Lab 2 Block 1

Hugo Morvan & Marijn Jaarsma & Simon Jorstedt

2023-12-06

Statement of Contribution:

- Assignment 1 was coded and analysed by Hugo
- Assignment 2 was coded and analysed by Marijn
- Assignment 3 was coded and analysed by Simon

All code and questions were analysed and discussed together.

Assignment 1. Explicit regularization

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.

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

1.1

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.

```
## train error (using predict()): 0.005549593
```

```
## train error (using fitted()): 0.005549593
```

```
## test error (using predic()): 808.9211
```

```
## test error (using fitted()): 303.8813
```

Test results are awful, is is probably overfitting. Also the dataset size is very small (215 observations) therefore test and train are both ~100 observations, which is barely more than the number of features (100).

1.2

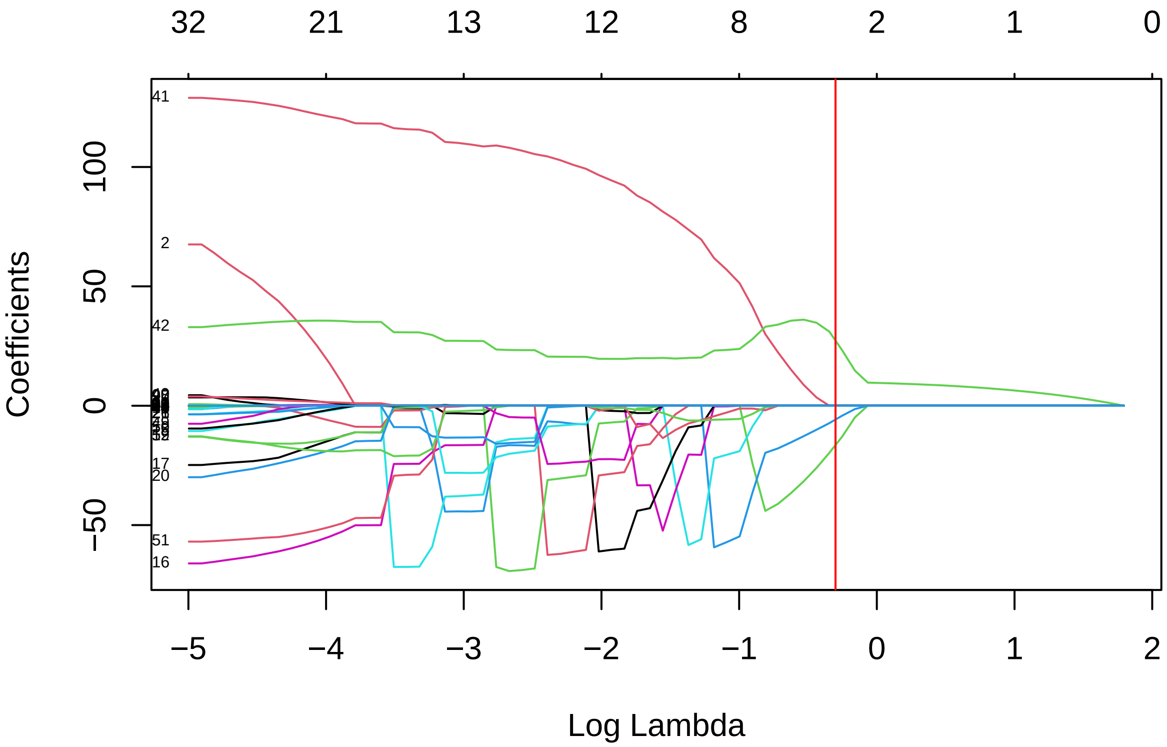
2. Assume now that Fat can be modeled as a LASSO regression in which all Channels are used as features. Report the cost function that should be optimized in this scenario.

The cost function that should be optimized is the following:

$$\hat{\theta}^{lasso} = argmin\{\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta_0 - \theta_1 x_{1i} - ... - \theta_p x_{pi})^2 + \lambda\sum_{j=1}^{p}|\theta_j|\}$$

1.3

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 (log lambda) and interpret this plot.



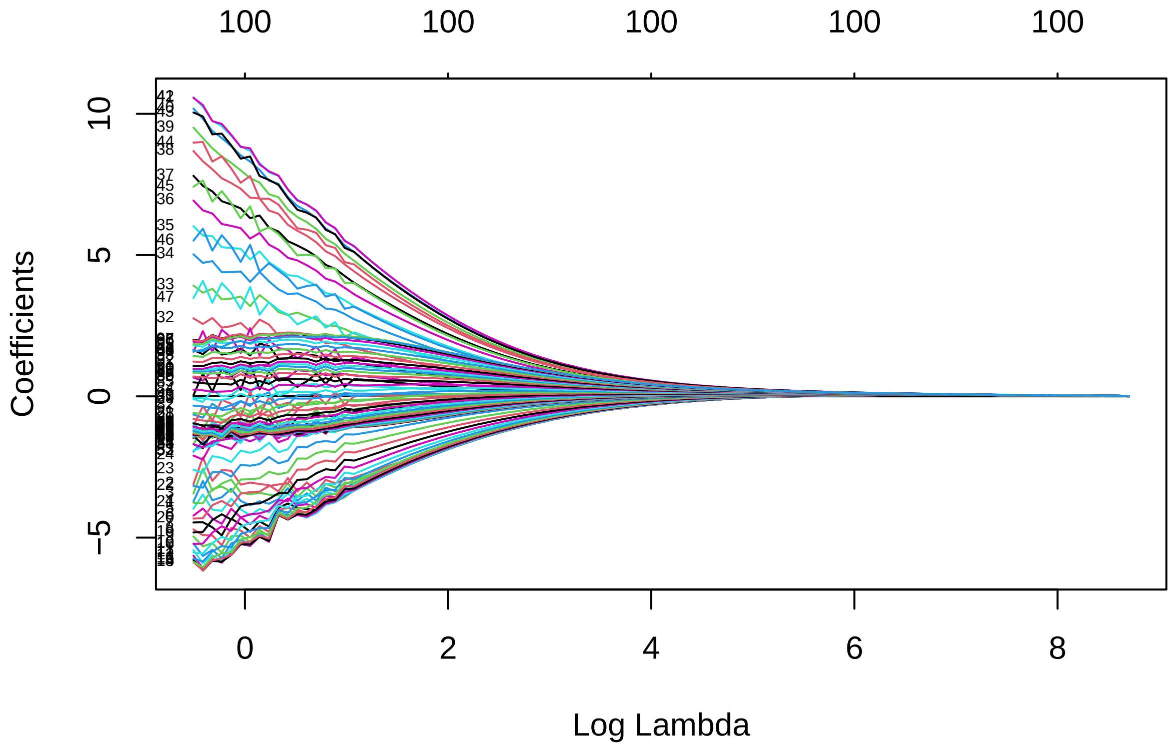
Interpret this plot: the higher the penalty factor, the more coefficients are set to zero.

What value of the penalty factor can be chosen if we want to select a model with only three features?

At log(lambda) = -0.3, there are only 3 non-zero features left, Hence we can choose $\lambda = e^{-0.3} = 0.74$.

