Machine Learning - Lab 2, Block 1

Group A13: Arash Haratian, Connor Turner, Yi Hung Chen

2022-11-30

Statement of Contribution: *Assignment 1 was contributed by Yi Hung, Assignment 2 was contributed by Arash, and Assignment 3 was contributed by Connor. Each assignment was then discussed in detail with the group before piecing together each section of the final report.*

Assignment 1: Explicit Regularization

For this assignment, the task was to create a model that predicts the fat content of samples of meat given data from a near infrared absorbance spectrum. To do this, a linear regression model was built using various explicit regularization techniques.

The given data set was shuffled at random and split 50/50 into training and testing sets. Then, a basic linear regression model was trained on the training data, with all of the near infrared absorbance characteristics (channels) as features. Below is the underlying probabilistic model of the linear regression:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_{100} x_{100} + \epsilon$$
$$y|x \sim N(\theta^T x, \sigma^2)$$

where:

$$y = Fat$$
$$\theta = \theta_0, ..., \theta_{100}$$
$$x = \{1, Channel_1, Channel_2, ..., Channel_{100}\}$$
$$\epsilon \ is \ the \ error \ term$$

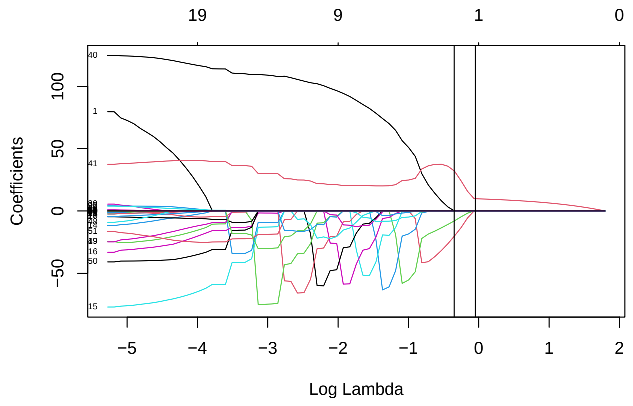
For this basic model, the training error was 0.0057, and the testing error was 722.4294. The large mean squared error (MSE) on the testing data indicates that the many features of the linear regression model have led to overfitting. To address this issue, a variety of explicit regularization techniques were used to simplify the model.

First, a LASSO regression model was used to improve upon the basic regression. To do so, the following cost function was optimized to find the optimal regression coefficients, $\theta^{\hat{lasso}}$.

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

This LASSO model was fitted to the training data, and the dependency of the regression coefficients on $log(\lambda)$ is shown below in Figure 1.

Figure 1: LASSO Regression Coefficients Over log(lambda)

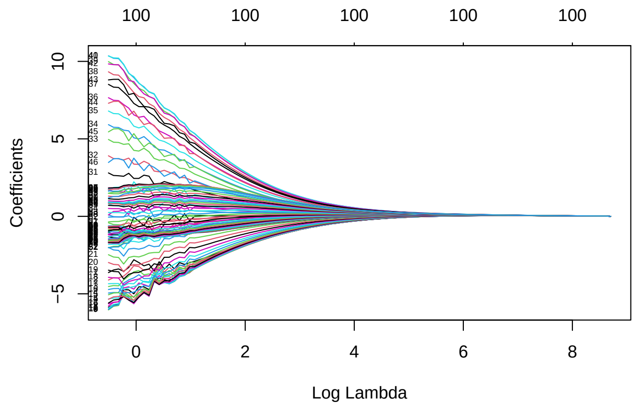


.
The plot above shows that the LASSO model will drop some coefficients as $log(\lambda)$ increases, as the penalty factor pushes their value to 0. Interestingly, some coefficients that were dropped reappeared as $log(\lambda)$ increased. But in general, the larger $log(\lambda)$ is, the fewer features are actually contained in the model. For example, a LASSO regression with a λ value around 0.9512 to 0.7047 would yield a model with only three features.

