Lab 2 Block 1

Pontus Olsson, Max Goldbeck-Löwe, Paripurna Bawonoputro

2025-12-06

Assignment 1 Explicit regularization

This assignment contains all the answers for assignment 1, which contains exercises on explicit regularizations.

1. Creating a model without any shrinkage functions

To start this assignment we assume that the level of fat is dependent on 100 channel spectrum of of absorbance records.

We can express it as a linear regression model like:

$$Y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + ... + \beta_{100} X_{100} + \epsilon_i$$

While the output of this model is at least 100 coefficients, which can be hard to show all at once, we

Table 1: Output of linear model: B0, B1, B2, B3, . . . , B100

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-1.10	4.39	-0.25	0.81
Channel1	22851.13	18988.96	1.20	0.27
Channel2	-32753.78	35451.20	-0.92	0.39
Channel3	-4293.85	48676.28	-0.09	0.93
Channel100	12079.82	26395.94	0.46	0.66

If we look at the output in table 1, we see there are many insignificant coefficients. While $t_{Statistic}$ are not a good measurement of parameter fitness because of confounding effects, it indicates that we have to many insignificant parameters in our model.

Another way to show this is by having a general overlook and comparing model output on testdata.

Table 2: Linear model overview

# parameters	Train MSE	Train Rr	Test MSE	Test R^2	Null MSE (test)
101	0.087	0.9995	339.1794	3.3855	157.1418

From table 2, we can see that the model is extremely overfitted. There are 100 parameters and there are 108 observations. There are around the same amount of parameters as observations. The descipency in the R^2 measurement between training data and test data is a good showcase of this. The R^2 value for the training data is almost 1, while the R^2 for the testdata is higher than 1, which means that the estimated sum of squares ($ESS = \sum_{i=1}^{n}(\hat{y} - \bar{y})^2$) is higher than the total variation in the data $TSS = \sum_{i=1}^{n}(y - \bar{y})^2$. This is, in a nonacademic of expressing it, terrible since it means that a model without *any* parameters has better accuracy than the model. We can see this comparing the Null MSE with the test MSE.

2. Shrinkage with LASSO

There are many ways to reduce the overfitting in this linear model. One way is by penalizing complexity by introducing a cost function within the model.

From MLFC page 110: "The penalty cost $||\theta||_1$ is added to the cost function. [...] The regularized cost function for linear regression with squared error loss then becomes:"

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} ||\mathbf{X}\theta - \mathbf{y}||_2^2 + \lambda ||\theta||_2^2$$

or from James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An introduction to statistical learning: With applications in R (2nd ed.) page 241

$$\arg\min f(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \, , \lambda \geq 0$$

Depending on what literature you're reading, basically they describe the same thing though. The first definition just does it in a more optimized way.

3. Plotting how LASSA coefficients depends on lambda

By plotting the coefficients from different iterations of λ, we can see how coefficients depend on the log penalty factor.

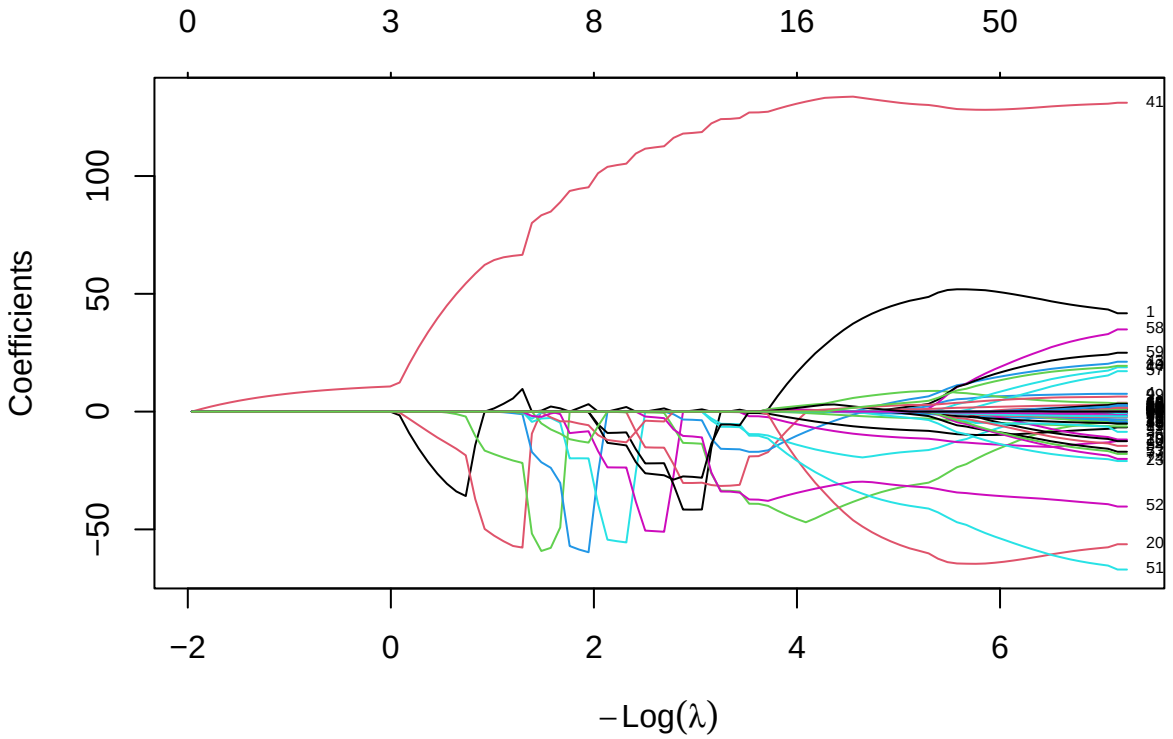


Figure 1: Coefficients and lambda, LASSO

The Y axis shows the changes in coefficient, the lower X-axis shows the $-\log(\lambda)$ penalty factor and the upper X-axis shows the number of parameters in the model.

If we look at the plot, we can see that if we choose a value between $-\log(\lambda) = 0$ and approximately $-\log(\lambda) = 0.5$ it account for 3 variables. A more precise way to show this is by looking at the object:

Table 3: Coefficients through lambda iterations

	-0.006	0.087	0.18	0.273	0.366	0.924
Channel7	0.00000	-1.5750493	-9.064868	-15.58934	-21.315207	0.00000
Channel8	0.00000	-0.4345616	-3.060518	-5.74811	-8.414918	-49.82120
Channel9	0.00000	0.0000000	0.000000	0.00000	0.000000	-16.55091
Channel41	10.66752	12.3440772	20.140250	27.24146	33.712118	62.27446

4. Plotting how RIDGE coefficients depends on lambda

We do the same as figure 1, but on a Ridge regression.