1.4

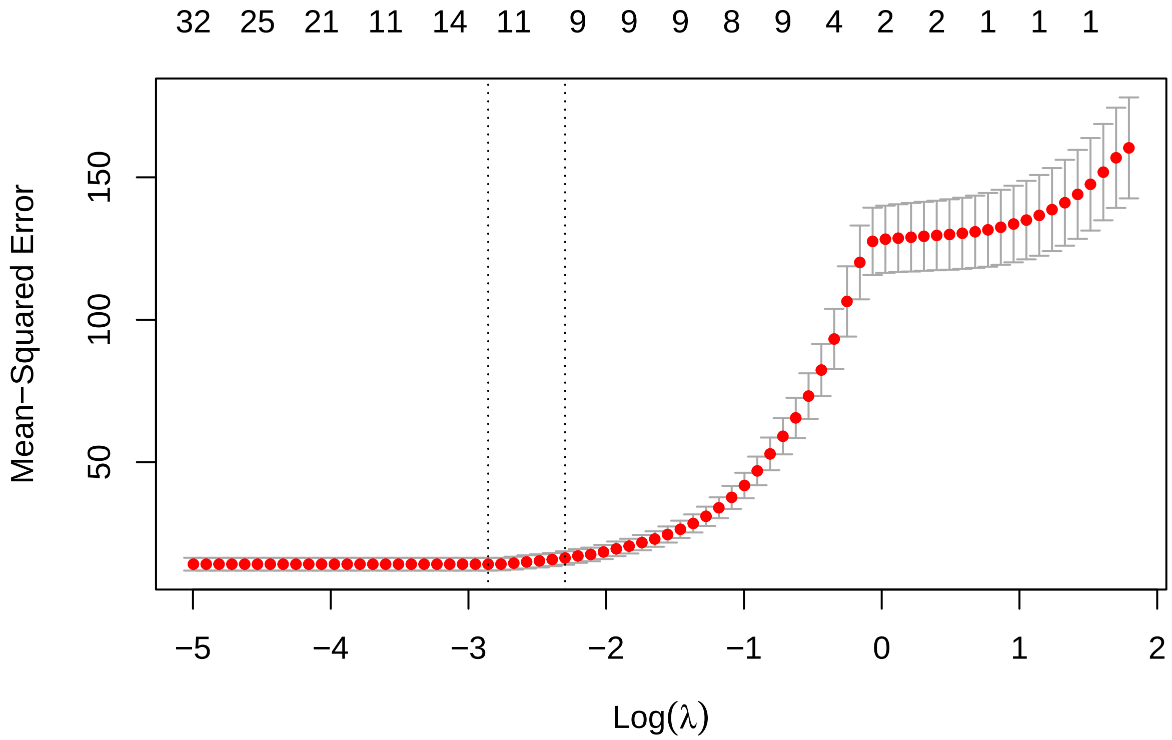
4. Repeat step 3 but fit Ridge instead of the LASSO regression and compare the plots from steps 3 and 4. Conclusions?



The higher the penalty factor, the smaller the coefficients. The coefficients are never set to zero, unlike in the LASSO regression.

1.5

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 $\log\lambda$.



```
## best lambda: 0.05744535
```

Report the optimal λ and how many variables were chosen in this model.

The optimal λ is 0.05744535 or log(lambda) = -2.86.

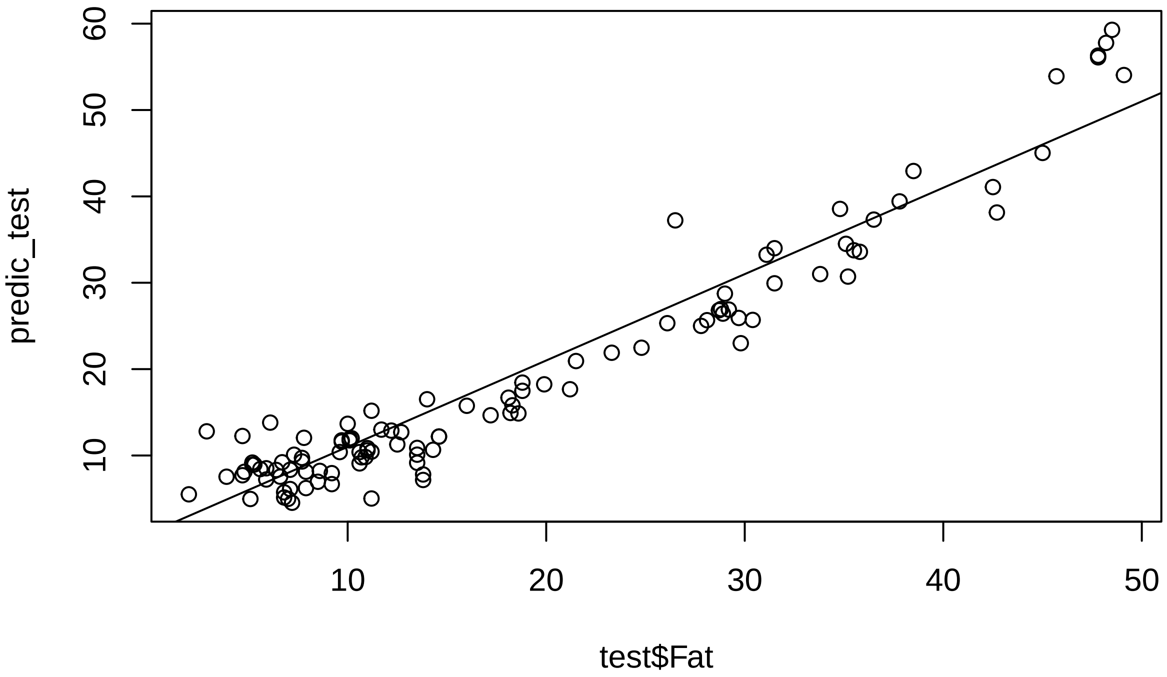
```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  29.622922448
## Sample        0.006967984
## Channel1      .
## Channel2      .
## Channel3      .
## Channel4      .
## Channel5      .
## Channel6      .
## Channel7      .
## Channel8      .
## Channel9      .
## Channel10     .
## Channel11     .
## Channel12     .
## Channel13   -44.086683444
## Channel14   -37.211163911
## Channel15   -16.407635384
## Channel16    -3.448269598
## Channel17     .
## Channel18     .
## Channel19     .
## Channel20     .
## Channel21     .
## Channel22     .
## Channel23     .
## Channel24     .
## Channel25     .
## Channel26     .
## Channel27     .
## Channel28     .
## Channel29     .
## Channel30     .
## Channel31     .
## Channel32     .
## Channel33     .
## Channel34     .
## Channel35     .
## Channel36     .
## Channel37     .
## Channel38     .
## Channel39     .
## Channel40   108.603577898
## Channel41    27.093395458
## Channel42     .
## Channel43     .
## Channel44     .
## Channel45     .
## Channel46     .
## Channel47     .
## Channel48     .
## Channel49     .
## Channel50     .
## Channel51    -1.888132124
## Channel52   -13.191414273
## Channel53   -28.033425615
## Channel54     .
## Channel55     .
## Channel56     .
## Channel57     .
## Channel58     .
## Channel59     .
## Channel60     .
## Channel61     .
## Channel62     .
## Channel63     .
## Channel64     .
## Channel65     .
## Channel66     .
## Channel67     .
## Channel68     .
## Channel69     .
## Channel70     .
## Channel71      .
## Channel72      .
## Channel73      .
## Channel74      .
## Channel75      .
## Channel76      .
## Channel77      .
## Channel78      .
## Channel79      .
## Channel80      .
## Channel81      .
## Channel82      .
## Channel83      .
## Channel84      .
## Channel85      .
## Channel86      .
## Channel87      .
## Channel88      .
## Channel89      .
## Channel90      .
## Channel91      .
## Channel92      .
## Channel93      .
## Channel94      .
## Channel95      .
## Channel96      .
## Channel97      .
## Channel98      .
## Channel99      .
```

9 variables were chosen in this model.

