Lab 2 Report

Group A11 - Christian Kammerer, Victor Guillo, Jakob Lindner

2024-12-06

Statement of Contribution

Assignment 1 was mainly contributed by Jakob Lindner. Christian Kammerer worked on Assignments 2, Victor Guillo on Assignment 3 and 4. Every student was responsible for the respective parts of the lab report.

Assignment 1

1.

The underlying linear model can be described with the following formula:

$$\text{Fat} = \beta_0 + \sum_{i=1}^{100} \left(\beta_i \text{Channel}_i \right) + \epsilon$$

The model is fit using the lm-method in R:

```
model_lr <- lm(Fat~.- Protein - Moisture - Sample, data=train)
```

A function is defined to calculate the errors. It take the true and predicted values as inputs and returns the mean sqaured error:

```
mean_squared_error <- function(true, predicted){
  error <- mean((true-predicted)^2)
  return(error)
}
```

The train error is very low, but the test error is very high. This strongly indicates that the model is highly overfitted and the quality of the model is bad.

```
## MSE on train data: 0.005709117
```

```
## MSE on test data: 722.4294
```

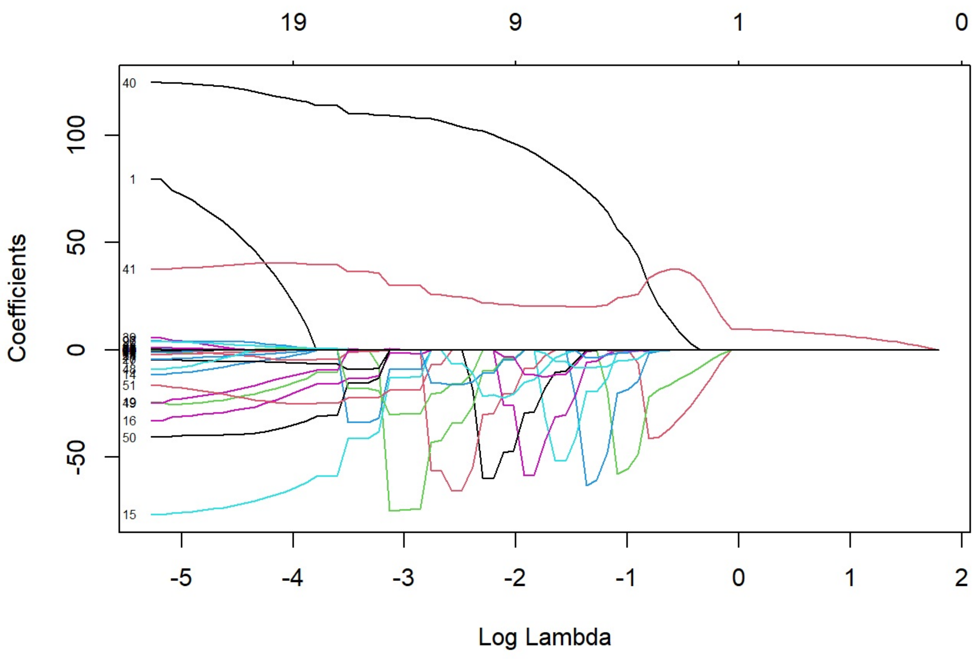
2.

The cost function to optimize for the LASSO regression looks like this:

$$\mathcal{L}(\beta) = \frac{1}{n} \sum_{i=1}^{n} \left(\text{Fat}_i - \beta_0 - \sum_{j=1}^{100} \beta_j \text{Channel}_{ij} \right)^2 + \lambda \sum_{j=1}^{100} |\beta_j|$$

3.

```
features <- train[,2:101]
target <- train$Fat
model_lasso <- glmnet(as.matrix(features), target, alpha=1, family="gaussian")
plot(model_lasso, xvar="lambda", label=TRUE)
```



The plot shows how the coefficients of the different channels get zero for higher lambda values. It shows that many of the 100 features do not have a high influence even for low lambda and go to zero quiet early (around log lambda = -4). After that, around 10 features are included, the respective value of the coefficients may increase and decrease until log lambda =-1, where only 5 features are left, with channel 41 being the feature that remains last.

```
# Find the lambda value that results in exactly three non-zero coefficients

num_features <- apply(coef(model_lasso) != 0, 2, sum) - 1  # Exclude the intercept
lambda_values <- model_lasso$lambda
selected_lambdas <- lambda_values[which(num_features == 3)]
```

The following three values for lambda can be chosen for a model with only three features:

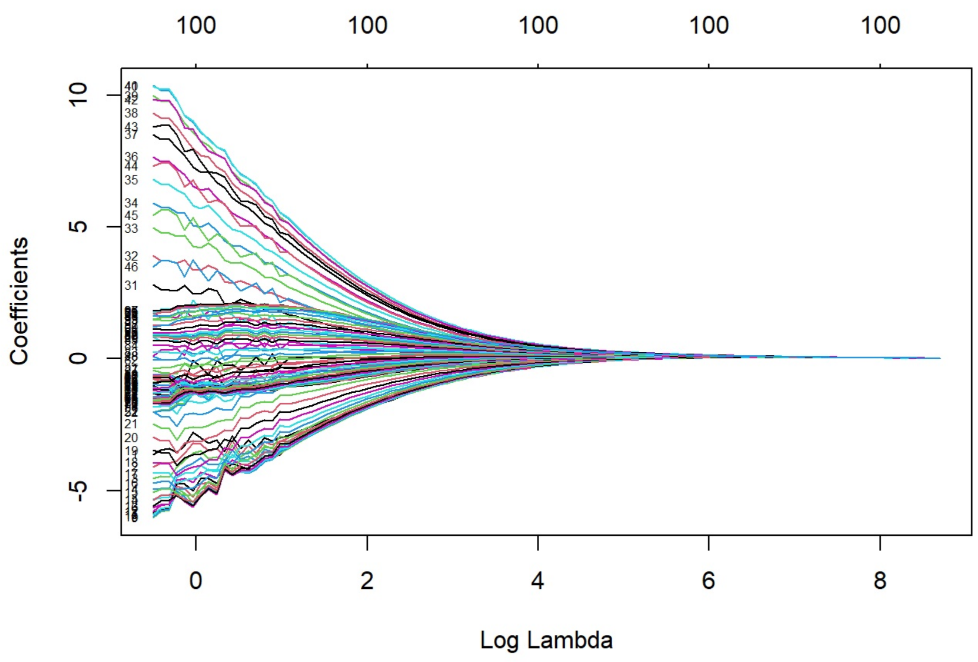
```
## Possible lambdas for exactly 3 features: 0.8530452 0.777263 0.7082131
```

```
## Corresponding log(lambda) values: -0.1589428 -0.2519765 -0.3450102
```

The log lambda values correspond with the visualization in the plot. The black line gets to zero at around -0.33 and the green line close to 0. This is the intervall, where exactly three coefficients are not zero.