Note: This range was chosen by directly observing the plot (the range is shown by the vertical lines in Figure 1). Additionally, by using default λ values in `glmnet`*, it was calculated that λ = 0.853, 0.7773, 0.7082 gives coefficients for only three features. However, as the plot shows, the λ values should be in a range, not distinct values.*

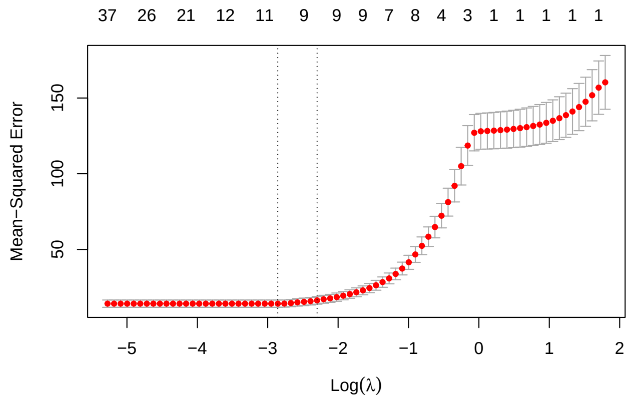
Second, a ridge regression model was fitted to the training data, and the results were compared to the output from the LASSO regression. Figure 2 below shows the value of the coefficients in the ridge regression plotted against $log(\lambda)$. Compared to the LASSO regression, the penalty term of ridge regression does not "collapse" to zero. Also, the coefficient value of ridge regression scales slower when lambda increases compared to the LASSO regression.

Figure 2: Ridge Regression Coefficients Over log(lambda)



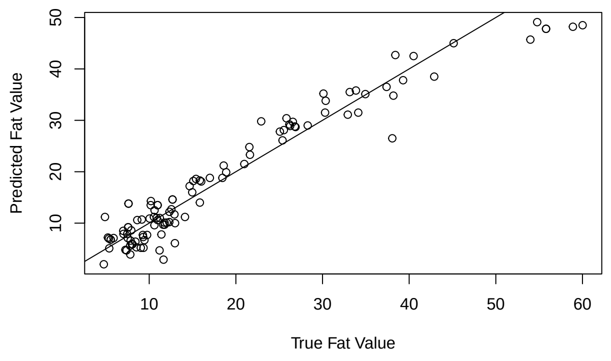
.
Using the `cv.glmnet` function in R, a cross-validation analysis was conducted for the LASSO model. In this cross-validation, the number of folds was not specified; instead, the default number of folds was used to compute the cross-validation. As shown in Figure 3 below, the MSE started increasing dramatically as $log(\lambda)$ approached -2, which means that the model became less predictive and less usable. This may be a result of too many coefficients being ditched by the LASSO regression as $log(\lambda)$ increased.

Figure 3: LASSO Cross−Validation Score Over log(lambda)



.
The optimal model was obtained when λ =0.0574 and 7 variables were chosen. According to Figure 3, the optimal λ ($log(\lambda) = -2.8569$) is not statistically significantly better than $log(\lambda) = -4$, as they have a similar MSE. The scatter plot in Figure 4 shows the output of this optimal model on the testing data compared to the true fat content of the meat, with the reference line representing perfect 1:1 correlation.

Figure 4: Prediction of Fat Content Using Optimal LASSO Model



.
Based on the plot above, it appears that the LASSO model provides a fairly reliable prediction of fat content. The testing error for this model was 13.2998, which is significantly better the basic linear regression model derived previously. Therefore, this model is much more effective for this purpose.

Assignment 2: Decision Trees and Logistic Regression

For this assignment, the task was to build a model that predicts the effectiveness of direct marketing campaigns for a Portuguese banking institution. The dataset consists of 16 different variables about the potential client and how often they were contacted - as well as the target variable indicating whether or not the potential client made a deposit. Because the target variable is a classification, this model will be trained using a mixture of decision trees and logistic regression. The dataset was shuffled randomly and split 40/30/30 into training, validation, and testing sets, and all independent variables were used as features except for the duration of the last contact.

Note: even though last contact duration is highly correlated with the target variable, one cannot know the duration of a call before it is made. Thus, it is counterintuitive to use it for training a predictive model.

After the dataset was split, the training data were used to train three different decision trees with three different settings:

The first model is a decision tree with default settings of the `tree` function in R. The default values are as follows:

$$mincut = 5$$

$$minsize = 10$$

$$mindev = 0.01$$

The training and testing errors for this model are 0.1048 and 0.1093, respectively.

The second model is a decision tree with a constraint on the size of each node; in this case, the smallest allowed node size is 7,000. The training and testing errors for this model are 0.1048 and 0.1093, respectively.