Does the information displayed in the plot suggests that the optimal λ value results in a statistically significantly better prediction than $\log \lambda = -4$?

The information displayed in the plot suggests that the optimal lambda has same prediction MSE as 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.



```
## integer(0)
```

The model predictions seems to be good as the points are close to the line y=x.

Assignment 2: Decision trees and logistic regression for bank marketing

2.1

Import the data to R, remove variable "duration" and divide into training/validation/test as 40/30/30: use data partitioning code specified in Lecture 2a.

2.2

Fit decision trees to the training data so that you change the default settings one by one (i.e. not simultaneously): a. Decision Tree with default settings. b. Decision Tree with smallest allowed node size equal to 7000. c. Decision trees minimum deviance to 0.0005. and report the misclassification rates for the training and validation data. Which model is the best one among these three? Report how changing the deviance and node size affected the size of the trees and explain why.

```
## Misclassification rate model a:
##  Train: 0.1048441
##  Validation: 0.1092679
##
## Misclassification rate model b:
##  Train: 0.1048441
##  Validation: 0.1092679
##
## Misclassification rate model c:
##  Train: 0.09400575
##  Validation: 0.1119221
```

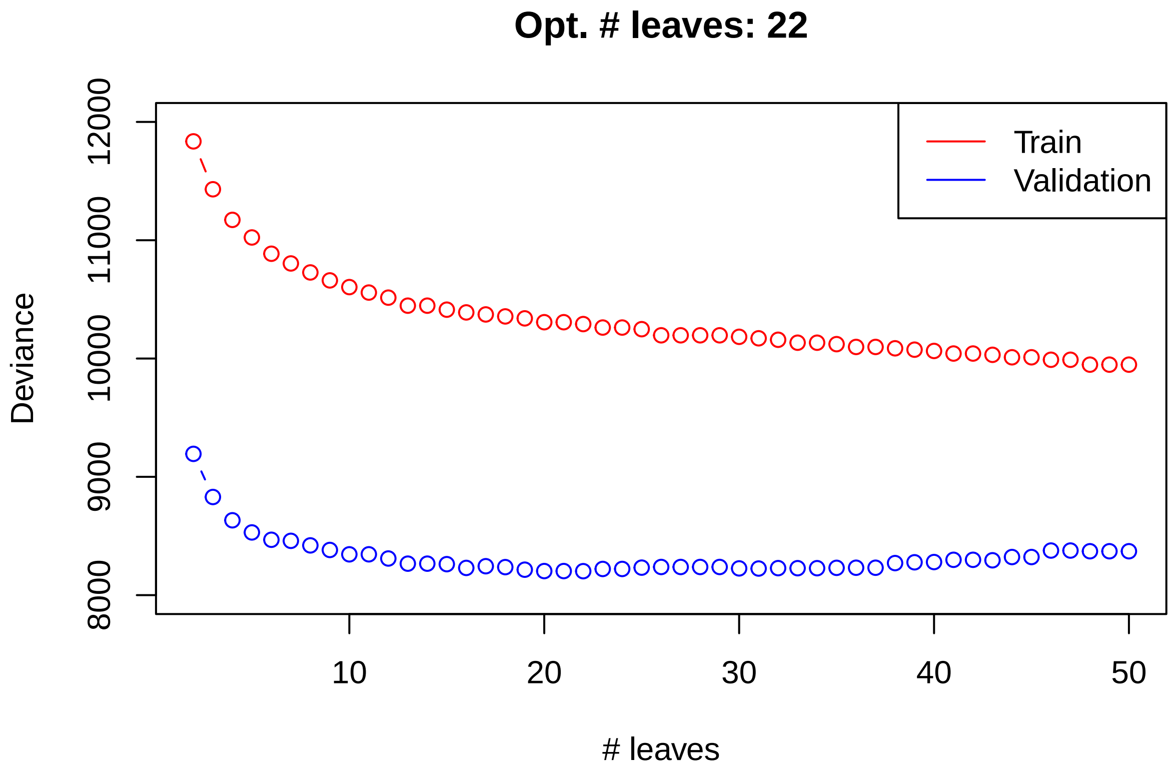
Model a and b are incredibly close to each other in performance. Both of their validation errors are better than model c, but since model b has one set of terminal nodes less than model a this model is less complex and thus the better option between the two.

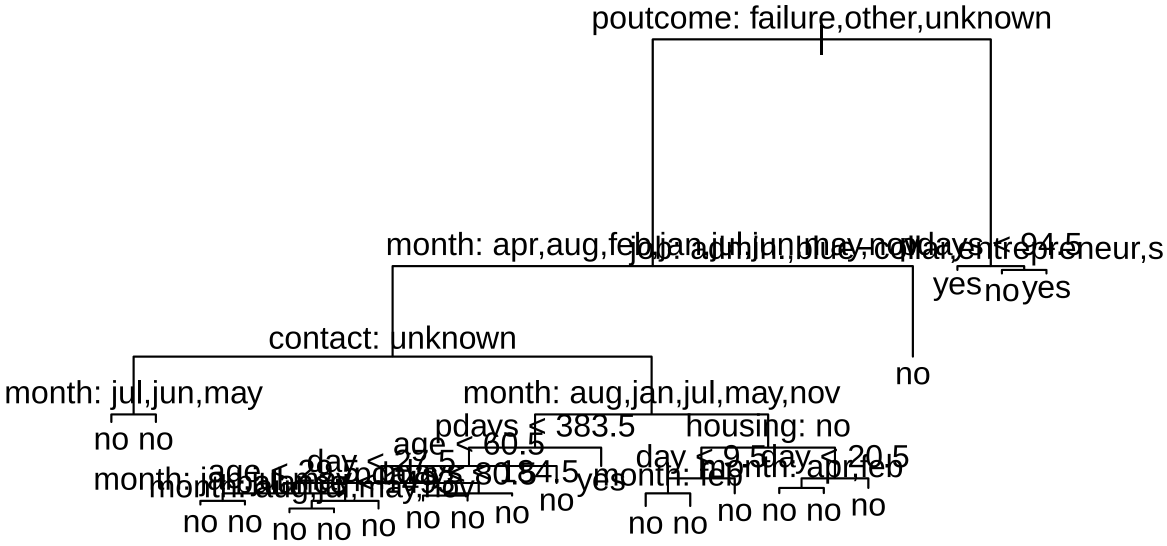
Setting the minimal node size to 7000 in model b means resulted in the pruning of one branch. The split that occurred here would have resulted in a node with less than 7000 observations, which was not allowed with this option.