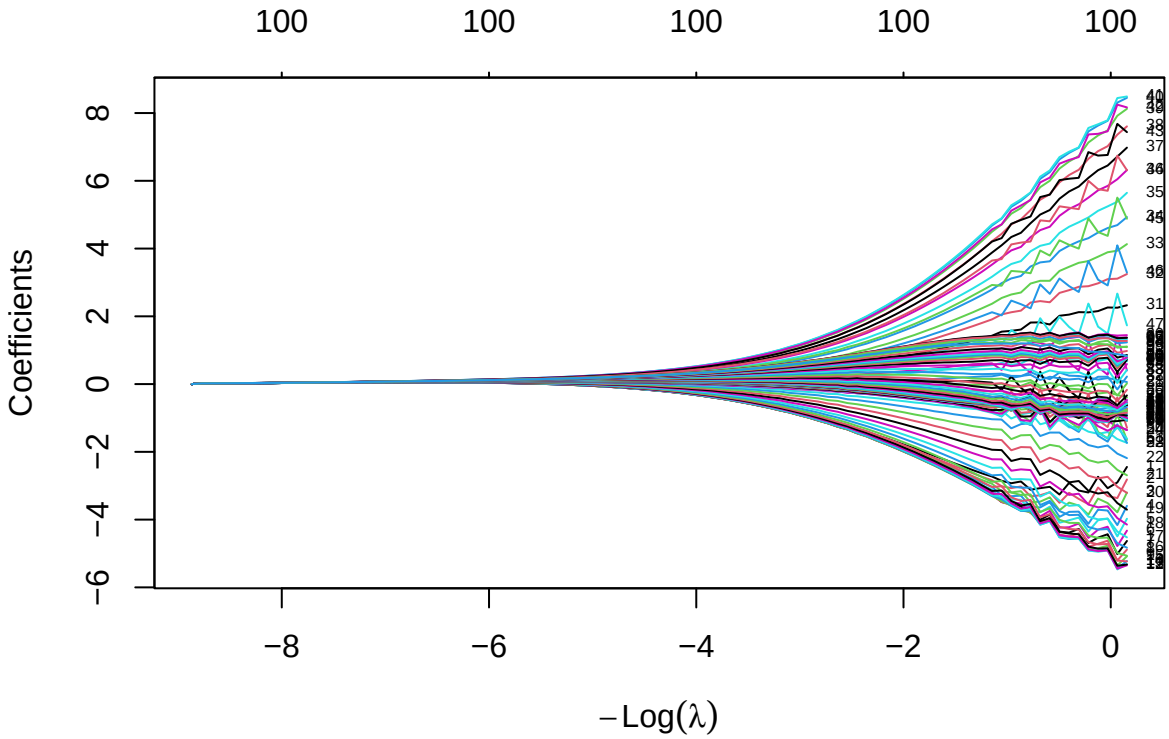


Figure 2: Coefficients and lambda, Ridge

The plots differ because a squared loss penalizes deviations less aggressively near zero, causing parameters to shrink more slowly compared to an absolute loss.

MFLC page 110: "As for L^2 regularization, the regularization parameter λ has to be chosen by the user and has a similar meaning: $\lambda = 0$ gives the ordinary least squares solution and $\lambda \to \infty$ gives $\hat{\theta} = 0$. Between these extremes, however, L^1 and L^2 tend to give different solution. Whereas L^2 regularization pushes all parameters towards small values (but not necessarily exactly zero), L^1 tends to favor so-called sparse solutions, where only a few of the parameters are non-zero, and the rest are exactly zero. Thus L^1 regularization can effectively"switch off" some inputs (by setting the corresponding parameter $theta_k$ to zero), and it can therefore be used as an input (or feature) selection method."

5. Cross-validation to choose lambda

By using cross-validation through K-fold, one can find the optimal λ. K-fold is built into the function *cv.glmnet*. The default argument for the number of folds are 10. The result of the cross-validation is shown in figure 3.

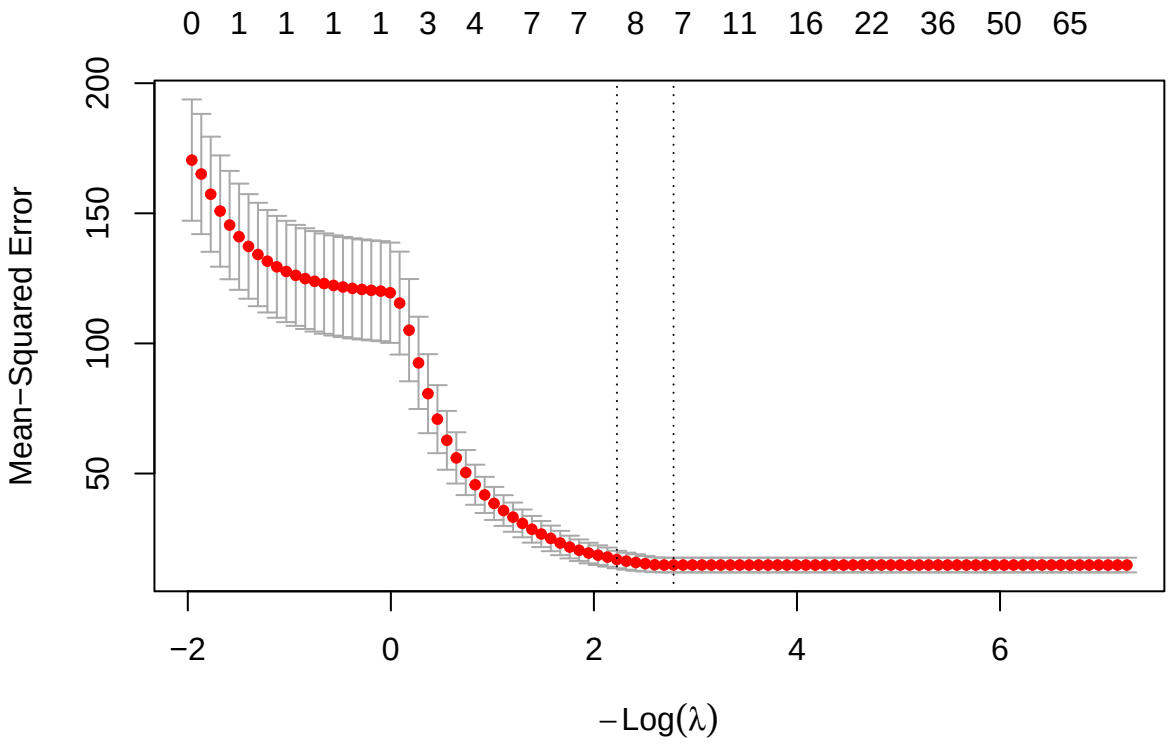


Figure 3: Best lambda from k−fold validation

Figure 3 shows that the lambda with the lowest MSE is when $\lambda = 0.0617565$ or $-\log(\lambda) = 2.7845558$. We do however see that most penalties after 2 do not significantly differ from the actual optimal value.

Table 4 shows the coefficients from the model with the lowest MSE in the K-fold cross-validation.

Table 4: Model coefficients when λ = -2.784

	Coefficient
(Intercept)	30.075298
Channel12	-27.085632
Channel13	-30.980052
Channel14	-22.359741
Channel15	-6.757625
Channel16	-1.131233
Channel41	116.214495
Channel52	-7.574339
Channel53	-28.886522

As we commented on in figure 3. The MSE stands still after $-\log(\lambda) \approx 2$, the interval for MSE still catches most of the values after. We can hence assume that $\log(\lambda) = 4$ do not significantly different from the optimal λ.

This is even shown by comparing "the optimal" λ to $\log(\lambda) = 4$, which is done in table 5.

	Cross validated λ	$\log(\lambda = -4)$
MSE	16.98	13.28
R^2	0.61	0.67

This shows that, despite $\lambda = -2.784$ being the value with the lowest MSE in the cross-validation, it does not necessarily mean that the same applies to the test data.

We can however conclude that we have significantly improved the model compared to the one shown in exercise 1. The prediction improvement has gone from 0 on actual data to ≈ 0.61. Evaluating model performance is always relative the goals that the model is trying to perform and previous models, by comparing the prediction values with its actual value we can however get a hint what improvements we could do in our model.