4.

```
model_ridge <- glmnet(as.matrix(features), target, alpha=0, family="gaussian")
plot(model_ridge, xvar="lambda", label=TRUE)
```



```
num_features <- apply(coef(model_ridge) != 0, 2, sum) - 1

lambda_values <- model_ridge$lambda
selected_lambdas <- lambda_values[which(num_features == 3)]
```

The plot for ridge regression draws a different picture than the one for LASSO. Here all coefficients get smaller for higher lambda, but none ever gets zero, as the following output shows (“Number of features for every lambda value is 100). Therefore the LASSO regression simplifies the model, while Ridge regression only shrinks the coefficients.

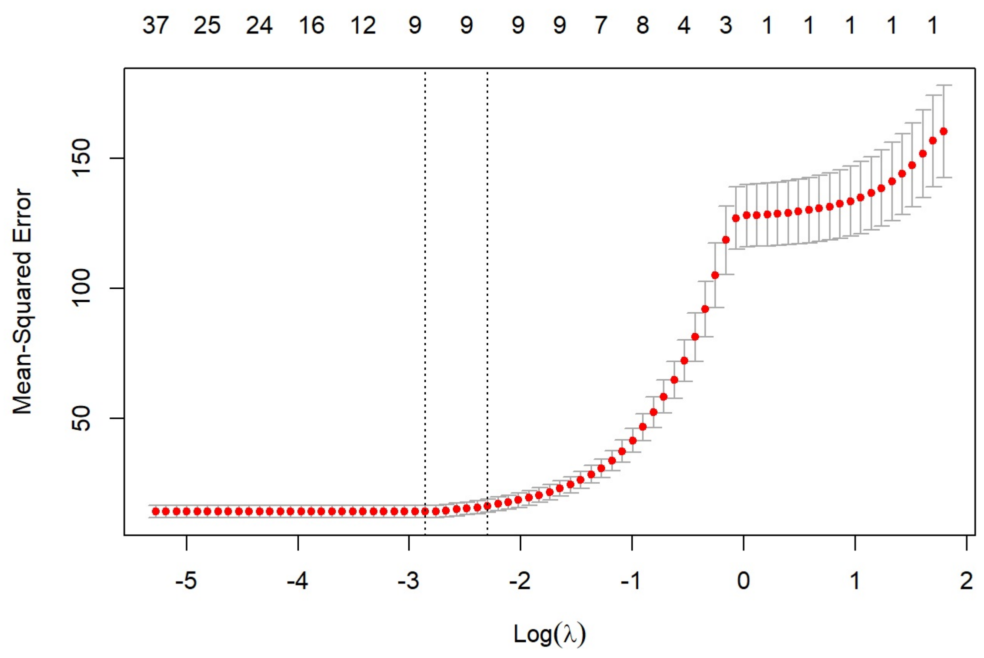
```
##
## Possible lambdas for exactly 3 features:
```

```
## Number of features for lambda values:  100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 1
00 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 10
0 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
```

5.

Cv.glmnet is used to perform cross validation on LASSO:

```
model_cv <- cv.glmnet(as.matrix(features), target, alpha=1, family="gaussian")
plot(model_cv)
```



The plot shows, that the MSE is constantly very low for small lambda until -2.8, then increases exponentially until almost 0 and then increases slower again.

The optimal lambda is stored in the model_cv objected and can be retrieved with $lambda.min. For this lambda 8 features are used. The optimal lambda does not seem to produce a significantly better result then for lambda -4, as the graph is constant for values smaller than the optimal lambda.

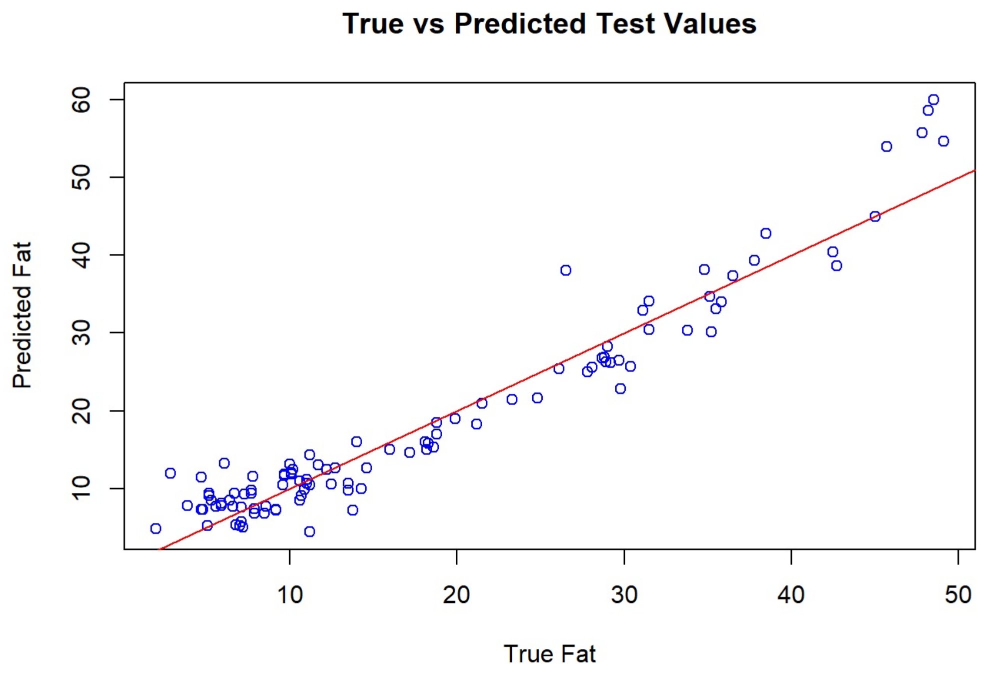
```
optimal_lambda <- model_cv$lambda.min
optimal_coefficients <- coef(model_cv, s = "lambda.min")
num_nonzero <- sum(optimal_coefficients != 0) - 1
```

```
## Optimal lambda: 0.05744535
```

```
## Number of selected features: 8
```

In the following the predictions and true values of the test data are plotted. The red line indicates the values where the prediction would be the same as the actual value.

```
y_test_pred <- predict(model_cv,newx = as.matrix(test[,2:101]), s="lambda.min")

plot(test$Fat, y_test_pred, main = "True vs Predicted Test Values",
     xlab = "True Fat", ylab = "Predicted Fat", col = "blue")
abline(0, 1, col = "red") # plot the line of y = x
```



Most points are very close to the red line, so the prediction quality is good. It is a big improvement compared to the linear regression in the first task, as the test error is significantly lower:

```
## Test error for LASSO: 13.67339
```

Assignment 2

Experiment regarding over- and underfitting

Task statement:
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.

