mod1)[9]) # adjusted quality of fit
colnames(qual) <- c("$R^2$", "$R^2_{adj}$")
knitr::kable(qual, caption = "Quality of Fit for linear regression")
scaler <- preProcess(train[,-c(101:103)]) # learn scaler based on training data
train1 <- predict(scaler, train) # scale train data
covariates <- train1[,1:100]
response <- train1[,101]

mod2 <- glmnet(as.matrix(covariates),
               response, alpha = 1, family = "gaussian") # alpha = 1 meaning LASSO

plot(mod2, xvar = "lambda", label = TRUE)
mod3 <- glmnet(as.matrix(covariates),
               response, alpha = 0, family = "gaussian") # alpha = 0 meaning Ridge

plot(mod3, xvar = "lambda", label = TRUE)
# compute LASSO model with cross-validation and default number of folds = 10
mod4 <- cv.glmnet(as.matrix(covariates),
```

```
                  response, alpha = 1, family = "gaussian", nfolds = 10)

plot(mod4)
test1 <- predict(scaler, test) # scaling test data with the scaler learned on training

pred4 <- predict(mod4, newx = as.matrix(test1[,1:100]), type = "response") # predict
plot(test$Fat, pred4)
library(tree)
library(caret)
data_1<-read.csv2("bank-full.csv",stringsAsFactors = TRUE)
data_2<-data_1[,-12]

n=dim(data_2)[1]
set.seed(12345) #Set seed first
id = sample(1:n, floor(n*0.4))
train = data_2[id,]

id1 = setdiff(1:n,id)
set.seed(12345) #Set seed second
id2 = sample(id1, floor(n*0.3))
val = data_2[id2,]

id3 = setdiff(id1,id2)
test = data_2[id3,]
model1<-tree(y ~ . , data = train)

model2<-tree(y~.,data = train,control = tree.control(nobs = 22605, minsize = 7000))

model3<-tree(y~.,data = train,control = tree.control(nobs = 22605,mindev = 0.0005))


#prediction for model1
Ypredict1_train<-predict(model1,newdata = train,type = "class") #miss rate: 0.1071
Ypredict1_val<-predict(model1,newdata = val,type = "class")  #miss rate: 0.1101

#confusionMatrix(Ypredict1_train,train$y)
#confusionMatrix(Ypredict1_val,val$y)

list(
  "Model1 train" = 1-as.data.frame(confusionMatrix(Ypredict1_train,train$y)[3])[1,1],
  "Model1 val" = 1-as.data.frame(confusionMatrix(Ypredict1_val,val$y)[3])[1,1]
)

#prediction for model2
Ypredict2_train<-predict(model2,newdata = train,type = "class")  #miss rate: 0.1071
Ypredict2_val<-predict(model2,newdata = val,type = "class")  #miss rate: 0.1101

#confusionMatrix(Ypredict2_train,train$y)
#confusionMatrix(Ypredict2_val,val$y)
```

```
list(
  "Model2 train" = 1-as.data.frame(confusionMatrix(Ypredict2_train,train$y)[3])[1,1],
  "Model2 val" = 1-as.data.frame(confusionMatrix(Ypredict2_val,val$y)[3])[1,1]
)

#prediction for model3
Ypredict3_train<-predict(model3,newdata = train,type = "class")  #miss rate: 0.0983
Ypredict3_val<-predict(model3,newdata = val,type = "class")  #miss rate: 0.1133

#confusionMatrix(Ypredict3_train,train$y)
#confusionMatrix(Ypredict3_val,val$y)

list(
  "Model3 train" = 1-as.data.frame(confusionMatrix(Ypredict3_train,train$y)[3])[1,1],
  "Model3 val" = 1-as.data.frame(confusionMatrix(Ypredict3_val,val$y)[3])[1,1]
)
trainScore=rep(0,100)
testScore=rep(0,100)
for(i in 2:100) {
  prunedTree=prune.tree(model3,best=i)
  pred=predict(prunedTree, newdata=val,
               type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:100, trainScore[2:100], type="b", col="red", ylim = c(5000,16000))

points(2:100, testScore[2:100], type="b", col="blue" )
#which(trainScore == min(trainScore[2:50]))
which(testScore == min(testScore[2:50]))