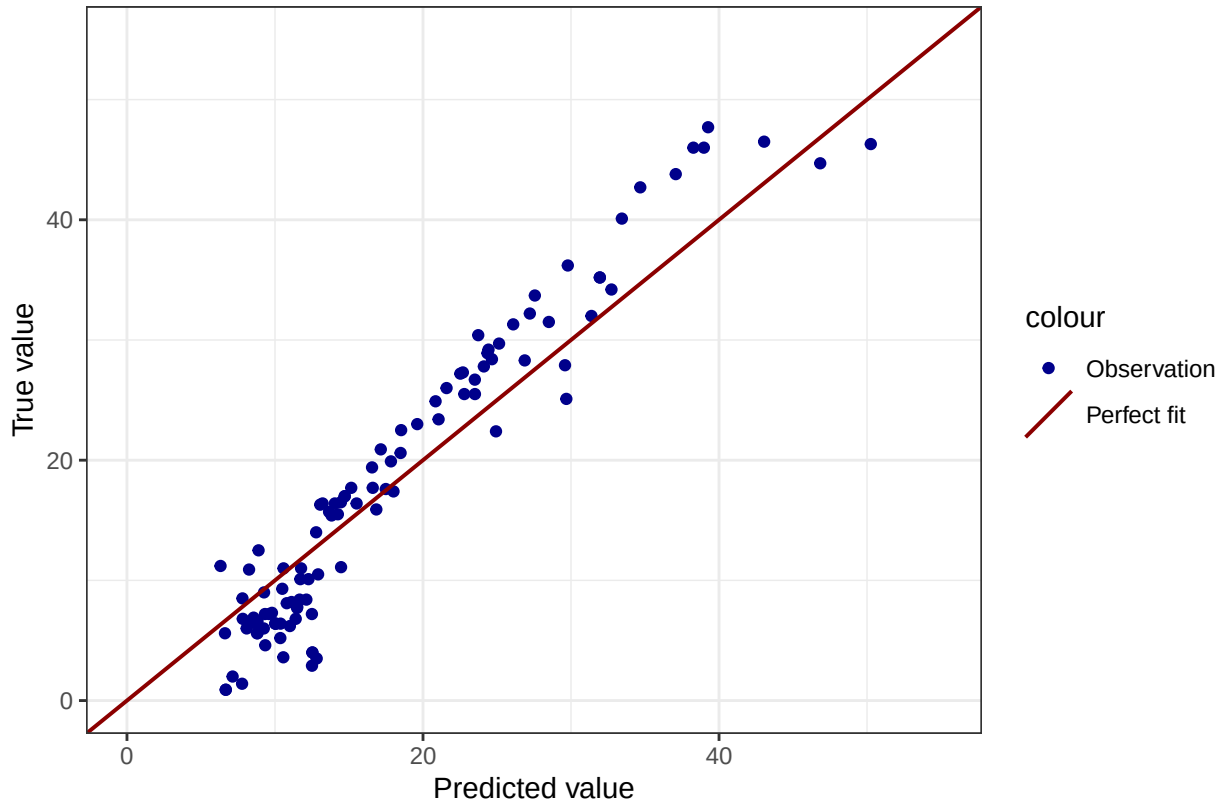


Figure 4: Comparison between predicted value and true value for test data

Figure 4 shows that there is at least a somewhat linear correlation between the predicted value and it's actual shown value. This is a huge improvement compared to the model that was shown in exercise 1. However, the predicted values have a hard time when it comes to lower true values, which shows with its higher variance, the model also tends to underestimate several true values, for example, values around and higher than 45 can be. All of this could indicate bias in the model.

The model works as a draft, but needs some improvements if it tries to predict the levels of fat with high accuracy.

Assignment 2. Decision Trees and Logistic Regression for Bank Marketing

1. Data Preperation

The data was divided into training (40%), validation (30%) and test (30%) sets and the variable "duration" was removed.

2. Decision Trees

Three different decision trees were fitted to the training data, out of which one had default settings, one had the smallest allowed node size equal to 7000 and the last had the minimum deviance equal to 0.0005. Below are the misclassification rates, rounded to four decimals, for both training and validation data for the three trees.

Table 6: Misclassification Rates on Training and Validation Data

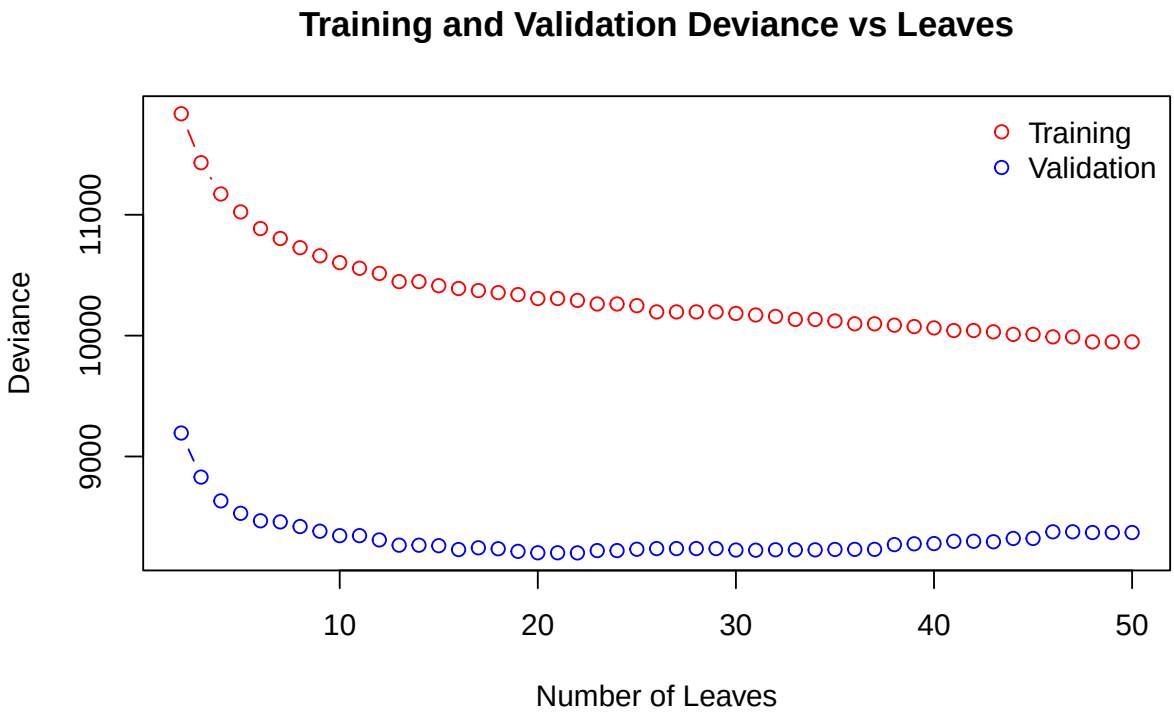
Trees	Training	Validation
Default Tree	0.1048	0.1093
Node Tree	0.1048	0.1093
Deviance Tree	0.0936	0.1118

While the tree with a minimum deviance fits the training data best out of the three, it performs slightly worse on the validation data compared to the other two, suggesting possible overfitting. Based on this, the default tree or the node tree, having identical misclassification rates, seem better choices.

When the minimum node size was set to 7000, it forced each terminal node to have more observations, reducing the number of splits. Hence, the tree for which this was done became slightly smaller and simpler than the default tree, with 5 terminal nodes as opposed to 6. When lowering the minimum deviance to 0.0005, however, more splits were allowed, resulting in a considerably larger and more complex tree with 122 nodes.

3. Optimal Tree Depth

Next, the optimal tree depth for the large (minimum deviance equal to 0.0005) tree was to be found. The development of deviance over up to 50 leaves was studied for both the training and validation sets, and the following graph was obtained.



As seen in the figure, the deviance for the training data continuously decreased as the number of leaves increased, though improvements became marginal at higher leaf counts. This indicates that the tree fit the training data better as it grew more complex. For the validation data, however, the deviance initially decreased, after which it leveled and even started to increase slightly. This demonstrates a bias-variance tradeoff, where more leaves at first resulted in a lower bias and thereby better fit, but as the tree became increasingly larger and more complex, while capturing even more of the training data, variance increased. The model started to overfit the training data and lost generalization ability, causing the increase in validation deviance.