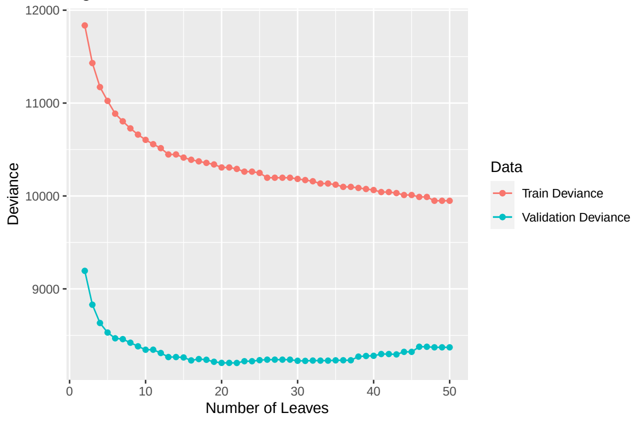
For the third model, the minimum deviance is set to 0.0005. The training and testing errors for the this model are 0.094 and 0.1119, respectively. The table below compares the three decision trees:

Values	First Model	Second Model	Third Model
Size of Trees	6	5	122
Number of Features Used	4	3	13
Training Error	0.1048	0.1048	0.094
Testing Error	0.1093	0.1093	0.1119

.
Based on the information in the table above, the first model and second model produce quite similar results, as they have about the same error rate and the second model only uses one less tree branch and one less feature in its model. In contrast, the third model is overfitted on the training data and it is too complex; as a result, it may preform poorly for predicting new values. Overall, the second model is more proper for interpreting the structure of the tree, but for predicting new values, the first model may preform better, as it is more complex compared to the second model.

However, this does not mean that the third model cannot still be effective, as we can prune it to create a more optimal model that captures the general structure better than the other two models. In other words, by pruning this fully-grown tree, the overfitting observed earlier can be mitigated. Figure 5 below shows the training and testing deviance for the model based on the number of leaves:

Figure 5: Deviance vs Size of Decision Tree



As shown in the plot above, the deviance for the training data decreases as the number of leaves increases (the model gets more complex). However, the deviance for the validation data decreases at first, but then starts to increase once the model reaches a certain complexity. Thus, as the model increases in complexity beyond a certain level, the model will become more and more overfitted.

Note: The deviance for the training data is higher than deviance for the validation data because deviance is proportional to the number of observations in the dataset, and the testing data contains more observations.

Based on the model described above, the optimal number of leaves appears to be 22. The most important variables in the optimal tree are as follows: outcome of the previous campaign, month of last contact, contact communication type, number of days since last contact, age of the potential client, day of the week of last contact, bank balance, whether or not the potential client has a housing loan, and what type of job the potential client has. Based on the structure of the tree, the first variable that has been used to split the data was the outcome of the previous marketing campaign (`poutcome`). Additionally, the month of last contact (`month`) was used frequently to split the data.

To evaluate the performance of the optimal tree, a confusion matrix was estimated for the testing data, and the accuracy and F1-score were calculated and analyzed. F1-score is a better metric to measure the performance of this model, as the target variable in this case is unbalanced. The table below shows the confusion matrix for the optimal tree on the testing data:

	no	yes
no	11872	107
yes	1371	214

This model had an accuracy of 89.1%, which indicates that the model is quite accurate in predicting new values. However, since the target variable has more `no` values than `yes` values, a model that predicts all the values as `no` would yield a high accuracy as well. A good model is a model that can predict `yes` more accurately in this dataset, and the F1-score is calculated to measure this. For this model, the F1-score is 0.2246. This is quite low, and it indicates that the model is not preforming well when it comes to predicting true positives.

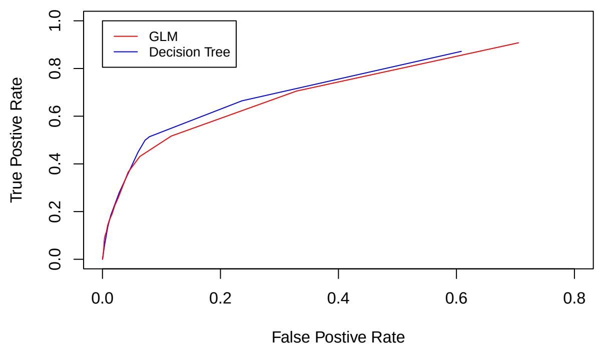
One way to increase the performance of the model is to use a loss matrix that predicts the values based on their probabilities (for example, using .2 as the threshold instead of using 0.5). In other words, we are penalizing the model if it predicts positives incorrectly. The confusion matrix for the testing data using the loss matrix is shown in the table below:

	0	1
no	11030	949
yes	771	814

The new F1-score is 0.4863, which is just over a 100% increase over the previous model. This means that this model is more accurate at predicting true positives; however, the overall accuracy of the model decreased to 87.32%.

One way to show the effectiveness of the decision tree model is to compare it with a logistic regression model trained on the training data. Figure 6 below shows the receiver operating characteristic (ROC) curves for both the optimal decision tree model and the logistic regression model.