The default minimum deviance is 0.01 (https://cran.r-project.org/web/packages/tree/tree.pdf). Setting the minimum deviance at a much smaller number (0.0005) means that more branches are allowed and results in a much bigger tree.

2.3

Use training and validation sets to choose the optimal tree depth in the model 2c: study the trees up to 50 leaves. Present a graph of the dependence of deviances for the training and the validation data on the number of leaves and interpret this graph in terms of bias-variance tradeoff. Report the optimal amount of leaves and which variables seem to be most important for decision making in this tree. Interpret the information provided by the tree structure (not everything but most important findings).


Opt. # leaves: 22
Train
Validation
Deviance
8000
9000
10000
11000
12000
10
20
30
40
50
# leaves



The number of leaves that produces the lowest deviance on the validation set is 22. Interestingly, the deviance on the training data is higher than that on the validation data for all values of k. If k goes beyond 50, they get closer to each other, but validation will always remain lower. Though it should also be tested on the training data, this at least means the model is not overfitting. The poutcome variable is the first split, after which pdays is used for a split followed by entrepeneur. On the left side of the tree, where most decisions lead to "no", poutcome is followed by month, contact, month again, and pdays to lead to a yes decision. As the tree gets bigger it becomes difficult to read the overlapping text, but month seems to be an important variable that comes back often, and pdays a variable that leads to a split between "yes" and "no".

As the number of leaves on the tree increases, the bias of the model gets reduced. However, the variance also goes up as the deviance of the validation set increases with a more complex model, signaling overfitting. Purely aiming to improve the bias of the model, then, is not necessarily a good thing if this means the variance is not staying constant. Other methods, such as bagging or random forests, may be applied to improve one of these while keeping the other constant (or at least not increasing the other enough for an undesirable effect), but in the case of a simple decision tree such as we have here, the sweet spot is somewhere in the middle.

2.4

Estimate the confusion matrix, accuracy and F1 score for the test data by using the optimal model from step 3. Comment whether the model has a good predictive power and which of the measures (accuracy or F1-score) should be preferred here.

```
## Number of leaves: 22
## Accuracy: 0.891
## F1: 0.225
```

```
##
## Confusion matrix:
```

```
##      Pred
## True     no   yes
##   no  11872   107
##   yes  1371   214
```

The model is quite good at prediction “no”, but bad at prediction “yes”. There are much fewer “yes” values in the data (1585 versus 11979 “no”), but the model predicts many false negatives (1371 out of 1585). The accuracy measure would suggest this model performs very well, but if we really care about correctly predicting “yes” values the F1 measure would be a better measure to consider. This takes into account precision and recall, so it works better on an imbalanced dataset. The F1 score for this model is quite low (0.225), suggesting that it is bad at accurately predicting positive values. This can also be seen from the confusion matrix. If we do not care much about accurately predicting “yes” values, we can accept the accuracy rate as a good measure, but considering the imbalanced nature of the data, F1 would be preferred.

2.5

Perform a decision tree classification of the test data with the following loss matrix,

$$L = \begin{pmatrix} 0 & 5 \\ 1 & 0 \end{pmatrix}$$

and report the confusion matrix for the test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```
## Number of leaves: 22
## Accuracy: 0.859
## F1: 0.449
##
## Confusion matrix:
```