Results from Experiment 1

Model	Missclassification.Error.Training	Missclassification.Error.Validation	Tree_Size
Default	0.1048	0.1093	2
Min Node Size = 7000	0.1142	0.1208	1
Min Deviance = 0.0005	0.0854	0.1170	115

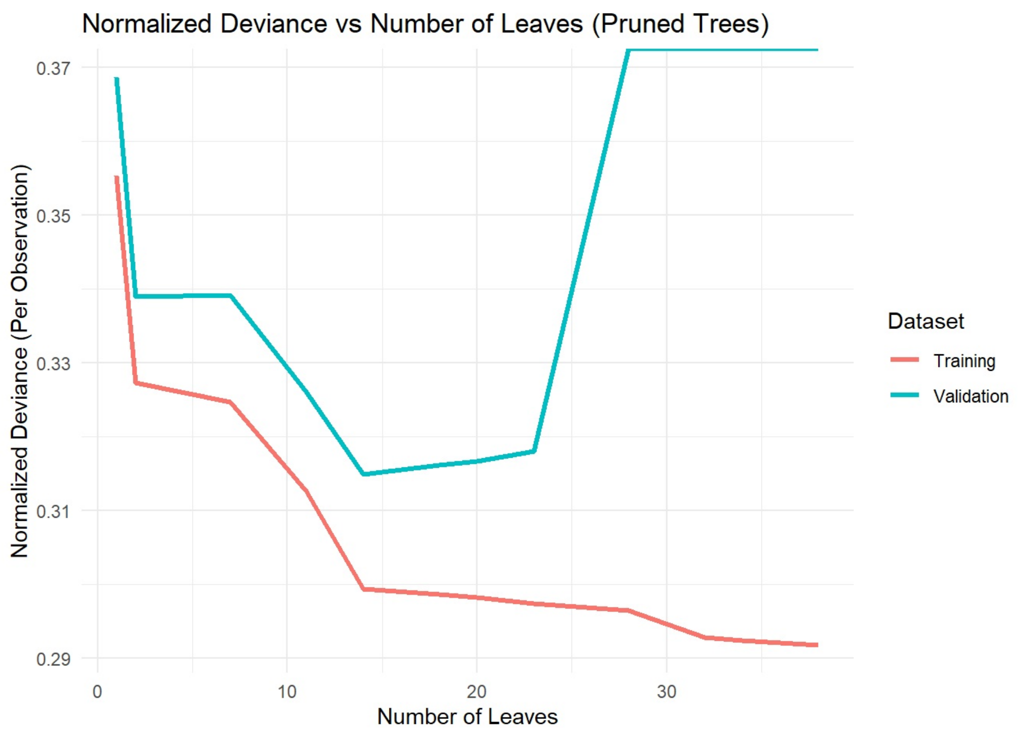
As we can observe in the table above, the model which performs best on the validation data set, is the one that is being trained using the default model parameters. This is because the two other models, are either underfitting, or overfitting the training data.

Why is model 2 underfitting?
Model 2 requires at least 7000 observations per leaf node. However, as the training data consists only of 2066 observations for the minority class, this prevents the model from performing any splits. Therefore the model consists of a single root node (see Tree Size column), which assigns every data point the label of the majority class. This can further be proven by the fact, that the number of minority class data points (2066) in the training set divided by the total number of observations (18084) in the training data equals to the missclassification error of model 2 on the training data (0.1142).

Why is model 3 overfitting?
The fact that model 3 performs significantly better on the training data, than on the validation data is indicative of overfitting. This is caused by us lowering the minimum deviance parameter from the default value of 0.01 to 0.0005. This causes the tree to branch out to an extreme degree, which is shown by the 115 leaf nodes that are present.

Determining the optimal tree depth

Task statement:
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).