Figure 6: ROC Curves for Decision Tree and Logistic Regression Mod



.

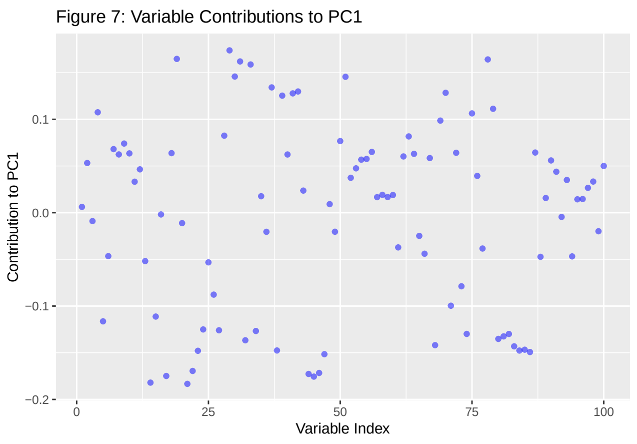
Based on the ROC curves shown above, both models perform almost identically, but the decision tree is slightly better. For the same value of π and the same false positive rate, the decision tree has a slightly higher true positive rate. However, a precision-recall curve may be a better choice to compare these two models, as the target data is imbalanced. The model that has the higher recall for the same value of precision is more accurate at predicting positive values.

Assignment 3: Principal Components and Implicit Regularization

For this assignment, the task was to create a model that predicts crime rates per capita in a community given 100 different variables about the community. As with any dataset that contains a large number of variables, the initial challenge is to find the most relevant variables and remove irrelevant ones. This helps to lower the complexity of the model, and therefore lower the risk of overfitting.

To do this, a principal components analysis was conducted using the given dataset. All of the variables except for the dependent variable (violent crimes per 100,000 people) were scaled appropriately, and the data were used to construct a covariance matrix, $\frac{1}{n}X^TX$. By taking the eigenvectors and eigenvalues of that matrix, these 100 variables were converted into 100 principal components (PCs), which are composed of linear combinations of the other variables. Of these, PC1 explains the most of the variation, and PC100 explains the least. By analyzing the percentage of variation explained by each principal component, it is revealed that the first 34 components explain 95% of the variation, and the first two components explain 25.02% and 16.94%, respectively.

The trace plot in Figure 7 below shows the contribution of each variable in the dataset to PC1, calculated using the `princomp` function in R.



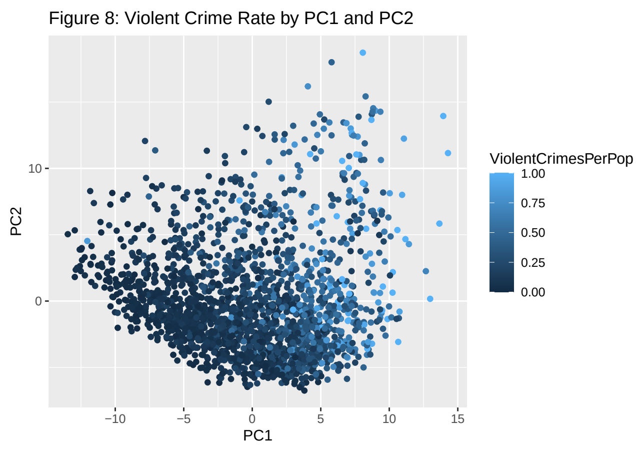
.

At first glance, there seems to be no pattern to the data. Some variables have a significant impact on PC1, and others have very little impact – but the dispersion is quite random. However, when one looks into which variables have the greatest impact (in terms of absolute value), the patterns become quite clear. Below are the five variables with the largest contributions to PC1:

	x
medFamInc	0.1833
medIncome	0.1820
PctKids2Par	0.1755
pctWInvInc	0.1749
PctPopUnderPov	0.1738

According to the table, the top five contributors to PC1 are median family income, median income, the percentage of kids with two parents in the household, the percentage of households with investment income, and the percentage of the population under the poverty line. Four of these five are indicators of income, wealth, and poverty, which is unsurprising given that these factors have historically been some of the best predictors of crime rates in a given neighborhood.

Figure 8 below is a scatterplot, where PC1 is used as the x-coordinate and PC2 is used as the y-coordinate. The color of each point varies along with the crime rate per 100,000 people, where darker blue indicates a lower crime rate and lighter blue indicates a higher crime rate.



.