The optimal number of leaves would be found where the validation deviance was at its lowest, approximately somewhere between 17 and 23, from reviewing the graph. To confirm and get an exact value, the minimum of the curve was evaluated and proved to occur when the number of leaves was equal to 22. The tree was then pruned to this optimal depth, and which variables were most important in decision making was examined. The variables "poutcome" (outcome of the previous marketing campaign) and "month" (last contact month of year) were the two most impactful, suggesting customers previous attitude towards bank term deposits, as well as in which month they were contacted, greatly affected their decision.

4. Confusion Matrix, Accuracy and F1 score

The confusion matrix for the test data, seen below, was estimated using the optimal tree obtained previously.

Table 7: Confusion Matrix 1

	Predicted: no	Predicted: yes
Observed: no	11872	107
Observed: yes	1371	214

The accuracy and F1 score were also calculated based on the confusion matrix. Accuracy was computed as

the proportion of correctly classified observations:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total observations}}$$

where TP and TN represent true positives and true negatives. Precision, recall, and F1 were derived as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Where FP and FN represent false positives and false negatives.

The model achieved an accuracy of **0.89** and an F1 score of **0.22**. While the accuracy appears high, this is largely due to the strong class imbalance in the dataset, as there were 39,922 observations labeled as "no" compared to only 5,289 labeled as "yes". In such cases, accuracy can be misleading because correctly predicting the majority class dominates the metric. The low F1 score reflects poor performance on the minority class ("yes"), as F1 combines precision and recall and is more sensitive to imbalanced data.

5. Loss Matrix

The test data was then classified again using the optimal tree, but this time with the addition of the loss matrix shown below.

Table 8: Loss Matrix

	Predicted: no	Predicted: yes
Observed: no	0	1
Observed: yes	5	0

To apply the loss matrix, the expected loss (EL) for each class was computed as:

$$\text{EL}(\text{predict yes}) = P(\text{no}) \cdot 1 + P(\text{yes}) \cdot 0, \quad \text{EL}(\text{predict no}) = P(\text{yes}) \cdot 5 + P(\text{no}) \cdot 0$$

The predicted class was chosen as the one with the lower expected loss. A new confusion matrix was then obtained.

Table 9: Confusion Matrix 2

	Predicted: no	Predicted: yes
Observed: no	11030	949
Observed: yes	771	814

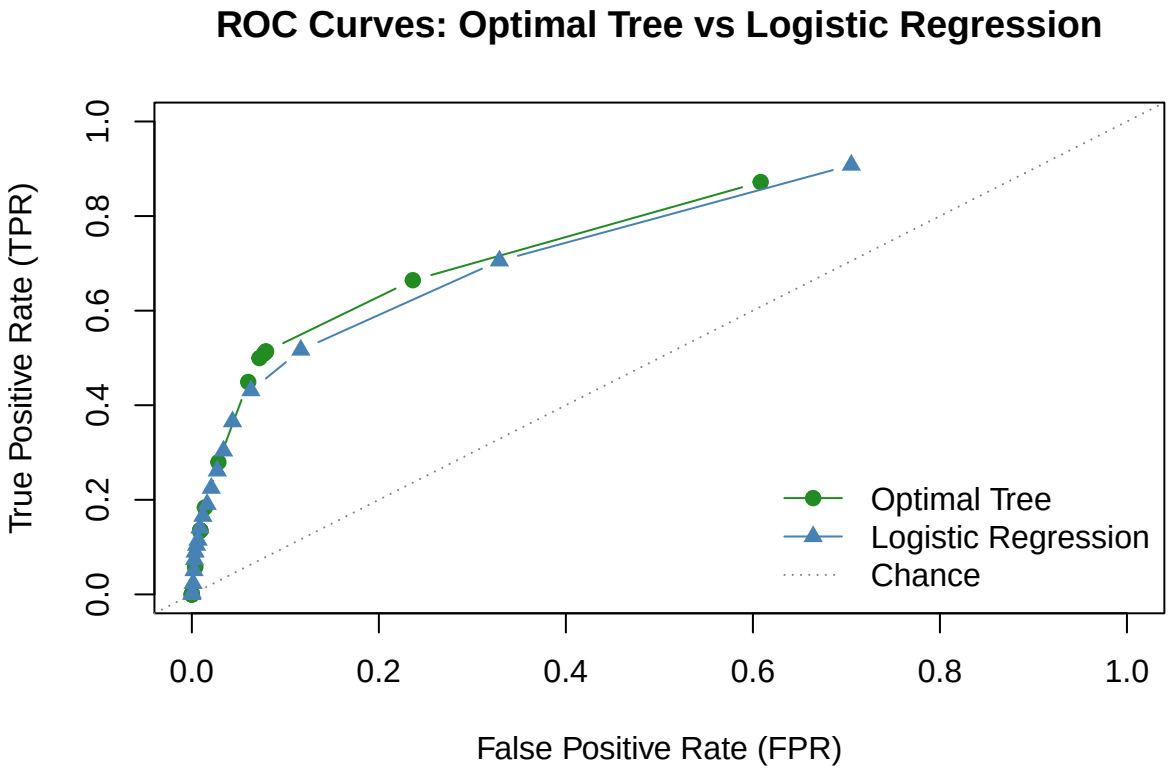
The confusion matrix shows a considerable increase in how much the model predicted "yes", which is explained by the relatively much larger penalization of misclassifying a "yes" as opposed to misclassifying a "no", imposed by the loss matrix. This resulted in a small decrease of the model accuracy, now being **0.87** (previously 0.89), but a quite large improvement of the F1 score to **0.49** (previously 0.22). While the model performed slightly less accurate predictions overall, it was significantly better when it came to the minority class ("yes").

6. ROC Curves

A logistic regression model was implemented on the training data, after which both that model and the optimal tree were used to classify the test data by the following principle:

$$\hat{Y} = \begin{cases} \text{yes}, & \text{if } p(Y = \text{'yes'} \mid X) > \pi \\ \text{no}, & \text{otherwise} \end{cases} \quad \text{where } \pi \in \{0.05, 0.10, 0.15, \ldots, 0.95\}$$

The TPR and FPR values where then computed for both models and the corresponding ROC curves are shown in the following plot.



The ROC curves for the optimal decision tree and logistic regression model both show performance well above the random baseline (represented by the dashed diagonal line). The optimal tree performed slightly better overall, as its curve lies marginally above the logistic regression, meaning it achieved higher true positive rates for the same false positive rates. However, because the dataset is highly imbalanced (many "no" and few "yes"), ROC curves can be misleading, as the false positive rate remains small even when the absolute number of false positives is large. In such cases, a precision–recall curve could be more informative because it focuses on the positive class, balancing precision (how many predicted "yes" are correct") and recall (how many of actual "yes" were captured), which is critical for campaign targeting. Essentially, you would want most of the people who you predicted would answer "yes" to actually do so, as contacting those who will not costs time and money, while simultaneously ensuring not to miss out on potential customers.

Assignment 3 Principal components and implicit regularization

First we have to load the data from communities.csv

Task 1

In this task, we're not asked to divide the data to train/test.