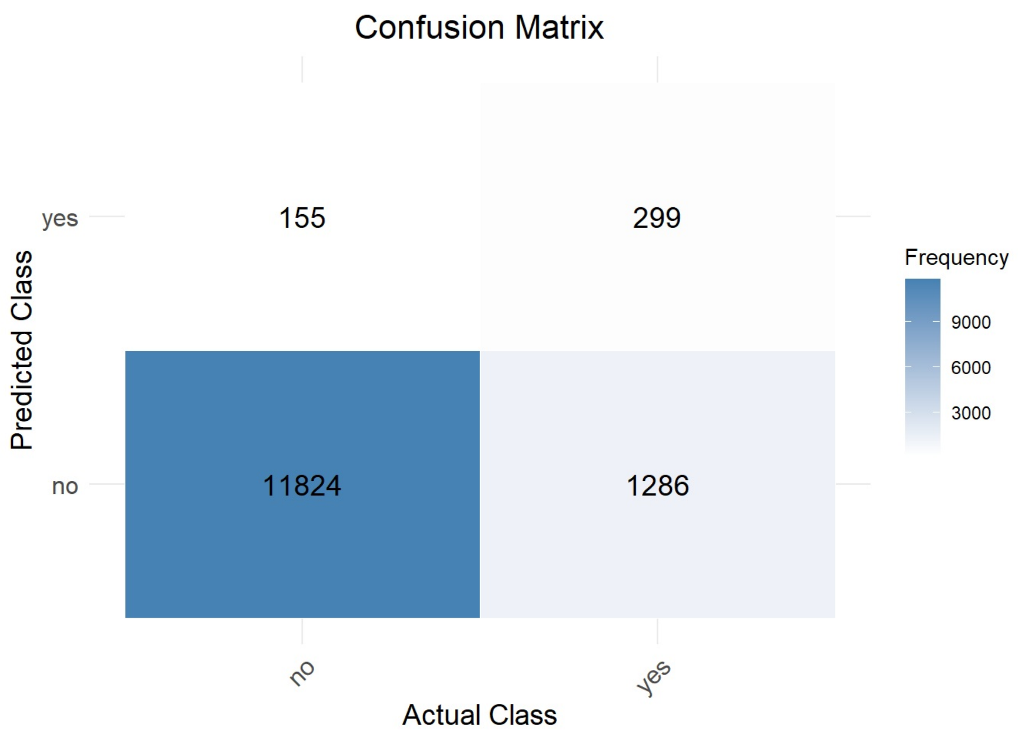


In this experiment we have trained the decision tree with the same parameters as the one in 2c) which we criticized for heavily overfitting the data. This time, we run an experiment to determine the optimal tree depth, by iteratively pruning the node, that least reduces the deviance of the tree. We then plot the normalized deviance of the trees of different depth, for both the training data set, as well as the validation data set.As observable in the graph, we can see that the deviance is decreasing for both data sets, up until a leaf count of 14. At which point the model starts overfitting, which is illustrated by a rise in deviance on the validation data set, while it continues to shrink on the training data. Therefore the optimal leaf count is 14.

The most important variables for the tree with the optimal leaf count are poutcome, month, pdays.

Estimating a confusion-matrix

Task statement:
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.



The accuracy of the model is 0.8937629 and the F1-Score is 0.293281. The confusion matrix shows that only around a fifth of minority classes are accurately predicted, this means despite having a relatively high accuracy, the model does not actually perform much better, than the tree consisting of merely the root node. Since there is a heavy class imbalance, it is reasonable to emphasize recall through a metric such as the F1-Score. If the class imbalance is even more extreme, one could even consider using the F2-Score, which places twice as much emphasize on recall.

Custom loss matrix

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

$$L = \begin{bmatrix} \text{Observed} & \text{Predicted} & \\ & \text{yes} & \text{no} \\ \text{yes} & 0 & 5 \\ \text{no} & 1 & 0 \end{bmatrix}$$

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

Confusingly, the changed loss function seems to have little to no impact on our model, making it even more conservative, than it was before. Believing I set up the loss matrix incorrectly, I toyed with different values, and no matter the set up, an altered loss function would always lead to more conservative predictions.

Accuracy is 0.8902978 and F1-Score is 0.2384852.

Assignment 3

Question 1

```
## Number of components to explain at least 95% of variance: 34
```

```
## Proportion of variance explained by first two components: 0.4224434
```

```
## Top 5 features contributing to the first principal component:
```

```
##      medFamInc      medIncome    PctKids2Par     pctWInvInc PctPopUnderPov
##      0.1832453      0.1819115      0.1755956      0.1749060      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)

These 5 features represents the socioeconomic gradient. It seems logical that poverty can leed to crime.

```
## Top 5 features contributing to the second principal component:
```

```
##  PctRecImmig10   PctRecImmig8   PctRecImmig5 PctRecentImmig PctForeignBorn
##      0.2192089      0.2191678      0.2170298      0.2145446      0.2126992
```

PctRecImmig10 : percent of population who have immigrated within the last 10 years (numeric - decimal)

PctRecImmig8 : percent of population who have immigrated within the last 8 years (numeric - decimal)

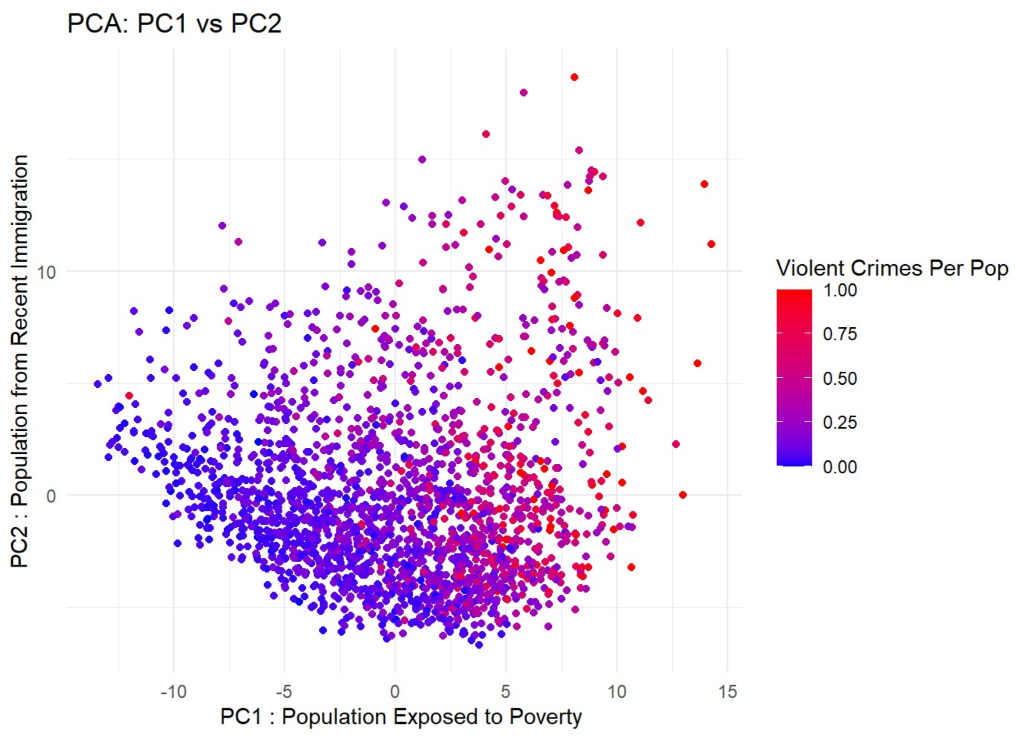
PctRecImmig5 : percent of population who have immigrated within the last 5 years (numeric - decimal)

PctRecentImmig : percent of population who have immigrated within the last 3 years (numeric - decimal)

PctForeignBorn: percent of people foreign born (numeric - decimal)

The second componant contains features regarding the pourcentage of the population that immigrated in the last 10 years.

Question 2



```
## Correlation of PC1 with ViolentCrimesPerPop: 0.6298838
```

```
## Correlation of PC2 with ViolentCrimesPerPop: 0.26818
```

PC1, the component representing a socioeconomic gradient dominated by poverty, has a stronger correlation with the target variable, violent crimes per population, compared to PC2. This suggests that poverty is a significant factor influencing crime rates.

The second principal component, related to recent immigration, shows a weaker correlation with crime but may still play a role indirectly, potentially through its association with socioeconomic challenges.

To deepen this analysis, we can examine the interaction between immigration and poverty within these populations to understand their combined impact on crime rates.

Question 3