The figure above shows that there appears to be a relationship between PC1 and violent crime rate. As PC1 increases, the points in the plot tend to get lighter, and vice versa. However, there does not appear to be a strong relationship between PC2 and crime rate, as a similar trend does not appear along the y-axis.

The next part of the task was to train a linear regression model and optimize the resulting coefficient estimates. The first step was to shuffle the dataset, split it 50/50 into training and testing sets, and scale each set according to the training data. Once that was completed, a basic linear regression model was created using `ViolentCrimesPerPop` as the dependent variable and all 100 variables as features. The MSE for the training and testing data is shown below:

$$Training\ Error = 0.2752071$$

$$Testing\ Error = 0.4248011$$

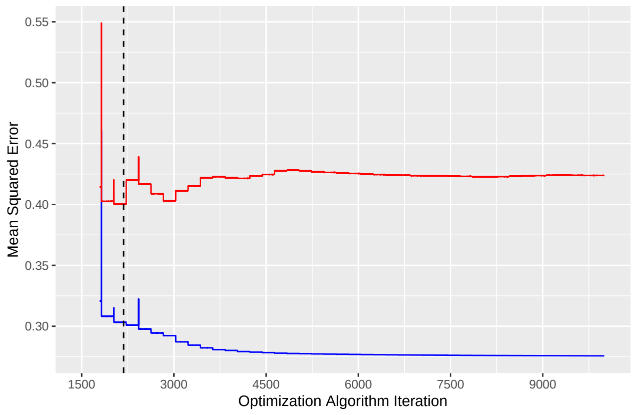
There is quite a large difference between the testing error and the training error for the basic linear regression model, indicating that the model may be overfitted. To address this issue, implicit regularization was used to find optimized regression coefficient estimates that minimized the testing error.

Using the data from the training and testing sets, the `optim` function was used (with method "BFGS") to minimize the training MSE. The vector of regression coefficients, θ, served as the optimizing variables, with each coefficient starting with an initial value of 0. While the function was running, the training and testing MSE were computed and recorded for each iteration of the algorithm (and by extension, each θ).

In total, the optimization algorithm underwent 20,114 iterations before converging. However, the "optimal" θ given by the algorithm is not the one that will be used for the model. Because the algorithm was designed to minimize the *training* MSE, the resulting model from these θ values would be overfitted. For the purposes of this assignment, the optimal values for θ are the ones that minimize the *testing* MSE. This can be found by going through each iteration of the algorithm and finding the values for θ that minimize the training error.

Figure 9 below shows the training and testing errors for the model over iterations 1,800-10,000. The first 1,800 values were removed from the plot, as they prevent the dependencies from being shown in the plot.

Figure 9: Training and Testing MSE During Optimization



.
Figure 9 shows that the testing error is minimized at iteration 2,182 (represented by a dashed line above). At this point, the training error is 0.3033 and the testing error is 0.4002. This not only improves upon the previous testing error, but it also significantly closes the gap between the training and testing error compared to the previous values. Therefore, the model in this iteration of the optimization algorithm would likely produce the best results when it comes to predicting violent crime rate from new data.

Appendix 1: Code for Assignment 1

```r
# ==Q1==
rm(list = ls())
library(glmnet)
# Split and Prepare Data
# --------------------------------------------------
# Split the Data Into Training and Testing Data
# (50/50):
tecator <- read.csv("tecator.csv")
set.seed(12345)
n = nrow(tecator)
id = sample(1:n, floor(n * 0.5))
train = tecator[id, ]
test = tecator[-id, ]

# =====Fit Linear Regression=====
lm_tecator <- lm(formula = Fat ~ . - Protein - Moisture -
    Sample, data = train)

train_fitvalues <- lm_tecator$fitted.values
test_predict <- predict(lm_tecator, test)


MSE_train <- mean(lm_tecator$residuals^2)
MSE_test <- mean((test$Fat - test_predict)^2)


# ==Q2,Q3== =====Lasso===== Make argument in the
# format that glmnet can use (data matrix for x)
x_name <- colnames(tecator)[-1]
x_name <- x_name[-102]
x_name <- x_name[-102]
x_name <- x_name[-101]
x <- data.matrix(train[, x_name])
y <- train$Fat

# Train Lasso
lasso_tecator <- glmnet(x = x, y = y, alpha = 1)

plot(lasso_tecator, xvar = "lambda", label = TRUE, main = "Figure 1:\n Dependence of Lasso Regression c
    abline(v = -0.05) + abline(v = -0.35)

# pick lambda for 3 features
coef_matrix <- as.matrix(coef(lasso_tecator))
coef_matrix <- coef_matrix != 0
coef_matrix <- colSums(coef_matrix)
lamda_index <- which(coef_matrix == 4)  # we use 4 because there is alway intercept with in the matric
lambda_3features <- lasso_tecator$lambda[lamda_index]  # the lamda for 3 features


# ==Q4== =====Ridge======
ridge_tecator <- glmnet(x, y, alpha = 0)
plot(ridge_tecator, xvar = "lambda", label = TRUE, main = "Figure 2:\n Dependence of Ridge Regression c
# ==Q5==
cv_lasso_tecator <- cv.glmnet(x = x, y = y, alpha = 1)  #Use cv.glmnet to do cross validation

plot(cv_lasso_tecator, main = "Figure 3:\n Dependence of the Lasso CV score on log lambda\n\n")

best_lambda <- cv_lasso_tecator$lambda.min
best_model <- glmnet(x, y, alpha = 1, lambda = best_lambda)
variables <- coef(best_model)
variables <- as.vector(variables != 0)
coef_index <- which(match(variables, TRUE) == 1)
coef_index <- coef_index[-1]

coef_get_select <- rownames(coef(best_model))[coef_index]
```