First, we need to separate the features and the target Then we scale the features Implement PCA using eigen() and computing variation explained

Implement PCA using eigen() and computing variation explained

Table 10: Variance Explained by Principal Components (1–40)

PC_1to20	VarExp_1to20	CumVar_1to20	PC_21to40	VarExp_21to40	CumVar_21to40
1	0.2502	0.2502	21	0.0062	0.8927
2	0.1694	0.4195	22	0.0057	0.8984
3	0.0930	0.5125	23	0.0054	0.9038
4	0.0756	0.5881	24	0.0052	0.9090
5	0.0566	0.6447	25	0.0050	0.9140
6	0.0424	0.6871	26	0.0048	0.9188
7	0.0323	0.7194	27	0.0047	0.9235
8	0.0297	0.7491	28	0.0045	0.9280
9	0.0207	0.7697	29	0.0043	0.9323
10	0.0162	0.7859	30	0.0039	0.9362
11	0.0157	0.8017	31	0.0037	0.9398
12	0.0147	0.8164	32	0.0035	0.9434
13	0.0141	0.8305	33	0.0034	0.9467
14	0.0103	0.8408	34	0.0031	0.9498
15	0.0093	0.8501	35	0.0029	0.9527
16	0.0089	0.8590	36	0.0026	0.9553
17	0.0075	0.8665	37	0.0026	0.9579
18	0.0071	0.8736	38	0.0025	0.9603
19	0.0065	0.8801	39	0.0024	0.9627
20	0.0064	0.8864	40	0.0022	0.9649

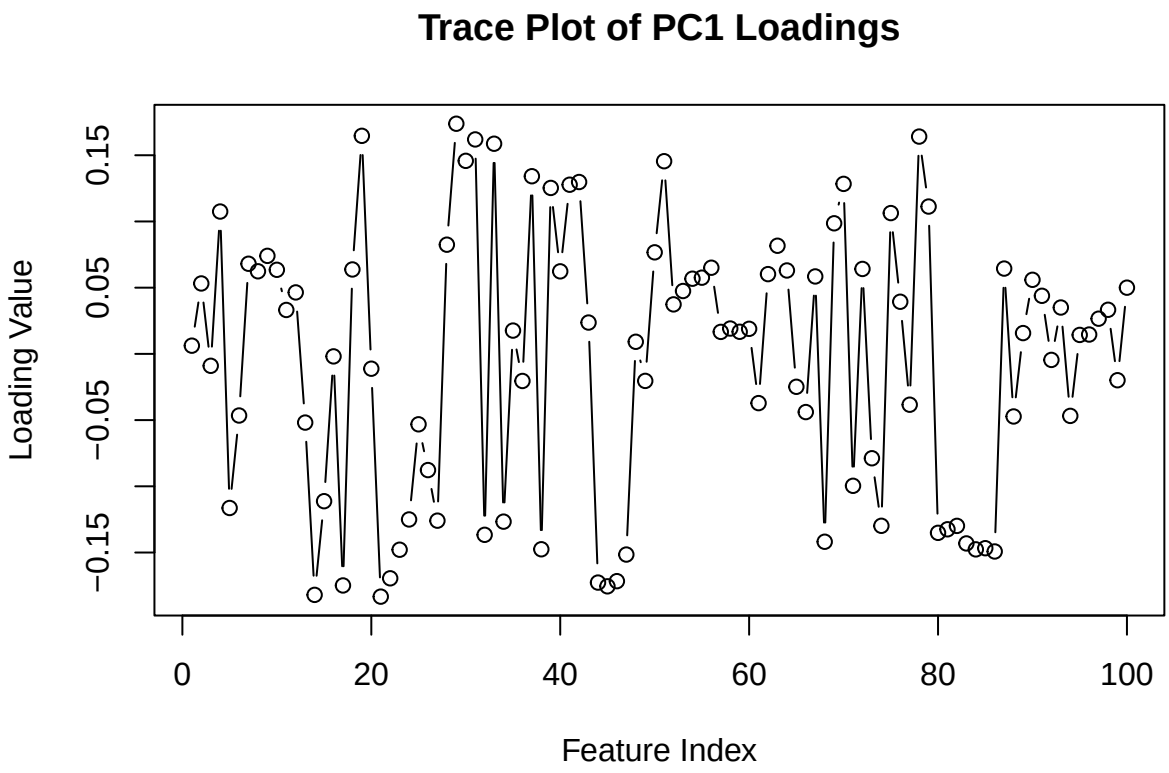
Based on Table 1, components needed to obtain at least 95% of variance in the data is

```
## [1] 35
```

The proportion of variation explained by PC1 and PC2 based on Table 1 are

```
## [1] 0.2501699 0.1693597
```

Task 2



Many loadings are close to 0, showing several variables do not contribute strongly to PC1.

PC1 is influenced by many moderately strong variables rather than a few dominant ones.

Both positive and negative contributions appear, means some features are positively associated with the PC1 direction while other features are negatively associated.

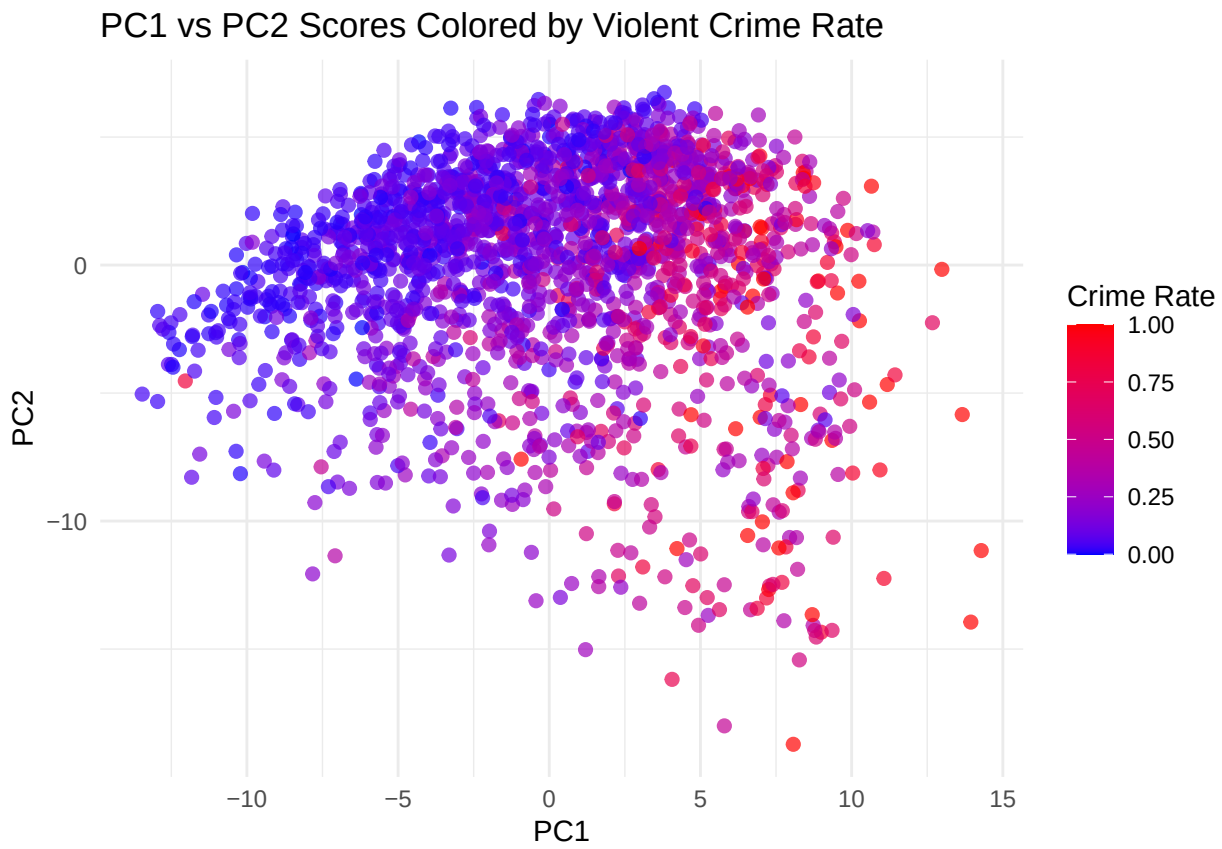
The five variables with the highest loadings on PC1 are:

```
## [1] "medFamInc"      "medIncome"      "PctKids2Par"    "pctWInvInc"
## [5] "PctPopUnderPov"
```

```
##      medFamInc      medIncome    PctKids2Par     pctWInvInc PctPopUnderPov
##     -0.1833080     -0.1819830     -0.1755423     -0.1748683      0.1737978
```

medFamInc (Median Family Income) negative
medIncome (Median Household Income) negative
PctKids2Par (Percentage of kids in family housing with two parents) negative
pctWInvInc (Percentage of households with investment / rent income in 1989) negative
PctPopUnderPov (Percentage of people under the poverty level) positive

PC1 is primarily driven by income variables and family structure, with high positive loading from poverty level. Communities with higher income and more two-parent households load negatively on PC1, while poorer communities load positively. In other words, place where the socio-economic status higher are less likely to have high crime records and vice versa.