#Based on the training data, the optimal leaves is 22 .
model4=prune.tree(model3, best=22)

plot(model4)
text(model4,pretty = 0)
Ypredict4_test<-predict(model4, newdata = test,type = "class")
#confusionMatrix(test$y,Ypredict4_test)
t<-table(test$y,Ypredict4_test)
accur<-sum(diag(table(test$y,Ypredict4_test)))/sum(table(test$y,Ypredict4_test))
t
accur

tp<-t[2,2]
fp<-t[1,2]
fn<-t[2,1]
```

```r
#F1- Score:
p<-tp/(tp+fp)
r<-tp/(tp+fn)
F1_score<- (2*p*r)/(p+r)
F1_score
Ypredict4_test_2<-predict(model4, newdata = test,type = "vector")

final<-c()
for (i in 1:nrow(Ypredict4_test_2)){
  if(Ypredict4_test_2[i,1]/Ypredict4_test_2[i,2] > 5){
    final[i]<-"no"
  }else{
    final[i]<-"yes"
  }
}
final<-as.factor(final)
#summary(final)


#confusionMatrix(test$y,final)

t<-table(test$y,final)
accur<-sum(diag(table(test$y,final)))/sum(table(test$y,final))
t
accur

tp<-t[2,2]
fp<-t[1,2]
fn<-t[2,1]

#F1- Score:
p<-tp/(tp+fp)
r<-tp/(tp+fn)
F1_score<- (2*p*r)/(p+r)
F1_score
thres<-seq(0.05,0.95,0.05)

model5<-glm(y~.,data = train, family = "binomial")
Ypredict5<-predict(model5,newdata = test, type = "response")

TPR_l<-c()
TPR_t<-c()

FPR_l<-c()
FPR_t<-c()

j<-1

for(i in thres){
```

```
  #for logistic
  pred<-ifelse(Ypredict5>i,"yes","no")
  pred<-factor(pred,levels = c("no","yes"))
  a<-confusionMatrix(test$y,pred)


  TPR_l[j]<- a$table[4]/(a$table[2]+a$table[4])
  FPR_l[j]<- a$table[3]/(a$table[1]+a$table[3])

  #for decision tree
  final<-c()
  for (k in 1:nrow(Ypredict4_test_2)){
    if(Ypredict4_test_2[k,2] > i){
      final[k]<-"yes"
    }else{
      final[k]<-"no"
    }
  }
  final<-factor(final,levels = c("no","yes"))
  b<-confusionMatrix(test$y,final)

  TPR_t[j]<- b$table[4]/(b$table[2]+b$table[4])
  FPR_t[j]<- b$table[3]/(b$table[1]+b$table[3])


  j<-j+1
}


plot(FPR_l,TPR_l,type = "l",col = "red")
points(FPR_t,TPR_t,type = "l", col = "blue")


#Values of TPR and FPR for both the models:
TPR_l
TPR_t
FPR_l
FPR_t



# reading the data
com <- read.csv('communities.csv')


# scaling all varaibles except for state and ViolentCrimesPerPop
# removing state as the link said its not counted as predictive(nominal so no order and scale would be w
com_scale <- scale(com[,-c(1,ncol(com))])
# cov matrix
```

```r
S <- (1/length(com_scale)) * t(com_scale) %*% com_scale

# implementation of pca
pca <- eigen(S)


# calculating the variance explained
lambda <- (pca$values/sum(pca$values)) *100

lamb_95 <- c()