Appendix 2: Code for Assignment 2

```r
## Attaching Packages
library(tidyverse)
library(tree)

# loading & splitting the data
bank_full <- read.csv2("./data/bank-full.csv")
bank_full <- bank_full %>%
    select(!duration) %>%
    mutate(across(where(is.character), as.factor))

n <- nrow(bank_full)
set.seed(12345)
train_idx <- sample(seq_len(n), floor(n * 0.4))
train_data <- bank_full[train_idx, ]
remainder_idx <- setdiff(seq_len(n), train_idx)
set.seed(12345)
valid_idx <- sample(remainder_idx, floor(n * 0.3))
valid_data <- bank_full[valid_idx, ]
test_idx <- setdiff(remainder_idx, valid_idx)
test_data <- bank_full[test_idx, ]

# First Decision Tree:
tree_default <- tree(y ~ ., train_data)
y_hat_train <- predict(tree_default, train_data, type = "class")
cm_train <- table(train_data$y, y_hat_train)
error_train_a <- 1 - (sum(diag(cm_train))/nrow(train_data))
# mean(y_hat_train != train_data$y)
y_hat_train <- predict(tree_default, valid_data, type = "class")
cm_valid <- table(valid_data$y, y_hat_train)
error_test_a <- 1 - (sum(diag(cm_valid))/nrow(valid_data))

# Second Decision Tree:
tree_minsize <- tree(y ~ ., train_data, control = tree.control(nrow(train_data),
    minsize = 7000))
y_hat_train <- predict(tree_minsize, train_data, type = "class")
cm_train <- table(train_data$y, y_hat_train)
error_train_b <- 1 - (sum(diag(cm_train))/nrow(train_data))
# mean(y_hat_train != train_data$y)
y_hat_train <- predict(tree_minsize, valid_data, type = "class")
cm_valid <- table(valid_data$y, y_hat_train)
error_test_b <- 1 - (sum(diag(cm_valid))/nrow(valid_data))

# Third Decision Tree:
tree_mindev <- tree(y ~ ., train_data, control = tree.control(nrow(train_data),
    mindev = 5e-04))
y_hat_train <- predict(tree_mindev, train_data, type = "class")
cm_train <- table(train_data$y, y_hat_train)
error_train_c <- 1 - (sum(diag(cm_train))/nrow(train_data))
# mean(y_hat_train != train_data$y)
y_hat_train <- predict(tree_mindev, valid_data, type = "class")
cm_valid <- table(valid_data$y, y_hat_train)
error_test_c <- 1 - (sum(diag(cm_valid))/nrow(valid_data))
```

```r
# pruning the third DT
train_dev <- vector("numeric", 50)
valid_dev <- vector("numeric", 50)

for (num_leaves in 2:50) {
    prune_tree <- prune.tree(tree_mindev, best = num_leaves)
    valid_tree <- predict(prune_tree, newdata = valid_data,
        type = "tree")
    train_dev[num_leaves] <- deviance(prune_tree)
    valid_dev[num_leaves] <- deviance(valid_tree)
}

data.frame(num_of_leaves = 2:50, train = train_dev[2:50],
    valid = valid_dev[2:50]) %>%