The PC1–PC2 score plot shows that the violent crime rate increases along PC1. Communities on the right side of PC1 have higher crime and correspond to high-poverty, low-income areas identified in the loading analysis. Low-crime communities appear on the left side, corresponding to higher-income and more stable neighborhoods. PC2 shows less association with crime, meaning PC1 is the primary component capturing the socioeconomic factors driving crime variation

Task 3

Split data into training and test. Then separate the features and the target.

Then we have to apply scale to the feature and target. We should only use the parameter of train data to scale both train and test data. The actual code can be found in the appendix.

Then we fit the model and use it for prediction

To evaluate the fitted model, we compute the **mean squared error (MSE)** and the **coefficient of determination** R^2 for both the training and test sets.

The MSE is defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

which measures the average squared difference between the observed target values and the model predictions. Lower MSE indicates better predictive accuracy.

The R^2 score is computed as

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where $\bar{y}$ is the mean of the true target values.
An R^2 close to 1 indicates a strong model fit, while a **negative** R^2 means the model performs **worse than simply predicting the mean** of the target.

Table 11: Model Performance on Training and Test Sets

Dataset	MSE	R2
Train	0.2752	0.7245
Test	1.8139	-0.6155

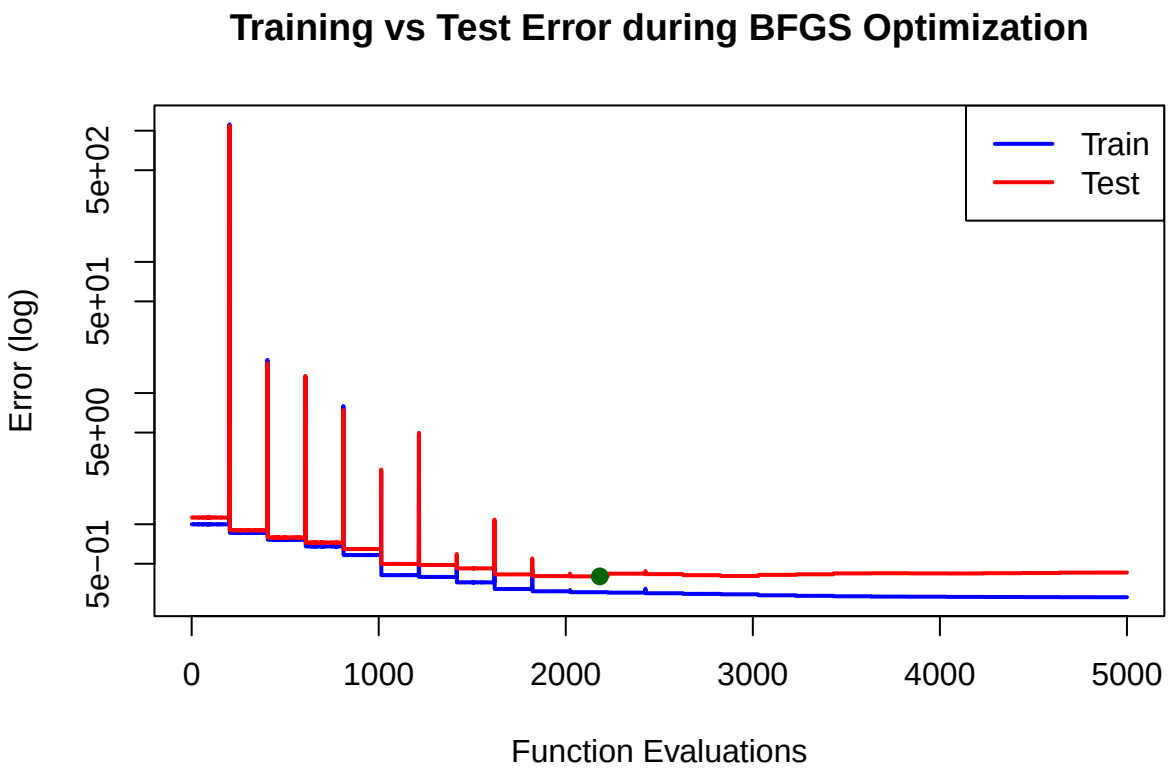
The linear regression model shows good performance on the training set. However, the test set performance is very poor (R^2 = -0.62), meaning the model predicts worse than a simple mean-based model. The large discrepancy between training and test errors indicates overfitting. This concludes that the model is not suitable for predictive purposes in its current form. One of the possible reasons is there are too many predictors relative to sample size. If this is the case, feature selection may be needed to improve generalization.

Task 4

The cost function for linear regression without intercept is:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - X_i\theta)^2$$

Plot the errors



Find optimal iteration with minumum error

```
## Optimal iteration: 2182 Test error: 0.4002329
```

The BFGS optimization successfully minimized the training cost, with training error approaching zero. The optimization achieved a much lower test error (≈ 0.40) compared to the linear regression in Step 3 (≈ 1.81). This indicates that the iterative approach provided better prediction. The iteration after the optimum might decrease the error for training data but will increase the error for test data due to overfitting.

Assignment 4

1. Practical approach for decreasing expected new data error

The book points out how models with low error in its training can fail to adapt to new data. You can estimate the error from new data by using some of the data for model testing (either through hold out method or K-fold). With this estimation the model designer can adjust its parameter so that both the generalization gap is small and the expected training error is high. In other words, we choose the one with the lowest estimated new data error. Another way is to increase the amount of observations since that decreases the generalization gap.

(MLFC page 66-71 and page 76-77).

2. Important aspect when selecting batches

The important aspect when selecting mini-batches stated in the book (page 125) is that they should be randomly formed from shuffled data, to ensure they represent the overall dataset. For instance, if the dataset is sorted based on output class, a mini-batch might only include observations of class. One wants to avoid

this, since such a mini-batch would "not give a good approximation of the gradient for the full dataset", which is what it is meant to do.

3. Loss function for inbalanced data

According to the MLFC book page 101-102, imbalanced data can be handled either by *modifying the loss function* or *by modifying the training data.* To implement modification of the loss function, we can use cost-sensitive loss, where errors on the minority class are *penalised C-times more heavily.* We can also modify the data instead, by *duplicating minority class* samples C times in the training set.

Appendix

Assignment 1 and question 1 in assignment 4 was done by Pontus Olsson.

Assignment 2 and question 2 in assignment 4 was done by Max Goldbeck-Löwe.

Assignment 3 and question 3 in assignemtn 4 was done by Paripurna Bawonoputro.