```
## Training MSE: 0.01564263
```

```
## Test MSE: 0.01982713
```

The training and test MSE have a low and close value, the model doesnt seem to overfit

Question 4

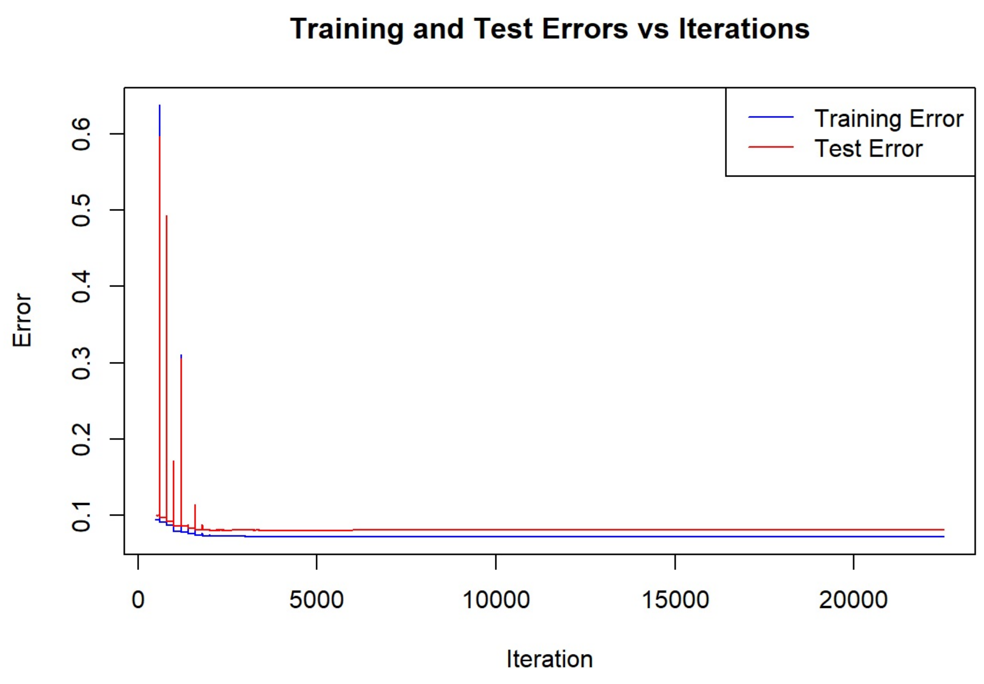
```
# Step 1: Define the cost function
linear_regression_cost <- function(theta, X, y) {
  # Compute predictions
  predictions <- X %*% theta

  # Compute MSE (mean squared error)
  mse <- mean((predictions - y)^2)

  return(mse)
}
```

```r
# Step 2: Optimize using BFGS with training data
train_test_errors <- function(train_X, train_y, test_X, test_y, max_iter = 2000) {
  # Initialize theta (parameter vector) to zeros
  initial_theta <- rep(0, ncol(train_X))

  # Store training and test errors for each iteration
  train_errors <- numeric(max_iter)
  test_errors <- numeric(max_iter)

  # Define a wrapper for the optim function to track errors
  cost_tracking <- function(theta) {
    # Compute training and test errors
    train_errors[curr_iter <<- curr_iter + 1] <<- linear_regression_cost(theta, train_X, train_y)
    test_errors[curr_iter] <<- linear_regression_cost(theta, test_X, test_y)

    # Return the cost for the optim function
    return(train_errors[curr_iter])
  }

  # Initialize iteration counter
  curr_iter <<- 0

  # Use optim to minimize the cost function
  optim_res <- optim(
    par = initial_theta,
    fn = cost_tracking,
    method = "BFGS",
    control = list(maxit = max_iter)
  )

  # Return training and test errors
  return(list(train_errors = train_errors, test_errors = test_errors, optim_res = optim_res))
}
```

```r
# Step 3: Prepare the data
# Use the scaled train/test datasets from Question 3
train_X <- as.matrix(train_data_scaled %>% select(-ViolentCrimesPerPop)) # Features
train_y <- train_data_scaled$ViolentCrimesPerPop                      # Target
test_X <- as.matrix(test_data_scaled %>% select(-ViolentCrimesPerPop))
test_y <- test_data_scaled$ViolentCrimesPerPop
```

```r
# Step 4: Run the optimization and track errors
results <- train_test_errors(train_X, train_y, test_X, test_y, max_iter = 2000)
```

```r
# Step 5: Plot the training and test errors
train_errors <- results$train_errors
test_errors <- results$test_errors

# Remove the first 500 iterations
plot_range <- 501:length(train_errors)
# Adjust the range of the y-axis to focus on the error values
error_range <- range(train_errors[plot_range], test_errors[plot_range])
```



```
## Optimal Iteration: 4432
```

```
## Training Error at Optimal Iteration: 0.07158554
```

```
## Test Error at Optimal Iteration: 0.07945343
```

The model generalizes well, achieving the best balance between minimizing the training error and avoiding overfitting to the training data. The optimal iteration (4432) is the point at which the test error is minimized, continuing optimization beyond this point does not benefit the test performance and risks overfitting.

Training Error Comparison:

The training error in Part 3 (0.0145) is lower than the one in Part 4 (0.07). This difference arises because: In Part 3, we used a standard regression approach with an intercept. In Part 4, the model optimizes parameters without an intercept, making it less flexible and potentially leading to higher error.

Test Error Comparison:

The test error in Part 3 (0.0204) is also lower than in Part 4 (0.078). This suggests that the model in Part 3 generalizes better to unseen data compared to the simpler model used in Part 4.

Model Complexity:

The model in Part 3 includes an intercept term and uses the standard lm() fitting process, which likely leads to better performance. The model in Part 4 omits the intercept and uses raw optimization, making it less precise and leading to higher errors.

Theory

What are the practical approaches for reducing the expected new data error, according to the book?

Practical approaches for reducing the expected new data error (E_new): To minimize the expected new data error (E_new), it is essential to simultaneously reduce the training error (E_train) and the generalization gap. Increasing the size of the training dataset is a practical approach since it typically decreases the generalization gap while slightly increasing E_train. Adjusting the model's flexibility—either increasing it if E_train is too high or decreasing it to reduce overfitting—is another key strategy. Cross-validation can help monitor the trade-off between E_train and E_new effectively. Pages 63-71

What important aspect should be considered when selecting minibatches, according to the book?

An important aspect to consider when selecting minibatches is subsampling, as explained on page 124 of the book. Subsampling involves selecting only a subset of the training data to compute the gradient at each iteration, which reduces computational cost while maintaining sufficient information for optimization. The book emphasizes that this method efficiently balances the use of all training data over multiple iterations without needing to process the entire dataset at once, making it suitable for large datasets.

Provide an example of modifications in a loss function and in data that can be done to take into account the data imbalance, according to the book

Example of modifications in a loss function and data to address data imbalance: An example of modifications to a loss function and data to address data imbalance is provided on page 101-102. The book explains that the loss function can be modified to assign different penalties to different types of errors. For instance, in a binary classification problem, a misclassification loss can be adjusted such that predicting the positive class (y=1) incorrectly is considered C times more severe than predicting the negative class (y=−1) incorrectly. Alternatively, the data itself can be adjusted by duplicating the positive class examples C times in the dataset, effectively balancing the data without modifying the loss function.

Appendix