```
##      Pred
## True     no   yes
##   no  10880  1099
##   yes   807   778
```

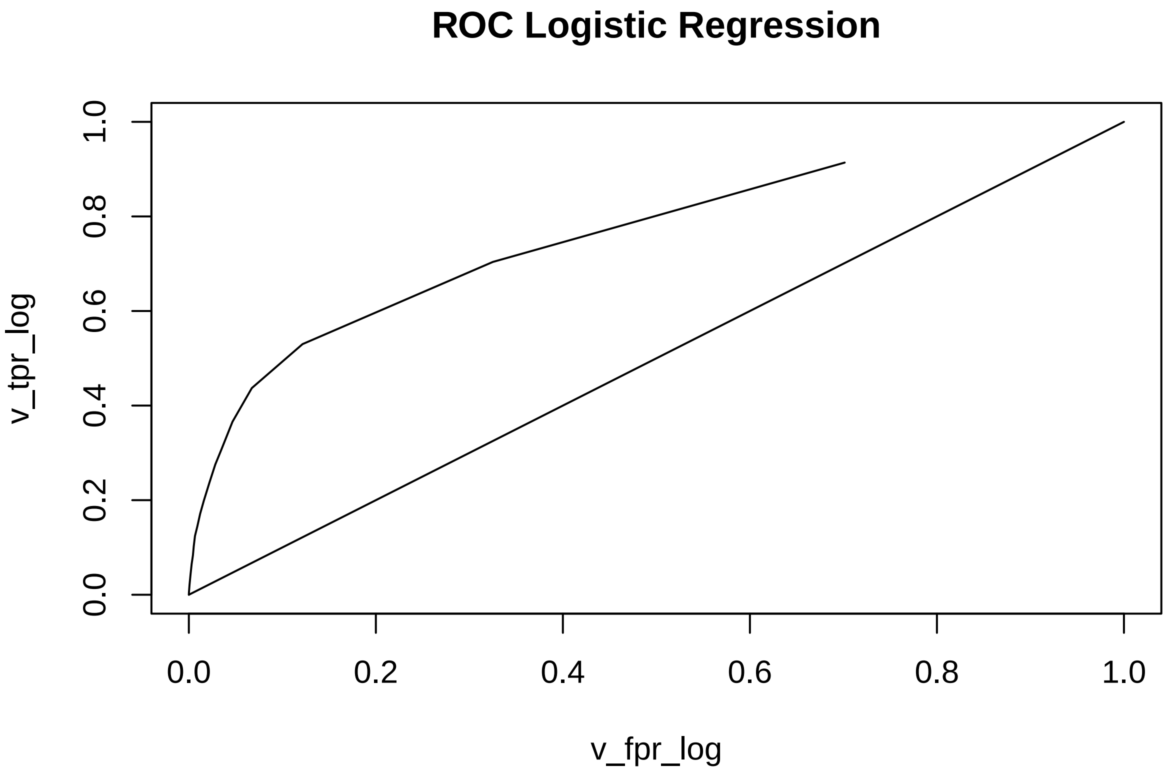
Using the loss matrix, the true positive rate has gone up, along with the false positive rate. The false negative rate has gone down slightly, but is still not very good compared to the true positive rate. As we are penalizing false negative rate by five times what we are penalizing false positives with, the model is now much more likely to predict “yes”. In combination with the unbalanced nature of the data, this has led to a highly increase false positive rate, rather than a better true positive rate.

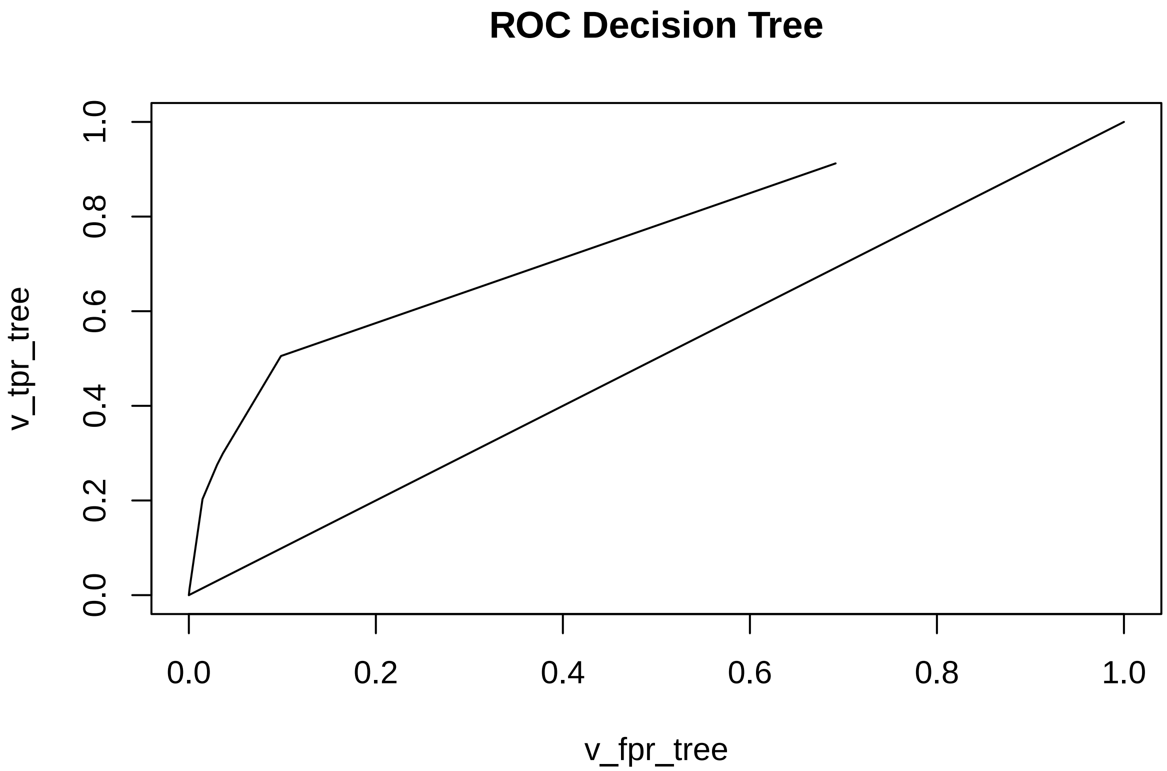
2.6

Use the optimal tree and a logistic regression model to classify the test data by using the following principle:

$$\hat{Y} = yes\ if\ p(Y =' yes'|X) > \pi,\ otherwise\ \hat{Y} = no$$

where $pi = 0.05, 0.1, 0.15, ..., 0.9, 0.95$. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion? Why precision-recall curve could be a better option here?


ROC Logistic Regression
v_tpr_log
0.0
0.2
0.4
0.6
0.8
1.0
v_fpr_log
0.0
0.2
0.4
0.6
0.8
1.0



Looking purely at the ROC curves, it seems that the logistic regression performs slightly better than the decision tree. However, considering the unbalanced nature of the data, it might be better to consider a precision-recall curve.

Assignment 3

Setup

Assignment 3.1

We are given some crime data. After excluding the response variable ViolentCrimesPerPop we analyse it using PCA, and find that the first component (PC1) explains about 25.017 % of the variance, and the second component (PC2) explains about 16.936 % of the variance in the data. To achieve 95 % variance explained, the first 35 principal components must be used.

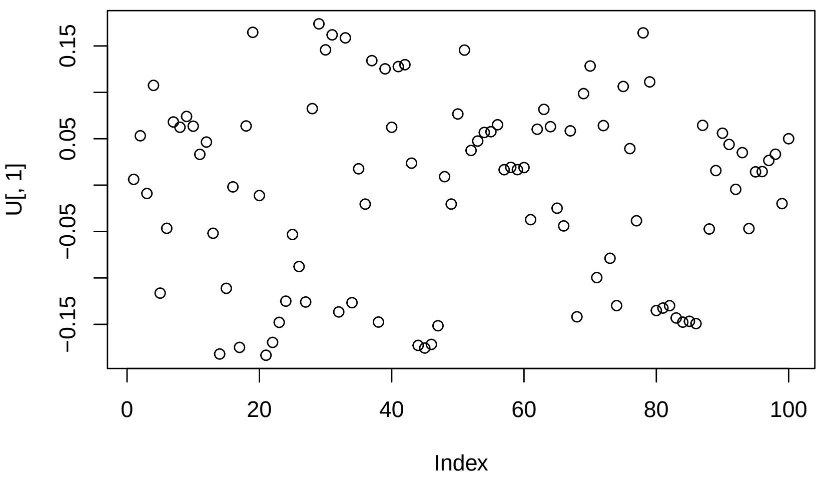
```
## Variance explained by PC1: 25.017 %
```

```
## Variance explained by PC2: 16.936 %
```

```
## Variance explained by first 35 PC’s: 95.269 %
```

Assignment 3.2

Now we repeat the PCA performed in Assignment 3.1 using prebuilt R functionality. We find that the contributions to PC1 are fairly well spread across the variables, and the five with largest (absolute) contribution are very close in their contribution. There is clearly underlying dependencies among these variables. Median family income and median income are of course highly correlated. The other variables are also likely highly explanatory for income and wealth in general.

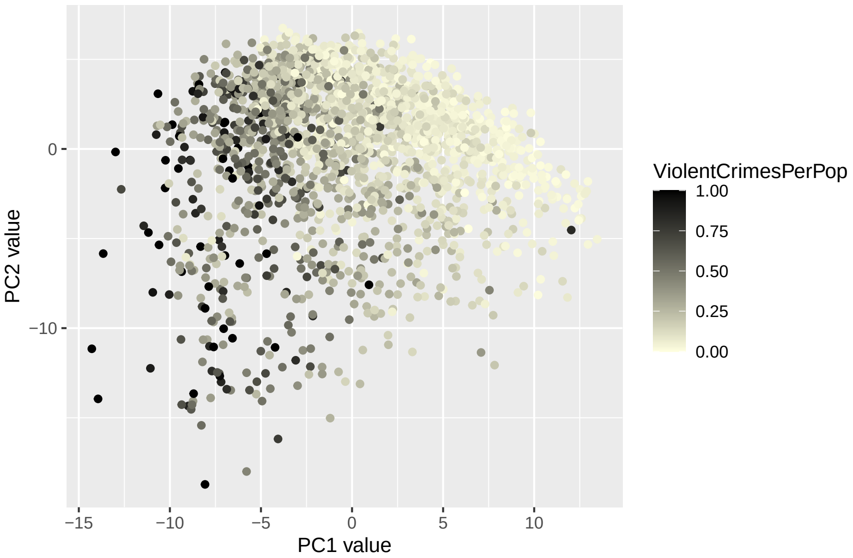