# a loop to see when over 95% of the variance is explained
for (i in 1:length(lambda)) {

    lamb_95[i] <-  lambda[i]
    if(sum(lamb_95) >= 95){
      break
    }
}

# printing the results
cat('To obtain 95% of the variance',length(lamb_95),'components are needed.')

knitr::kable(lamb_95[1:2],caption = "The first two components")
# pca with princomp
pc <- princomp(com_scale)


lambda_2 <- (pc$sdev^2)/ sum(pc$sdev^2) *100
contribution. <- pc$loadings[,'Comp.1']

plot(contribution.,main='Traceplot PC1', xlab='Feature')


contribution. <- contribution.[order(abs(contribution.), decreasing=TRUE)]

knitr::kable(contribution.[1:5],main='The 5 features that contributes mostly')
pc1 <- pc$scores[,'Comp.1']
pc2 <-  pc$scores[,'Comp.2']

# creating a df with the scores
df_score <- data.frame(pc1,pc1,'col'=com$ViolentCrimesPerPop)



# plotting the scores with ViolentCrimesPerPop as color
ggplot(df_score,aes(x=pc1,y=pc2,color=col))+ geom_point() +
  scale_color_continuous('Violent crimes per 100K')+ theme_bw() + ggtitle('PC scores for PC1 and 2')+
  theme(plot.title=element_text(hjust=0.5))
```

```r
# sampling test and train data
n <- nrow(com)
set.seed(12345)
id <- sample(1:n,floor(n*0.5))



train <- com[id,] # train
test <- com[-id,] # test

# scaling the data with mean and sd from train
 # removing state as the link said its not counted as predictive(nominal so no order and scale would be
scaler <- preProcess(train[,-1])
trainS <- predict(scaler,train[,-1])
testS <- predict(scaler,test[-1])

# creating the model
mod <- lm(ViolentCrimesPerPop~.,trainS)

pred <- predict(mod, trainS)
MSE_train <- mean((trainS$ViolentCrimesPerPop-pred)^2)# MSE for train

pred <- predict(mod, testS) # Predicting the test data

MSE_test <- mean((testS$ViolentCrimesPerPop-pred)^2) # MSE for test
# print
knitr::kable(data.frame('Train'=MSE_train , 'Test'=MSE_test))

# creating the function
linear_cost <- function(theta){
  # checking parameter vector theta
  stopifnot(is.vector(theta))
  # the predicted y with parameter vector theta
  y_hat <- as.matrix(trainS[,-ncol(trainS)]) %*% as.matrix(theta)

   # Saving outputs in environment
  .GlobalEnv$k <- .GlobalEnv$k+1
  if(.GlobalEnv$k > 500){ # not saving the first 1000 iterations
  .GlobalEnv$mse_train[[.GlobalEnv$k]] <- mean((trainS$ViolentCrimesPerPop - y_hat)^2)
  .GlobalEnv$mse_test[[.GlobalEnv$k]] <- mean((testS$ViolentCrimesPerPop - as.matrix(testS[,-ncol(testS)
  }
  # saving optimal parameters from the plot(added this after first run)
  if(k ==2160){
  .GlobalEnv$optimal <- theta
  }
  # cost function(MSE)
  return(mean((trainS$ViolentCrimesPerPop - y_hat)^2))
```

```
}

# starting point
theta <- c(rep(0,ncol(train)-2))

k=0
mse_train <- list()
mse_test <- list()
optimal <- c()

opt <- optim(theta, fn=linear_cost,gr= NULL, method = "BFGS")


# creating a df of my data
df <- data.frame("MSE" = c(unlist(mse_train)[-c(1:1000)],unlist(mse_test)[-c(1:1000)]), 'Errors'= as.fac
inx <- df$Iteration < 5000
cat('Optimal itiration for test MSE',which.min(df$MSE[df$Errors=='Test']) + 500) #finding min
# plotting the errors
ggplot(df[inx,], aes(x=Iteration, y=MSE, color=Errors)) + geom_line(linewidth=1.5)+ theme_bw()


# computing train and test for "optimal" model
train_mse <- mean((trainS$ViolentCrimesPerPop - (as.matrix(trainS[,-ncol(trainS)]) %*% optimal))^2)
test_mse <- mean((testS$ViolentCrimesPerPop - (as.matrix(testS[,-ncol(testS)]) %*% optimal))^2)

knitr::kable(data.frame("Train"=train_mse,"Test"=test_mse))

plot(model4)
text(model4,pretty = 0)
```