```r
# Assignment 1

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

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

# 1 #

# train model using the channels as features
model_lr <- lm(Fat~.- Protein - Moisture - Sample, data=train)

# function for mean squared error
mean_squared_error <- function(true, predicted){
  error <- mean((true-predicted)^2)
  return(error)
}

# get the train error
train_error <- mean_squared_error(train$Fat, predict(model_lr))
train_error

# get the test error
test_error <- mean_squared_error(test$Fat, predict(model_lr, newdata = test))
test_error

# The model is highly overfitted, as the training error is very low, but the test error is very high


# 2 #
# \mathcal{L}(\beta) = \frac{1}{n} \sum_{i=1}^n \left( \text{Fat}_i - \beta_0 - \sum_{j=1}^{100} \beta_j \text{Ch
annel}_{ij} \right)^2 + \lambda \sum_{j=1}^{100} |\beta_j|


# 3 #
library(glmnet)
features <- train[,2:101]
target <- train$Fat
model_lasso <- glmnet(as.matrix(features), target, alpha=1, family="gaussian")
plot(model_lasso, xvar="lambda", label=TRUE)
# Only three features at log lambda, where channel4 goes to zero ->

# Find the lambda value that results in exactly three non-zero coefficients

num_features <- apply(coef(model_lasso) != 0, 2, sum) - 1  # Exclude the intercept
lambda_values <- model_lasso$lambda
selected_lambdas <- lambda_values[which(num_features == 3)]
cat("Possible lambdas for exactly 3 features:", selected_lambdas, "\n")
log(selected_lambdas)


# 4 #

model_ridge <- glmnet(as.matrix(features), target, alpha=0, family="gaussian")
plot(model_ridge, xvar="lambda", label=TRUE)

num_features <- apply(coef(model_ridge) != 0, 2, sum) - 1  # Exclude the intercept
cat("Number of features for lambda values: ", num_features)
lambda_values <- model_ridge$lambda
selected_lambdas <- lambda_values[which(num_features == 3)]
cat("\nPossible lambdas for exactly 3 features:", selected_lambdas, "\n")
log(selected_lambdas)
```

```r
# 5 #
model_cv <- cv.glmnet(as.matrix(features), target, alpha=1, family="gaussian")
plot(model_cv)
model_cv$lambda.min
log(model_cv$lambda.min)

optimal_coefficients <- coef(cv_lasso, s = "lambda.min")
num_nonzero <- sum(optimal_coefficients != 0) - 1  # Exclude intercept
cat("Number of selected features:", num_nonzero, "\n")
predict(model_cv,newx = as.matrix(test[,2:101]))

y_test_pred <- predict(model_cv,newx = as.matrix(test[,2:101]), s="lambda.min")
mean_squared_error(test$Fat, y_test_pred)


plot(test$Fat, y_test_pred, main = "True vs Predicted Test Values",
     xlab = "True Fat", ylab = "Predicted Fat", col = "blue")
abline(0, 1, col = "red")
correlation <- cor(test$Fat, y_test_pred)
correlation


# Assignment 2
# Function to evaluate model
evaluate_decision_tree <- function(train_data, valid_data, control_params, model_name) {
  # Train the decision tree with specified control parameters
  model <- rpart(y ~ ., data = train_data, method = "class", control = control_params)

  # Predict on training and validation data
  train_pred <- predict(model, train_data, type = "class")
  valid_pred <- predict(model, valid_data, type = "class")

  # Calculate misclassification rates
  train_mis <- mean(train_pred != train_data$y)
  valid_mis <- mean(valid_pred != valid_data$y)

  # Return results as a list
  list(
    Model = model_name,
    Train_Misclassification = train_mis,
    Validation_Misclassification = valid_mis,
    Model_Object = model
  )
}

control_default <- rpart.control()  # Default settings
control_node_size <- rpart.control(minbucket = 7000)  # Minimum node size = 7000
control_deviance <- rpart.control(cp = 0.0005)  # Minimum deviance = 0.0005


results <- list(
  evaluate_decision_tree(train, valid, control_default, "Default"),
  evaluate_decision_tree(train, valid, control_node_size, "Min Node Size = 7000"),
  evaluate_decision_tree(train, valid, control_deviance, "Min Deviance = 0.0005")
)

results_df <- do.call(rbind, lapply(results, function(x) data.frame(
  Model = x$Model,
  "Missclassification Error Training" = round(x$Train_Misclassification, 4),
  "Missclassification Error Validation" = round(x$Validation_Misclassification, 4)
)))


tree_sizes <- sapply(results, function(x) {
  model <- x$Model_Object
  length(unique(model$where))  # Count leaf nodes
})

# Add tree sizes to the results
results_df$Tree_Size <- tree_sizes
knitr::kable(results_df, caption = "Results from Experiment 1")


full_tree <- rpart(y ~ ., data = train, method = "class",
                   control = rpart.control(cp = 0.0005))

# Ensure cptable exists and is valid
if (is.null(full_tree$cptable) || nrow(full_tree$cptable) == 0) {
  stop("cptable is empty or invalid. Check the full_tree object.")
}

# Extract the complexity table
cptable <- full_tree$cptable

# Initialize storage for results
results <- data.frame(Leaves = integer(), Train_Deviance = numeric(), Valid_Deviance = numeric())

# Loop through pruning levels
for (i in 1:nrow(cptable)) {
  # Prune the tree to the current cp value
  pruned_tree <- prune(full_tree, cp = cptable[i, "CP"])