```
## Five largest contributors to PC1 are:

## medFamInc
## medIncome
## PctKids2Par
## pctWInvInc
## PctPopUnderPov
```

Next, in Figure 3.3 we plot the data expressed by the vectors PC1 and PC2, and map the color of points to the Violent Crimes per Population feature. The result indicates first of all that most observations have a low value for the crime variable. There is a very large high-density region around the point $(0, 0)$ (by design). This cluster extends roughly such that PC1 and PC2 seem to be negatively correlated. For low values of PC2 (roughly below -5), there is however a less dense and moderatly sized group of datapoints where the average crime variable tends to have larger values. However these datapoints with higher crime rates appear to be fairly uniformly spread across the range of PC2. The component PC1 appears to be suboptimal in predicting density, but a significantly large number of datapoints with high crime rates have negative PC1 values, compared to among the datapoints with positive PC1 values. Thus PC1 may be useful for predicting crime rate.

Fig 3.3. Crime data expressed in PC1 and PC2



Assignment 3.3

Now we wish to attempt to construct a linear prediction model. First we split the available data into train and test groups (50/50). The Mean Squared Error for train and test data is reported below. There is a difference between them, but it should be noted that the MSE highly depends on problem context, and so it is difficult to make any strong statements about the significance of the difference between the achieved train and test errors. However, the test error is roughly 50 % larger than the train error, and this might indicate overfitting.

```
## Train mse: 0.2752071
```

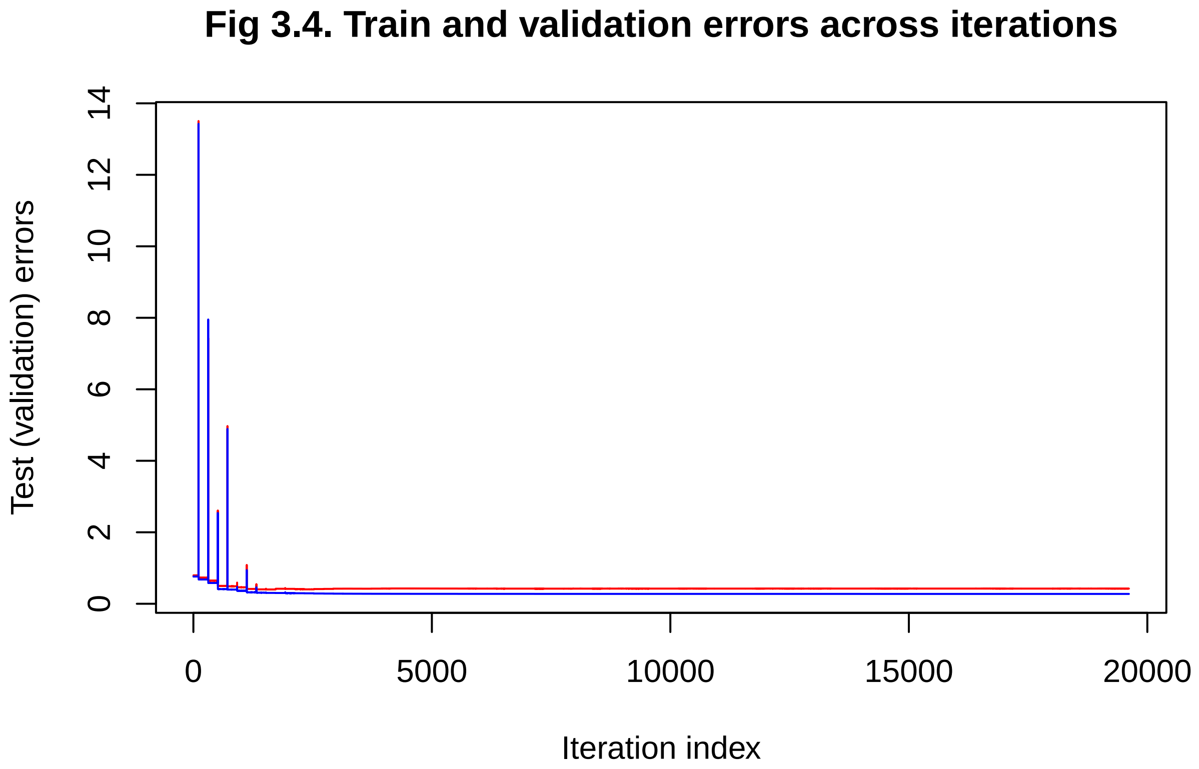
```
## Test mse: 0.4248011
```

Assignment 3.4

Now we will instead use prebuilt R functionality to try to optimise our model. We run the `optim` function and plot the resulting train and test (validation) errors against the iteration index in Figure 3.4. The optimisation quickly achieves a roughly optimal solution, and spends many iterations making small, unnecessary adjustments. We find an appropriate stopping place where the test (validation) error is minimal. When comparing the train and test errors at this point with the train and test error achieved in Assignment 3.3, we find that the train error has increased slightly (performing worse), while the test error has remained roughly the same (in fact it has decreased) so that accuracy is maintained. This indicates that the level of overfitting is lower in the new model, which is a generally desireable outcome.

It should be noted that the test data has been inappropriately used in Assignment 3.3 and 3.4, as per the explicit lab instructions. It has essentially been used as validation data. Instead, test data should be reserved and used only in the end to evaluate our final model.

```
## initial  value 0.998997
## iter  10 value 0.308219
## iter  20 value 0.280144
## iter  30 value 0.276877
## iter  40 value 0.276017
## iter  50 value 0.275680
## iter  60 value 0.275499
## iter  70 value 0.275372
## iter  80 value 0.275296
## iter  90 value 0.275243
## iter 100 value 0.275221
## final  value 0.275221
## stopped after 100 iterations
```



```
## Train mse: 0.3207301
```

```
## Test mse: 0.4144676
```

Appendix

```
knitr::opts_chunk$set(echo = FALSE)
library(glmnet)
library(ggplot2)
library(tree)
library(rpart)
```

```r
# 1.0
data=read.csv("tecator.csv")

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

# 1.1
lin_mod <- lm(Fat ~ . -Moisture -Protein, data = train)
#summary(lin_mod)

predic_train <- predict.lm(lin_mod, train)
train_error <- mean((train$Fat - predic_train)^2)
cat("train error (using predict()):", train_error, "\n")
fitted_train <- fitted(lin_mod, train)
train_error2 <- mean((train$Fat - fitted_train)^2)
cat("train error (using fitted()):", train_error2, "\n")

predic_test <- predict.lm(lin_mod, test)
test_error <- mean((test$Fat - predic_test)^2)
cat("test error (using predic()):", test_error, "\n")
fitted_test <- fitted(lin_mod, test)
test_error2 <- mean((test$Fat - fitted_test)^2)
cat("test error (using fitted()):", test_error2, "\n")

# 1.3
lasso <- glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 1)
plot(lasso, xvar = "lambda", label = TRUE)
#add an x=0.4 line
abline(v = -0.3, col = "red")