    ggplot() + geom_line(aes(num_of_leaves, train, color = "Train Deviance")) +
    geom_point(aes(num_of_leaves, train, color = "Train Deviance")) +
    geom_line(aes(num_of_leaves, valid, color = "Validation Deviance")) +
    geom_point(aes(num_of_leaves, valid, color = "Validation Deviance")) +
    labs(x = "Number of Leaves", y = "deviance", color = "dataset")
best_num_leaves <- which.min(valid_dev[2:50]) + 1
optimal_tree <- prune.tree(tree_mindev, best = best_num_leaves)

# structure of tree
optimal_tree

# Evaluating the optimal tree
y_hat <- predict(optimal_tree, newdata = test_data, type = "class")

cm_test <- table(test_data$y, y_hat)
acc <- sum(diag(cm_test))/nrow(test_data)

TP <- cm_test[2, 2]
FP <- cm_test[1, 2]
FN <- cm_test[2, 1]

F1_score <- TP/(TP + 0.5 * (FP + FN))

knitr::kable(cm_test, label = "the confusion matrix of optimal tree on test data")

# Evaluating the optimal tree with loss matrix
y_hat <- predict(optimal_tree, newdata = test_data)
y_hat <- as.factor(ifelse(y_hat[, 2]/y_hat[, 1] > 1/5, 1,
    0))


cm_test <- table(test_data$y, y_hat)
acc <- sum(diag(cm_test))/nrow(test_data)

TP <- cm_test[2, 2]
FP <- cm_test[1, 2]
FN <- cm_test[2, 1]

F1_score <- TP/(TP + 0.5 * (FP + FN))
```

```r
knitr::kable(cm_test, label = "the confusion matrix of optimal tree on test data with loss matrix")

# ROC curve
y_hat_prob_tree <- predict(optimal_tree, test_data)
y_hat_prob_tree <- y_hat_prob_tree[, "yes"]

glm_model <- glm(y ~ ., family = "binomial", data = train_data)
y_hat_prob_glm <- predict(glm_model, test_data, type = "response")

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

ROC_tree <- matrix(nrow = length(pis), ncol = 2)
ROC_glm <- matrix(nrow = length(pis), ncol = 2)

for (i in seq_along(pis)) {

    y_hat_tree <- factor(ifelse(y_hat_prob_tree > pis[i],
        "yes", "no"), levels = c("no", "yes"))
    cm_test <- table(test_data$y, y_hat_tree)
    TP <- cm_test[2, 2]
    FP <- cm_test[1, 2]
    ROC_tree[i, 1] <- FP/sum(cm_test[1, ])
    ROC_tree[i, 2] <- TP/sum(cm_test[2, ])

    y_hat_glm <- factor(ifelse(y_hat_prob_glm > pis[i],
        "yes", "no"), levels = c("no", "yes"))
    cm_test <- table(test_data$y, y_hat_glm)
    TP <- cm_test[2, 2]
    FP <- cm_test[1, 2]
    ROC_glm[i, 1] <- FP/sum(cm_test[1, ])
    ROC_glm[i, 2] <- TP/sum(cm_test[2, ])

}

plot(ROC_tree, type = "l", col = "blue", xlim = c(0, 0.8),
    ylim = c(0, 1), xlab = "FPR", ylab = "TPR")
lines(ROC_glm, col = "red")
legend(0, 1, legend = c("GLM", "Decision Tree"), col = c("red",
    "blue"), lty = 1, cex = 0.8)
```