  # Count the number of leaves
  num_leaves <- length(unique(pruned_tree$where))

  # Skip trees with more than 50 leaves
  if (num_leaves > 50) {
    next
  }

  # Compute deviances
  train_probs <- predict(pruned_tree, train, type = "prob")
  train_dev <- -sum(log(train_probs[cbind(1:nrow(train), as.numeric(train$y))]))
  valid_probs <- predict(pruned_tree, valid, type = "prob")
  valid_dev <- -sum(log(valid_probs[cbind(1:nrow(valid), as.numeric(valid$y))]))

  results <- rbind(results, data.frame(Leaves = num_leaves, Train_Deviance = train_dev, Valid_Deviance = valid_de
v))
}

# Normalize deviances
results <- results %>%
  mutate(
    Train_Deviance_Normalized = Train_Deviance / nrow(train),
    Valid_Deviance_Normalized = Valid_Deviance / nrow(valid)
  )

# Create the plot
p <- ggplot(results, aes(x = Leaves)) +
  geom_line(aes(y = Train_Deviance_Normalized, color = "Training"), size = 1.2) +
  geom_line(aes(y = Valid_Deviance_Normalized, color = "Validation"), size = 1.2) +
  labs(
    title = "Normalized Deviance vs Number of Leaves (Pruned Trees)",
    x = "Number of Leaves",
    y = "Normalized Deviance (Per Observation)",
    color = "Dataset"
  ) +
  theme_minimal()

# Identify the optimal number of leaves
optimal_leaves <- results$Leaves[which.min(results$Valid_Deviance)]
optimal_cp <- cptable[which(results$Leaves == optimal_leaves), "CP"]
optimal_tree <- prune(full_tree, cp = optimal_cp)

# Extract variable importance
variable_importance <- optimal_tree$variable.importance
most_important_vars <- names(variable_importance)[order(-variable_importance)][1:3] # Top 3 variables
print(p)


evaluate_model <- function(optimal_tree, test_data) {
  predicted_probs <- predict(optimal_tree, test_data, type = "prob")

  predicted_classes <- ifelse(predicted_probs[, 2] >= 0.5, "yes", "no")
  predicted_classes <- factor(predicted_classes, levels = levels(test_data$y))


  confusion <- confusionMatrix(predicted_classes, test_data$y, positive = "yes")

  precision <- confusion$byClass["Precision"]
  recall <- confusion$byClass["Recall"]
  f1_score <- 2 * (precision * recall) / (precision + recall)

  list(
    Confusion_Matrix = confusion$table,
    Accuracy = confusion$overall["Accuracy"],
    F1_Score = f1_score
  )
}

results <- evaluate_model(optimal_tree, test)

visualize_confusion_matrix <- function(confusion_matrix) {
  cm_df <- as.data.frame(as.table(confusion_matrix))
  colnames(cm_df) <- c("Prediction", "Reference", "Frequency")

  ggplot(cm_df, aes(x = Reference, y = Prediction, fill = Frequency)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Frequency), color = "black", size = 5) +
    scale_fill_gradient(low = "white", high = "steelblue") +
    labs(
      title = "Confusion Matrix",
      x = "Actual Class",
      y = "Predicted Class"
    ) +
    theme_minimal() +
    theme(
      axis.text.x = element_text(angle = 45, hjust = 1),
      axis.text = element_text(size = 12),
      axis.title = element_text(size = 14),
      plot.title = element_text(size = 16, hjust = 0.5)
    )
}


confusion_plot <- visualize_confusion_matrix(results$Confusion_Matrix)
print(confusion_plot)

loss_matrix <- matrix(c(0, 5, 1, 0), nrow = 2, byrow = TRUE,
                      dimnames = list(c("yes", "no"), c("yes", "no")))


custom_tree <- rpart(
  y ~ .,
  data = train,
  method = "class",
  parms = list(loss = loss_matrix),
  control = rpart.control(cp = 0.0005)
)

# Prune to exactly 14 leaves
prune_to_leaves <- function(tree, target_leaves) {
  # Extract complexity table
  cptable <- tree$cptable

  # Find the cp value corresponding to the target number of leaves
  for (i in 1:nrow(cptable)) {
    pruned_tree <- prune(tree, cp = cptable[i, "CP"])
    num_leaves <- length(unique(pruned_tree$where))
    if (num_leaves <= target_leaves) {
      return(pruned_tree)
    }
  }
  stop("Could not find a pruning level with the desired number of leaves.")
}

# Prune the tree to 14 leaves
pruned_tree <- prune_to_leaves(custom_tree, 14)
results <- evaluate_model(custom_tree, test)
confusion_plot <- visualize_confusion_matrix(results$Confusion_Matrix)



##Assignment 3

### Question 1
communities <- read.csv("C:/Users/victo/OneDrive/Bureau/A1_SML/Machine Learning/Labs/Lab 2/communities.csv")

# Remove "state" column and scale all variables except 'ViolentCrimesPerPop'
communities_scaled <- communities %>%
  select(-state, -ViolentCrimesPerPop) %>%
  scale()


# Compute covariance matrix
cov_matrix <- cov(communities_scaled)
```

```r
# Perform PCA using eigen
eigen_decomp <- eigen(cov_matrix)

# Eigenvalues
eigen_values <- eigen_decomp$values

# Proportion of variance explained by each component
var_explained <- eigen_values / sum(eigen_values)

# Find the number of components needed to explain at least 95% variance
cum_var_explained <- cumsum(var_explained)
num_components <- which(cum_var_explained >= 0.95)[1]

# Proportion of variance explained by the first two principal components
first_two_var <- sum(var_explained[1:2])


# Print results
cat("Number of components to explain at least 95% of variance:", num_components, "\n")
cat("Proportion of variance explained by first two components:", first_two_var, "\n")

pca_result <- princomp(communities %>% select(-state, -ViolentCrimesPerPop), cor = TRUE)

# Extract loadings (weights for the principal components)
loadings <- pca_result$loadings

# Identify the top 5 features contributing to the first principal component
# Sort by absolute value of contributions
top_features <- abs(loadings[, 1]) %>%
  sort(decreasing = TRUE) %>%
  head(5)
top_features_names <- names(top_features)

cat("Top 5 features contributing to the first principal component:\n")
print(top_features)