# 1.4
ridge <- glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 0)
plot(ridge, xvar = "lambda", label = TRUE)

# 1.5
cv_lasso <- cv.glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 1)
plot(cv_lasso)
best_lambda <- cv_lasso$lambda.min
cat("best lambda:", best_lambda, "\n")
coef(cv_lasso, s = "lambda.min")

predic_test <- predict(cv_lasso, newx = as.matrix(test[,1:100]), s = "lambda.min")
plot(test$Fat, predic_test) + abline(1, 1)

# 2.1
# Read in data, remove 'duration' column, and add 1/0 bool column for y
df_bank <- read.csv("bank-full.csv", sep=";")
df_bank <- df_bank[, names(df_bank) != "duration"]

# Set categorical column as factor for modeling
# https://stackoverflow.com/questions/54768433/nas-introduced-by-coercion-error-when-using-tree-functio
# https://www.listendata.com/2015/05/converting-multiple-numeric-variables.html
df_bank[sapply(df_bank, is.character)] <- lapply(df_bank[sapply(df_bank, is.character)], as.factor)

# Train/validation/test split
n <- nrow(df_bank)
set.seed(12345)
id_train <- sample(1:n, floor(n * 0.4))
df_train <- df_bank[id_train,]

id_rest <- setdiff(1:n, id_train)
set.seed(12345)
id_valid <- sample(id_rest, floor(n * 0.3))
df_valid <- df_bank[id_valid,]

id_test <- setdiff(id_rest, id_valid)
df_test <- df_bank[id_test,]

# 2.2a
n <- nrow(df_train)
mod1 <- tree(y ~ ., data=df_train)
pred_train1 <- predict(mod1, newdata=df_train, type="class")
pred_valid1 <- predict(mod1, newdata=df_valid, type="class")
misclass_train1 <- (table(df_train$y, pred_train1)["no", "yes"] + table(df_train$y, pred_train1)["yes",
misclass_valid1 <- (table(df_valid$y, pred_valid1)["no", "yes"] + table(df_valid$y, pred_valid1)["yes",

# plot(mod1)
# text(mod1, pretty=0)

# 2.2b
# https://search.r-project.org/CRAN/refmans/tree/html/tree.control.html
mod2 <- tree(y ~ ., data=df_train, control=tree.control(nobs=n, minsize=7000))
pred_train2 <- predict(mod2, newdata=df_train, type="class")
pred_valid2 <- predict(mod2, newdata=df_valid, type="class")
misclass_train2 <- (table(df_train$y, pred_train2)["no", "yes"] + table(df_train$y, pred_train2)["yes",
misclass_valid2 <- (table(df_valid$y, pred_valid2)["no", "yes"] + table(df_valid$y, pred_valid2)["yes",

# plot(mod2)
# text(mod2, pretty=0)

# 2.2c
# https://search.r-project.org/CRAN/refmans/tree/html/tree.control.html
mod3 <- tree(y ~ ., data=df_train, control=tree.control(nobs=n, mindev=0.0005))
pred_train3 <- predict(mod3, newdata=df_train, type="class")
pred_valid3 <- predict(mod3, newdata=df_valid, type="class")
misclass_train3 <- (table(df_train$y, pred_train3)["no", "yes"] + table(df_train$y, pred_train3)["yes",
misclass_valid3 <- (table(df_valid$y, pred_valid3)["no", "yes"] + table(df_valid$y, pred_valid3)["yes",

# plot(mod3)
# text(mod3, pretty=0)

# Report values
cat("Misclassification rate model a:\n",
    "\tTrain: ", misclass_train1, "\n",
    "\tValidation: ", misclass_valid1, "\n\n",
    "Misclassification rate model b:\n",
    "\tTrain: ", misclass_train2, "\n",
    "\tValidation: ", misclass_valid2, "\n\n",
    "Misclassification rate model c:\n",
    "\tTrain: ", misclass_train3, "\n",
    "\tValidation: ", misclass_valid3,
    sep="")

# 2.3
train_score <- rep(0, 50)
valid_score <- rep(0, 50)

for (i in 2:50) {
  pruned_tree <- prune.tree(mod3, best=i)
  pred <- predict(pruned_tree, newdata=df_valid, type="tree")
  train_score[i] <- deviance(pruned_tree)
  valid_score[i] <- deviance(pred)

  # plot(pruned_tree)
  # text(pruned_tree, pretty=0)
  # title(paste0("k=", i))
}

# https://r-coder.com/add-legend-r/
plot(2:50, train_score[2:50], type="b", col="red", ylim=c(8000, 12000), xlab="# leaves", ylab="Deviance
points(2:50, valid_score[2:50], type="b", col="blue")
legend(x="topright", legend=c("Train", "Validation"), lty=c(1, 1), col=c("red", "blue"))
title(paste0("Opt. # leaves: ", match(min(valid_score[2:50]), valid_score)))

opt_pruned_tree <- prune.tree(mod3, best=22)
plot(opt_pruned_tree)
text(opt_pruned_tree, pretty=0)

# 2.4
# Find number of leaves with minimum validation score
best_k <- match(min(valid_score[2:length(valid_score)]), valid_score)
pruned_mod <- prune.tree(mod3, best=best_k)
pred_test <- predict(pruned_mod, newdata=df_test, type="class")

# Confusion matrix
# https://datascience.stackexchange.com/questions/27132/decision-tree-used-for-calculating-precision-ac
conf_mat <- table(df_test$y, pred_test, dnn=c("True", "Pred"))

fun_scoring <- function(conf_mat) {
  acc <- (conf_mat["yes", "yes"] + conf_mat["no", "no"]) / sum(conf_mat)
  prec <- conf_mat["yes", "yes"] / (conf_mat["yes", "yes"] + conf_mat["no", "yes"])
  rec <- conf_mat["yes", "yes"] / (conf_mat["yes", "yes"] + conf_mat["yes", "no"])
  F1 <- (2 * prec * rec) / (prec + rec)

  cat("Number of leaves: ", best_k, "\n",
      "Accuracy: ", round(acc, 3), "\n",
      "F1: ", round(F1, 3), "\n\n",
      "Confusion matrix:\n",
      sep="")
  conf_mat
}

fun_scoring(conf_mat)

# 2.5
# Ordered differently according to the documentation
# https://stackoverflow.com/questions/49646377/loss-matrix-in-rs-package-rpart
loss_matrix <- matrix(c(0, 5, 1, 0), nrow=2, ncol=2)
mod4 <- rpart(y ~ ., data=df_train, method="class", parms=list(loss=loss_matrix))
pred_test4 <- predict(mod4, newdata=df_test, type="class")
conf_mat <- table(df_test$y, pred_test4, dnn=c("True", "Pred"))
fun_scoring(conf_mat)