Appendix 3: Code for Assignment 3

```r
library(caret)
library(ggplot2)
library(tidyr)
setwd("Desktop/Statistics and Machine Learning/Machine Learning/Labs/Lab 2")
communities <- read.csv("communities.csv")


# Part 1: Initial PCA Analysis
# -------------------------------------------

# Scale variable data
n = nrow(communities)
scaledata <- communities
pcascaler <- preProcess(communities[-101])
scaledata <- predict(pcascaler, scaledata)

# Find eigenvectors, eigenvalues, and coordinates
xmatrix <- as.matrix(scaledata[-101])
covmatrix <- (1/n) * (t(xmatrix) %*% xmatrix)
eigenvectors <- eigen(covmatrix)$vectors
eigenvalues <- eigen(covmatrix)$values
coordinates <- xmatrix %*% eigenvectors

# Add variable names
namevec <- c()
for (i in 1:100) {
    name <- paste0("PC", i)
    namevec <- append(namevec, name)
}
names(eigenvalues) <- namevec
colnames(eigenvectors) <- namevec

# Find percentage of variation explained by each
# principal component
eigenpercent <- eigenvalues/sum(eigenvalues) * 100
firstpc <- eigenpercent[1]
secondpc <- eigenpercent[2]

# The first 34 Principal Components contain 95% of the
# variance PC1 contains 25% of the variance, PC2
# contains 16%


# Part 2: PCA Using prcomp()
# ---------------------------------------------

# Conduct PCA using prcomp() and find how each
# variable contributes to each PC
res <- prcomp(scaledata[-101])
lambda <- res$sdev^2
sprintf("%2.3f", lambda/sum(lambda) * 100)
```

```r
index <- c(1:100)
u <- data.frame(res$rotation, index)

# Plot the contributions from each variable to PC1
ggplot(data = u) + geom_point(mapping = aes(x = index, y = PC1),
    color = "blue", alpha = 0.5) + labs(title = "Figure X: Variable Contributions to PC1",
    x = "Variable Index", y = "Contribution to PC1")

# Find five biggest contributors to PC1
varnames <- rownames(u)
pc1sort <- abs(u[, 1])
names(pc1sort) <- varnames
pc1sort <- sort(pc1sort, decreasing = TRUE)
pc1sort <- round(pc1sort, 4)
kable(head(pc1sort, 5), title = "Biggest Contributing Variables to PC1")

# Plot PC1 and PC2 by violent crime rate
scores <- as.data.frame(res$x)
scores$ViolentCrimesPerPop <- communities$ViolentCrimesPerPop
ggplot(data = scores) + geom_point(mapping = aes(x = PC1,
    y = PC2, color = ViolentCrimesPerPop)) + labs(title = "Figure X: Scaled Violent Crime Rate by PC1 a


# Part 3: Linear Regression Model
# -----------------------------------------

# Split data into training and testing sets
set.seed(12345)
id <- sample(1:n, floor(n * 0.5))
train <- communities[id, ]
test <- communities[-id, ]

# Scale data appropriately
trainscaler <- preProcess(train)
scaledtrain <- predict(trainscaler, train)
scaledtest <- predict(trainscaler, test)

# Build regression model and caculate training and
# testing MSE
regmodel <- lm(ViolentCrimesPerPop ~ . - 1, data = scaledtrain)
trainfitvalues <- predict(regmodel)
testfitvalues <- predict(regmodel, newdata = scaledtest)
trainmse <- mean((scaledtrain$ViolentCrimesPerPop - trainfitvalues)^2)
testmse <- mean((scaledtest$ViolentCrimesPerPop - testfitvalues)^2)


# Part 4: Implicit Regularization
# -----------------------------------------

# Create Vectors, Matrices, and Values to Use in
# Calculations:
trainestimates <- as.matrix(regmodel[["coefficients"]])
trainxvalues <- as.matrix(scaledtrain[, 1:100])
```

```r
trainyvalues <- as.matrix(scaledtrain[, "ViolentCrimesPerPop"])
testxvalues <- as.matrix(scaledtest[, 1:100])
testyvalues <- as.matrix(scaledtest[, "ViolentCrimesPerPop"])
trainsigma <- summary(regmodel)$sigma

# Set up optimization function to record MSE:
trainerrors = c()
testerrors = c()
thetazero <- rep(0, 100)
yhat <- c(0)
k = 0

negloglikelihood <- function(thetas, trainx, trainy, testx,
    testy) {
    n = nrow(trainx)
    trainmse <- mean((trainy - (trainx %*% thetas))^2)
    .GlobalEnv$k = .GlobalEnv$k + 1
    .GlobalEnv$yhat = (testx %*% thetas)
    .GlobalEnv$trainerrors[k] = trainmse
    .GlobalEnv$testerrors[k] = mean((testy - (testx %*%
        thetas))^2)
    return(unname(trainmse))
}


# Run and record results of the optimization function
optimizer <- optim(par = thetazero, fn = negloglikelihood,
    trainx = trainxvalues, trainy = trainyvalues, testx = testxvalues,
    testy = testyvalues, method = "BFGS")

iterations <- c(1:k)
errordata <- data.frame(iterations, trainerrors, testerrors)
minimum <- errordata[errordata$testerrors == min(errordata$testerrors),
    ]

# Remove first 1800 data points
errordata <- errordata[1800:length(errordata$iterations),
    ]

ggplot(data = errordata) + geom_line(mapping = aes(x = iterations,
    y = trainerrors), color = "blue") + geom_line(mapping = aes(x = iterations,
    y = testerrors), color = "red") + scale_x_continuous(limits = c(1500,
    10000), breaks = c(1500, 3000, 4500, 6000, 7500, 9000)) +
    geom_vline(xintercept = 2182, linetype = 2) + labs(title = "Figure X: Training and Testing MSE Durin
    x = "Optimization Algorithm Iteration", y = "Mean Squared Error")
```