# Identify the top 5 features contributing to the second principal component
# Sort by absolute value of contributions
top_features_2 <- abs(loadings[, 2]) %>%
  sort(decreasing = TRUE) %>%
  head(5)
top_features_names_2 <- names(top_features_2)

cat("Top 5 features contributing to the second principal component:\n")
print(top_features_2)


### Question 2


# Create a data frame with PC1, PC2, and ViolentCrimesPerPop
pca_scores <- as.data.frame(pca_result$scores)
pca_scores$ViolentCrimesPerPop <- communities$ViolentCrimesPerPop

# Plot PC1 vs PC2, colored by ViolentCrimesPerPop
ggplot(pca_scores, aes(x = Comp.1, y = Comp.2, color = ViolentCrimesPerPop)) +
  geom_point() +
  scale_color_gradient(low = "blue", high = "red") +
  labs(title = "PCA: PC1 vs PC2",
       x = "PC1 : Population Exposed to Poverty ",
       y = "PC2 : Population from Recent Immigration",
       color = "Violent Crimes Per Pop") +
  theme_minimal()

# Compute PCA scores
pca_scores <- as.data.frame(scale(communities_scaled) %*% eigen_decomp$vectors)
#
# Add the target variable for correlation analysis
 pca_scores$ViolentCrimesPerPop <- communities$ViolentCrimesPerPop

# Correlation of PCs with the target
cor_pc1 <- cor(pca_scores$V1, pca_scores$ViolentCrimesPerPop)
cor_pc2 <- cor(pca_scores$V2, pca_scores$ViolentCrimesPerPop)

cat("Correlation of PC1 with ViolentCrimesPerPop:", cor_pc1, "\n")
cat("Correlation of PC2 with ViolentCrimesPerPop:", cor_pc2, "\n")

# Step 1: Prepare the data
# Remove "state" and ensure ViolentCrimesPerPop is the target variable
features <- communities %>% select(-state)
target <- communities$ViolentCrimesPerPop

# Combine features and target into a single dataframe
data <- cbind(features, ViolentCrimesPerPop = target)

# Step 2: Split data into training and testing sets (50/50 split)
set.seed(12345)
train_index <- createDataPartition(data$ViolentCrimesPerPop, p = 0.5, list = FALSE)
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

#  Step 3: Scale features only (do not scale the target)
train_features <- scale(train_data %>% select(-ViolentCrimesPerPop))
test_features <- scale(test_data %>% select(-ViolentCrimesPerPop),
                       center = attr(train_features, "scaled:center"),
                       scale = attr(train_features, "scaled:scale"))

# Add back the target variable
train_data_scaled <- as.data.frame(train_features)
train_data_scaled$ViolentCrimesPerPop <- train_data$ViolentCrimesPerPop

test_data_scaled <- as.data.frame(test_features)
test_data_scaled$ViolentCrimesPerPop <- test_data$ViolentCrimesPerPop

# Step 4: Fit a linear regression model using training data
lm_model <- lm(ViolentCrimesPerPop ~ ., data = train_data_scaled)

# Step 5: Predict on training and testing data
train_predictions <- predict(lm_model, newdata = train_data_scaled)
test_predictions <- predict(lm_model, newdata = test_data_scaled)

# Step 6: Compute Mean Squared Error (MSE) for training and test sets
train_mse <- mean((train_predictions - train_data_scaled$ViolentCrimesPerPop)^2)
test_mse <- mean((test_predictions - test_data_scaled$ViolentCrimesPerPop)^2)



# Output results
cat("Training MSE:", train_mse, "\n")
cat("Test MSE:", test_mse, "\n")


## Question 4



# Step 1: Define the cost function
linear_regression_cost <- function(theta, X, y) {
  # Compute predictions
  predictions <- X %*% theta

  # Compute MSE (mean squared error)
  mse <- mean((predictions - y)^2)

  return(mse)
}


# Step 2: Optimize using BFGS with training data
train_test_errors <- function(train_X, train_y, test_X, test_y, max_iter = 2000) {
  # Initialize theta (parameter vector) to zeros
  initial_theta <- rep(0, ncol(train_X))

  # Store training and test errors for each iteration
  train_errors <- numeric(max_iter)
  test_errors <- numeric(max_iter)

  # Define a wrapper for the optim function to track errors
  cost_tracking <- function(theta) {
    # Compute training and test errors
    train_errors[curr_iter <<- curr_iter + 1] <<- linear_regression_cost(theta, train_X, train_y)
    test_errors[curr_iter] <<- linear_regression_cost(theta, test_X, test_y)

    # Return the cost for the optim function
    return(train_errors[curr_iter])
  }
  # Initialize iteration counter
  curr_iter <<- 0

  # Use optim to minimize the cost function
  optim_res <- optim(
    par = initial_theta,
    fn = cost_tracking,
    method = "BFGS",
    control = list(maxit = max_iter)
  )

  # Return training and test errors
  return(list(train_errors = train_errors, test_errors = test_errors, optim_res = optim_res))
}


# Step 3: Prepare the data
# Use the scaled train/test datasets from Question 3
train_X <- as.matrix(train_data_scaled %>% select(-ViolentCrimesPerPop)) # Features
train_y <- train_data_scaled$ViolentCrimesPerPop                      # Target
test_X <- as.matrix(test_data_scaled %>% select(-ViolentCrimesPerPop))
test_y <- test_data_scaled$ViolentCrimesPerPop


# Step 4: Run the optimization and track errors
results <- train_test_errors(train_X, train_y, test_X, test_y, max_iter = 2000)


# Step 5: Plot the training and test errors
train_errors <- results$train_errors
test_errors <- results$test_errors

# Remove the first 500 iterations
plot_range <- 501:length(train_errors)
# Adjust the range of the y-axis to focus on the error values
error_range <- range(train_errors[plot_range], test_errors[plot_range])

# Plot training and test errors
plot(plot_range, train_errors[plot_range], type = "l", col = "blue",
     xlab = "Iteration", ylab = "Error",
     main = "Training and Test Errors vs Iterations",
     ylim = error_range)
lines(plot_range, test_errors[plot_range], col = "red")
legend("topright", legend = c("Training Error", "Test Error"),
       col = c("blue", "red"), lty = 1)


# Identify the iteration with the minimum test error
optimal_iteration <- which.min(test_errors)
cat("Optimal Iteration:", optimal_iteration, "\n")
cat("Training Error at Optimal Iteration:", train_errors[optimal_iteration], "\n")
cat("Test Error at Optimal Iteration:", test_errors[optimal_iteration], "\n")
```

Processing math: 100%