# 2.6
v_pi <- seq(from=0.05, to=0.95, by=0.05)
log_mod <- glm(y ~ ., family=binomial(link="logit"), data=df_train)
pred_log_prob <- predict(log_mod, newdata=df_train, type="response")
pred_tree_prob <- predict(mod4, newdata=df_train, type="prob")

v_fpr_log <- vector()
v_tpr_log <- vector()
v_fpr_tree <- vector()
v_tpr_tree <- vector()
for (pi in v_pi) {
  pred_log <- ifelse(pred_log_prob > pi, "yes", "no")
  pred_tree <- ifelse(pred_tree_prob[, "yes"] > pi, "yes", "no")

  fp_log <- length(pred_log[(pred_log == "yes") & (df_train$y == "no")])
  fn_log <- length(pred_log[(pred_log == "no") & (df_train$y == "yes")])
  tp_log <- length(pred_log[(pred_log == "yes") & (df_train$y == "yes")])
  tn_log <- length(pred_log[(pred_log == "no") & (df_train$y == "no")])
  v_fpr_log <- append(v_fpr_log, fp_log / (fp_log + tn_log))
  v_tpr_log <- append(v_tpr_log, tp_log / (tp_log + fn_log))

  fp_tree <- length(pred_tree[(pred_tree == "yes") & (df_train$y == "no")])
  fn_tree <- length(pred_tree[(pred_tree == "no") & (df_train$y == "yes")])
  tp_tree <- length(pred_tree[(pred_tree == "yes") & (df_train$y == "yes")])
  tn_tree <- length(pred_tree[(pred_tree == "no") & (df_train$y == "no")])
  v_fpr_tree <- append(v_fpr_tree, fp_tree / (fp_tree + tn_tree))
  v_tpr_tree <- append(v_tpr_tree, tp_tree / (tp_tree + fn_tree))
}

# Plot ROC logistic regression
plot(v_fpr_log, v_tpr_log, type="l", xlim=c(0, 1), ylim=c(0, 1))
lines(c(0, 1), c(0, 1))
title("ROC Logistic Regression")

# Plot ROC decision tree
plot(v_fpr_tree, v_tpr_tree, type="l", xlim=c(0, 1), ylim=c(0, 1))
```

```r
lines(c(0, 1), c(0, 1))
title("ROC Decision Tree")

# Setup

library(magrittr)
library(dplyr)
library(caret)
library(ggplot2)

crime_df <- read.csv("communities.csv")

# Assignment 3.1

# Establish a scaler for the entire dataset
scaler_crime1 <- crime_df %>%
  select(-"ViolentCrimesPerPop") %>%
  preProcess()

# Rescale the entire dataset and create X_crime
X_crime <- crime_df %>%
  select(-"ViolentCrimesPerPop") %>%
  predict(scaler_crime1, .) %>%
  as.matrix()

# Compute sample covariance matrix for X_crime
S_crime <- (1/nrow(X_crime)) * t(X_crime) %*% X_crime

# Calculate eigenvalues for the sample covariance matrix
U_crime <- eigen(S_crime)

# Calculate the percentage of variance explained by each PC
var_explained <- (100*U_crime$values / sum(U_crime$values)) %>%
  round(digits = 3)

cat("Variance explained by PC1:", var_explained[1] %>% sum(), "%\n")

cat("Variance explained by PC2:", var_explained[2] %>% sum(), "%\n")

# 35 PC's needed to achieve at least 95% variance explained
cat("Variance explained by first 35 PC's:",
    var_explained[1:35] %>% sum(), "%\n")

# Assignment 3.2

PCA_2 <- princomp(crime_df %>% select(-"ViolentCrimesPerPop") %>% scale())

# U contains the PC loadings
# The eigenvectors basically (columnwise)
U <- PCA_2$loadings

# Plot a trace plot for
plot(U[,1], main="Fig 3.2. Traceplot, PC1")
```

```r
largest_contributors <- U[,1] %>% abs() %>% sort(decreasing = T) %>% names()
cat("Five largest contributors to PC1 are:\n")
cat(largest_contributors[1:5], sep = "\n")

# Assignment 3.2

# Get the datapoints coordinates expressed in PC1 and PC2
Z_crime_2 <- X_crime %*% U_crime$vectors[,c(1,2)] %>%
  as.data.frame() %>%
  cbind(ViolentCrimesPerPop = crime_df$ViolentCrimesPerPop)

# Plot data in terms of PC1 and PC2
ggplot(data = Z_crime_2,
       aes(x = Z_crime_2[,1],
           y = Z_crime_2[,2],
           col = ViolentCrimesPerPop)) +
  scale_color_gradient(low = "lightyellow", high = "black") +
  geom_point() +
  ggtitle("Fig 3.3. Crime data expressed in PC1 and PC2") +
  xlab("PC1 value") +
  ylab("PC2 value")

# Assignment 3.3

## Split data into train and test
n = nrow(crime_df)
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=crime_df[id,]
test=crime_df[-id,]


# Establish Scaling
scaling <- train %>% preProcess()

# Scale train and test data
train <- predict(scaling, train)
test <- predict(scaling, test)

# Train lin reg model
model1 <- lm(data = train,
             formula = ViolentCrimesPerPop ~ .)

# Compute train error (MSE)
train_mse <- mean((train$ViolentCrimesPerPop - model1$fitted.values)^2)

# Predict new responses for the test data
test_pred <- predict(model1,
                     test[,-which(names(test) == "ViolentCrimesPerPop")])
test_mse <- mean((test$ViolentCrimesPerPop - test_pred)^2)

cat("Train mse:", train_mse, "\n")
cat("Test mse:", test_mse, "\n")
```

```
## Assignment 3.4

# Set up data environment
train_noresponse <- train[,-which(names(train) == "ViolentCrimesPerPop")]
y_true_train <- train[,which(names(train) == "ViolentCrimesPerPop")]

test_noresponse <- test[,-which(names(test) == "ViolentCrimesPerPop")]
y_true_test <- test[,which(names(test) == "ViolentCrimesPerPop")]

train_errors <- c()
test_errors <- c()

# cost function
# data should NOT contain the response variable
# No intercept is included in the underlying model
cost_linreg <- function(theta){

  # Calculate train error
  y_pred_train <- as.matrix(train_noresponse) %*% theta
  train_cost <- mean((y_pred_train - y_true_train)^2)
  train_errors <<- c(train_errors, train_cost)

  # Calculate test error
  y_pred_test <- as.matrix(test_noresponse) %*% theta
  test_cost <- mean((y_pred_test - y_true_test)^2)
  test_errors <<- c(test_errors, test_cost)

  return(train_cost)
}

# Attempt to optimise our model
optim_object <- optim(rep(0, 100), method="BFGS", fn = cost_linreg, control = list(trace=T))

# Plot train and test (validation) errors
ylim <- c(min(train_errors[-c(1:500)], test_errors[-c(1:500)]), max(train_errors[-c(1:500)], test_errors
plot(test_errors[-c(1:500)], type="l", col="red", ylim = ylim,
     main = "Fig 3.4. Train and validation errors across iterations",
     xlab = "Iteration index",
     ylab = "Test (validation) errors")
points(train_errors[-c(1:500)], type="l", col="blue")

# Find the iteration number where the validation error is the lowest
optimal_iteration <- which(test_errors[-c(1:500)] == min(test_errors[-c(1:500)]))

# Print the resulting and train and test (validation) errors
cat("Train mse:", train_errors[optimal_iteration], "\n")
cat("Test mse:", test_errors[optimal_iteration], "\n")
```