Appendix 1

```
# Assignment 1, done by Pontus Olsson

# Import and divide data ####
tecator <- read.csv("tecator.csv")
tecator <- tecator %>% select(-Sample)

N <- nrow(tecator)
set.seed(12345)
id <- sample(x = seq_len(N), size = floor(N * 0.5))
train <- tecator[-id,]
test <- tecator[id, ]

# 1. ####
model_formula <- paste0("Fat~", paste(paste0("Channel", 1:100), collapse = "+")) %>% as.formula()
linear <- lm(formula = model_formula, data = train)
kable(head(summary(linear)$coefficients[c(1:4, nrow(summary(linear)$coefficients)), ]), digits = 2, cap

## overview table ####
yhat <- predict(linear, newdata = train)
y <- train$Fat
train_mse <- sum((y-yhat)^2)/nrow(train)
train_r2 <- 1 - sum((y-yhat)^2)/sum((y-mean(y))^2)

yhat <- predict(linear, newdata = test)
y <- test$Fat
test_r2 <- sum((yhat - mean(y))^2)/sum((y-mean(y))^2)
test_mse <- sum((y-yhat)^2)/nrow(test)
null_mse <- sum((y-mean(y))^2)/nrow(test)


output <- data.frame(param = nrow(summary(linear)$coefficients),
           train_mse, train_r2,
           test_mse, test_r2, null_mse)
colnames(output) <- c("# parameters", "Train MSE", "Train $R^2$", "Test MSE", "Test $R^2$", "Null MSE
kable(output, digits = 4, caption = "Linear model overlook")

# 3. ####
# Read page 17 and onwards on lecture 2d
covariates <- train[, paste0("Channel", 1:100)]
response <- train[, "Fat"]


# The lambda argument in this function would control the range between lambda. Right now, we're going t
modelLasso <- glmnet(x = as.matrix(covariates), y = response, alpha = 1, family = "gaussian")
plot(modelLasso, xvar="lambda", label = TRUE, sub = "Figure 1: Coefficients and lambda, LASSO")
```

```r
## Coefficient table ####
inspect <- as.matrix(modelLasso$beta)
colnames(inspect) <- round(-log(modelLasso[["lambda"]]), digits = 3)

valid_col <- which(sapply(seq_len(ncol(inspect)), FUN = function(i){
  sum(inspect[, i] != 0) == 3
}))

inspect <- inspect[, c(min(valid_col) - 1, valid_col)]

valid_rows <- sapply(seq_len(nrow(inspect)), FUN = function(i){
  ifelse(all(inspect[i, ] == 0), yes = FALSE, no = TRUE)
}, simplify = TRUE)
inspect <- inspect[valid_rows, ]

kable(inspect, caption = "Coefficients through lambda iterations")

# 4. ####
# The lambda argument in this function would control the range between lambda. Right now, we're going t
modelRidge <- glmnet(x = as.matrix(covariates), y = response, alpha = 0, family = "gaussian")
plot(modelRidge, xvar="lambda", label = TRUE, sub = "Figure 2: Coefficients and lambda, Ridge")

# 5. ####
test_channel <- test[, paste0("Channel", 1:100)] %>% as.matrix()

set.seed(12345)
cv.out <- cv.glmnet(x = as.matrix(covariates), y = response, alpha = 1, family = "gaussian")
plot(cv.out, sub = "Figure 3: Best lambda from k-fold validation")

## table Model coefficients when lambda = -2.784 ####
output <- coef(cv.out, s = "lambda.min")
kable(output[output[, 1] != 0, ], col.names = "Coefficient", caption  = "Model coefficients when $\\laml

## table comparison lambda ####
bestlam <- cv.out$lambda.min
yhat <- predict(modelLasso, s = bestlam,
                      newx = test_channel)
opt_mse <- mean((yhat - test$Fat)^2)
opt_r2 <- sum((yhat - mean(y))^2)/sum((y-mean(y))^2)

yhat <- predict(modelLasso, s = exp(-4),
                      newx = test_channel)
mse_4 <- mean((yhat - test$Fat)^2)
r2_4 <- sum((yhat - mean(y))^2)/sum((y-mean(y))^2)

output <- data.frame(Opti = c(opt_mse, opt_r2),
                     lam4 = c(mse_4, r2_4), row.names =  c("MSE", "$R^2$"))
kable(x = output, col.names = c("Cross validated $\\lambda$", "$\\log(\\lambda = -4)$"), digits = 2)

## figure 4. ####
yhat <- predict(modelLasso, s = bestlam,
                      newx = test_channel)
```

```
ggplot(mapping = aes(y = test$Fat, x = yhat)) + geom_point(aes(col = "Observation")) + xlab("Predicted
```

Appendix 2

```
#Assignment 2, done by Max Goldbeck-Löwe

#1.
data <- read.csv("bank-full.csv", sep =";")

data <- subset(data, select = -duration)

data[] <- lapply(data, function(x) {
  if (is.character(x)) as.factor(x) else x
})

n <- dim(data)[1]
set.seed(12345)
id <- sample(1:n, floor(n * 0.4))
train <- data[id, ]

id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n * 0.3))
valid <- data[id2, ]

id3 <- setdiff(id1, id2)
test <- data[id3, ]

#2.
library(tree)

#a)
tree_default <- tree(y ~ ., data = train)
summary(tree_default)

#b)
tree_nodesize <- tree(y ~ ., data = train, minsize = 7000)
summary(tree_nodesize)

#c)
tree_deviance <- tree(y~ ., data = train, mindev = 0.0005)
summary(tree_deviance)

Yfit_default <- predict(tree_default, newdata = valid, type = "class")
table(valid$y, Yfit_default)
mean(Yfit_default != valid$y)

Yfit_nodesize <- predict(tree_nodesize, newdata = valid, type = "class")
table(valid$y, Yfit_nodesize)
mean(Yfit_nodesize != valid$y)

Yfit_deviance <- predict(tree_deviance, newdata = valid, type = "class")
table(valid$y, Yfit_deviance)
mean(Yfit_deviance != valid$y)

#3
```

```r
trainScore <- rep(0, 50)
testScore <- rep(0, 50)

for (i in 2:50) {
  prunedTree <- prune.tree(tree_deviance, best = i)
  trainScore[i] <- deviance(prunedTree)
  pred <- predict(prunedTree, newdata = valid, type = "tree")
  testScore[i] <- deviance(pred)
}

plot(2:50, trainScore[2:50], type = "b", col = "red", ylim = c(7000, max(trainScore, testScore)),
     xlab = "Number of Leaves", ylab = "Deviance")
points(2:50, testScore[2:50], type = "b", col = "blue")

trainScore[2:50]
testScore[2:50]

optimal_leaves <- which.min(testScore[2:50]) + 1
optimal_leaves

opt_tree <- prune.tree(tree_deviance, best = 22)
summary(opt_tree)
plot(opt_tree)
text(opt_tree, pretty = 0)

#4

test_pred <- predict(opt_tree, newdata = test, type = "class")
conf_matrix <- table(test$y, test_pred)
conf_matrix

accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
accuracy

TP <- conf_matrix["yes", "yes"]
FP <- conf_matrix["no", "yes"]
FN <- conf_matrix["yes", "no"]

precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
F1 <- 2 * precision * recall / (precision + recall)
F1

sum(data$y == "yes")
sum(data$y == "no")

#5


probs <- predict(opt_tree, newdata = test, type = "vector")

loss_matrix <- matrix(c(0, 1, 5, 0), nrow = 2, byrow = TRUE)
rownames(loss_matrix) <- colnames(loss_matrix) <- c("no", "yes")
```

```r
expected_loss_yes <- probs[, "yes"] * loss_matrix["yes", "yes"] + probs[, "no"] * loss_matrix["no", "yes
expected_loss_no  <- probs[, "yes"] * loss_matrix["yes", "no"]  + probs[, "no"] * loss_matrix["no", "no

test_pred_loss <- ifelse(expected_loss_yes < expected_loss_no, "yes", "no")

conf_matrix_loss <- table(test$y, test_pred_loss)
conf_matrix_loss

accuracy_loss <- sum(diag(conf_matrix_loss)) / sum(conf_matrix_loss)
accuracy_loss

TP_loss <- conf_matrix_loss["yes", "yes"]
FP_loss <- conf_matrix_loss["no", "yes"]
FN_loss <- conf_matrix_loss["yes", "no"]

precision_loss <- TP_loss / (TP_loss + FP_loss)
recall_loss <- TP_loss / (TP_loss + FN_loss)
F1_loss <- 2 * precision_loss * recall_loss / (precision_loss + recall_loss)
F1_loss

#6

logit_mod <- glm(y ~ ., data = train, family = binomial)

p_logit <- predict(logit_mod, newdata = test, type = "response")

p_tree <- probs[, "yes"]

compute_tpr_fpr <- function(p, y_true, thresholds) {
  res <- lapply(thresholds, function(pi) {
    pred <- ifelse(p > pi, "yes", "no")
    cm <- table(y_true, factor(pred, levels = c("no", "yes")))
    TP <- cm["yes", "yes"]; FN <- cm["yes", "no"]
    FP <- cm["no",  "yes"]; TN <- cm["no",  "no"]
    TPR <- ifelse((TP + FN) > 0, TP / (TP + FN), NA)
    FPR <- ifelse((FP + TN) > 0, FP / (FP + TN), NA)
    data.frame(threshold = pi, TPR = TPR, FPR = FPR)
  })
  do.call(rbind, res)
}

pis <- seq(0.05, 0.95, by = 0.05)

roc_tree  <- compute_tpr_fpr(p_tree,  test$y, pis)
roc_logit <- compute_tpr_fpr(p_logit, test$y, pis)

roc_tree
roc_logit

plot(roc_tree$FPR, roc_tree$TPR, type = "b", pch = 19, col = "forestgreen",
     xlim = c(0, 1), ylim = c(0, 1),
     xlab = "FPR",
     ylab = "TPR",
```

```
     main = "ROC Curves")
lines(roc_logit$FPR, roc_logit$TPR, type = "b", pch = 17, col = "steelblue")
abline(0, 1, lty = 3, col = "grey50")
```

Appendix 3

First we have to load the data from communities.csv

```
data <- read.csv("communities.csv", header = TRUE)
```

Task 1

In this task, we're not asked to divide the data to train/test.

First, we need to separate the features and the target Then we scale the features Implement PCA using eigen() and computing variation explained

```
X <- subset(data, select = -ViolentCrimesPerPop)

y <- data$ViolentCrimesPerPop

X_scaled <- scale(X)
```

Implement PCA using eigen() and computing variation explained

```
S <- cov(X_scaled)
eigen_decomp <- eigen(S)

eigenvalues <- eigen_decomp$values


var_explained <- eigenvalues / sum(eigenvalues)

cum_var <- cumsum(var_explained)

# Create variance explained table
pc_table <- data.frame(
  PC = 1:length(var_explained),
  Variance_Explained = var_explained,
  Cumulative = cum_var
)
rownames(pc_table) <- NULL

block1 <- pc_table[1:20, ]
block2 <- pc_table[21:40, ]

combined <- data.frame(
  PC_1to20 = block1$PC,
  VarExp_1to20 = round(block1$Variance_Explained, 4),
  CumVar_1to20 = round(block1$Cumulative, 4),

  PC_21to40 = block2$PC,
  VarExp_21to40 = round(block2$Variance_Explained, 4),
  CumVar_21to40 = round(block2$Cumulative, 4)
)

kable(combined, caption = "Variance Explained by Principal Components (1-40)")
```

Components needed to obtain at least 95% of variance in the data are

```r
which(cum_var >= 0.95)[1]
```

The proportion of variation explained by PC1 and PC2 are

```r
var_explained[1:2]
```

Task 2

```r
pca2 <- princomp(X_scaled, cor = FALSE)
```

```r
pc1_loadings <- pca2$loadings[,1]

plot(pc1_loadings, type = "b",
     main = "Trace Plot of PC1 Loadings",
     xlab = "Feature Index",
     ylab = "Loading Value")
```

The five variables with the highest loadings on PC1 are:

```r
pc1 <- pca2$loadings[,1]
top5_idx <- order(abs(pc1), decreasing = TRUE)[1:5]
colnames(X_scaled)[top5_idx]
pc1[top5_idx]
```

Plot

```r
pca2 <- princomp(X_scaled)

scores <- pca2$scores

library(ggplot2)

df_plot <- data.frame(
  PC1 = scores[,1],
  PC2 = scores[,2],
  Crime = data$ViolentCrimesPerPop
)

ggplot(df_plot, aes(x = PC1, y = PC2, color = Crime)) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_gradient(low = "blue", high = "red") +
  theme_minimal() + xlim(0, 10000)
  labs(
    title = "PC1 vs PC2 Scores Colored by Violent Crime Rate",
    x = "PC1",
    y = "PC2",
    color = "Crime Rate"
  )
```

Task 3

Split data into training and test. Then separate the features and the target.

```r
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
```

```r
train=data[id,]
test=data[-id,]

X_train <- subset(train, select = -ViolentCrimesPerPop)
y_train <- train$ViolentCrimesPerPop

X_test <- subset(test, select = -ViolentCrimesPerPop)
y_test <- test$ViolentCrimesPerPop
```

Then we have to apply scale to the feature and target. We should only use the parameter of train data to scale both train and test data.

```r
X_means <- apply(X_train, 2, mean)
X_sd   <- apply(X_train, 2, sd)

y_mean <- mean(y_train)
y_sd   <- sd(y_train)

X_train_scaled <- scale(X_train, center = X_means, scale = X_sd)
y_train_scaled <- (y_train - y_mean) / y_sd

X_test_scaled <- scale(X_test, center = X_means, scale = X_sd)
y_test_scaled <- (y_test - y_mean) / y_sd
```

Then we fit the model and use it for prediction

```r
linear_model <- lm(y_train_scaled ~ X_train_scaled)

train_pred <- predict(linear_model, newdata = as.data.frame(X_train_scaled))
test_pred <- predict(linear_model, newdata = as.data.frame(X_test_scaled))
```

```r
train_mse <- mean((y_train_scaled - train_pred)^2)
test_mse <- mean((y_test_scaled - test_pred)^2)

train_r2 <- 1 - sum((y_train_scaled - train_pred)^2) / sum((y_train_scaled - mean(y_train_scaled))^2)
test_r2 <- 1 - sum((y_test_scaled - test_pred)^2) / sum((y_test_scaled - mean(y_test_scaled))^2)

library(knitr)

results <- data.frame(
  Dataset = c("Train", "Test"),
  MSE = c(train_mse, test_mse),
  R2 = c(train_r2, test_r2)
)

kable(results, digits = 4,
      caption = "Model Performance on Training and Test Sets")
```

Task 4

The cost function for linear regression without intercept is:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - X_i\theta)^2$$

```r
cost_fn <- function(theta, X, y) {
  preds <- X %*% theta
  mean((y - preds)^2)
}
```

```r
train_errors <- c()
test_errors <- c()

cost_with_tracking <- function(theta) {
  # Compute training error
  train_err <- cost_fn(theta, X_train_scaled, y_train_scaled)
  test_err <- cost_fn(theta, X_test_scaled, y_test_scaled)

  # Store errors
  train_errors <<- c(train_errors, train_err)
  test_errors <<- c(test_errors, test_err)

  return(train_err)
}

init_theta <- rep(0, ncol(X_train_scaled))
result <- optim(par = init_theta, fn = cost_with_tracking, method = "BFGS", control = list(trace = 1, ma
```

Plot the errors

```r
optimal_iter <- which.min(test_errors)

start <- 1
end   <- 5000


idx <- start:min(end, length(train_errors))

plot(idx,
     train_errors[idx],
     type = "l", col = "blue", lwd = 2, log = "y",
     xlim = c(start, end),
     ylim = range(c(train_errors[idx], test_errors[idx])),
     xlab = "Function Evaluations", ylab = "Error (log)",
     main = "Training vs Test Error during BFGS Optimization")

lines(idx, test_errors[idx], col = "red", lwd = 2)

legend("topright",
       legend = c("Train", "Test"),
       col = c("blue", "red"),
       lty = 1, lwd = 2)


if (optimal_iter >= start && optimal_iter <= end) {
  points(optimal_iter, test_errors[optimal_iter],
         col = "darkgreen", pch = 19, cex = 1.1)
}
```

Find optimal iteration with minumum error

```r
cat("Optimal iteration:", optimal_iter, "Test error:", test_errors[optimal_iter], "\n")
```