Assignment 3 lab 2 block 1

Paripurna Bawonoputro

2025-11-27

Assignment 3 Principal components and implicit regularization

First we have to load the data from communities.csv

Task 1

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

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

Implement PCA using eigen() and computing variation explained

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

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

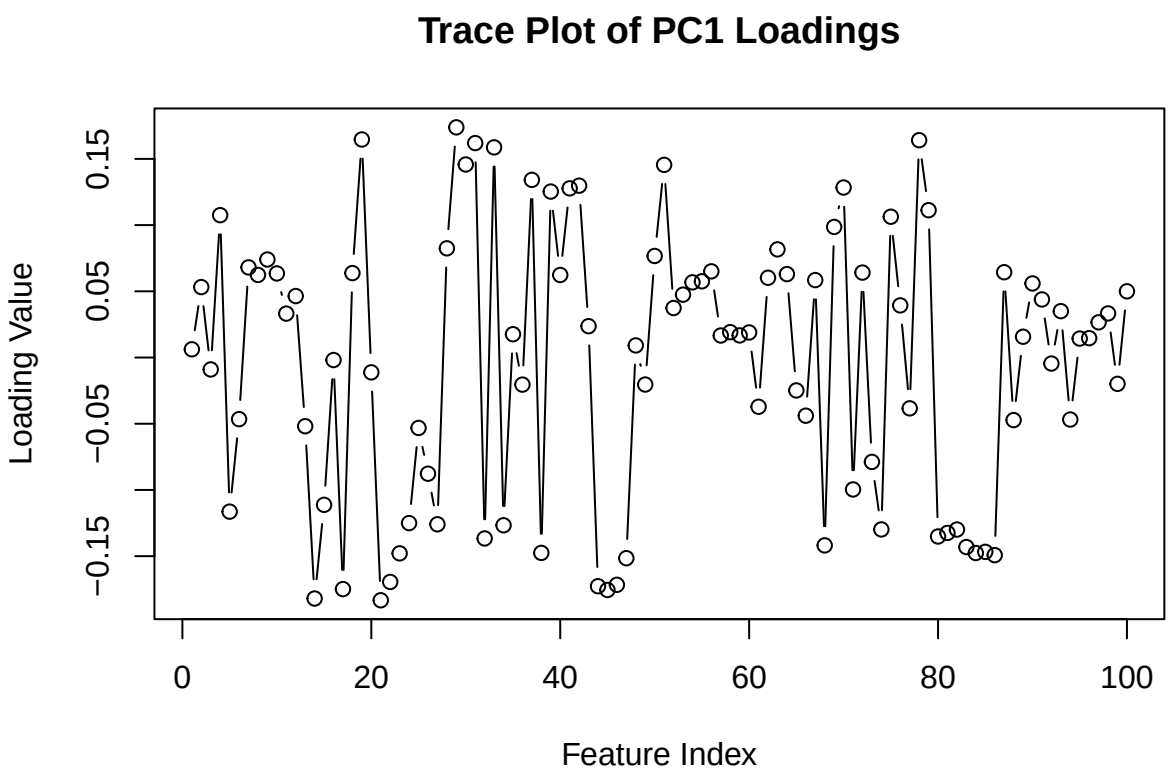
Based on Table 1, components needed to obtain at least 95% of variance in the data is

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

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

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

Task 2



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

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

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

The five variables with the highest loadings on PC1 are:

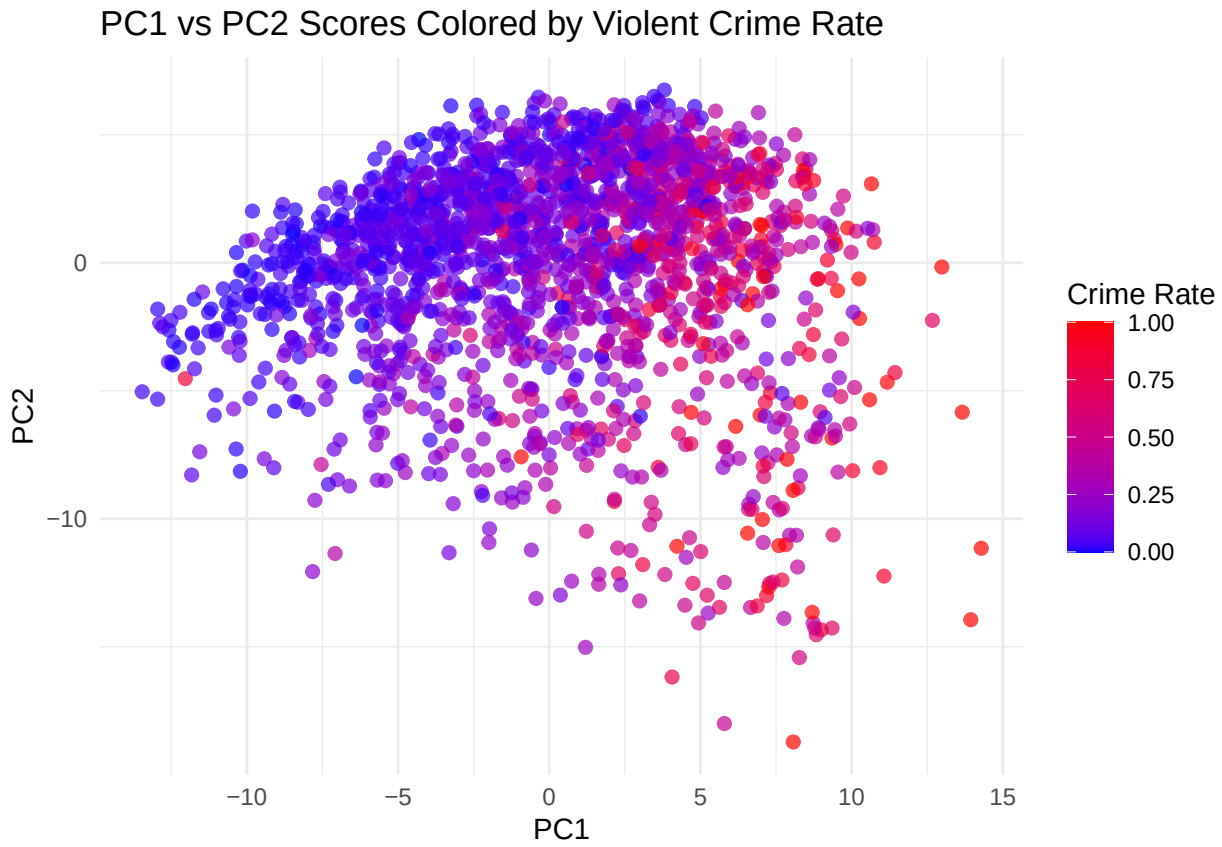
```
## [1] "medFamInc"      "medIncome"      "PctKids2Par"    "pctWInvInc"
## [5] "PctPopUnderPov"
```

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

medFamInc (Median Family Income) negative
medIncome (Median Household Income) negative
PctKids2Par (Percentage of kids in family housing with two parents) negative
pctWInvInc (Percentage of households with investment / rent income in 1989) negative

PctPopUnderPov (Percentage of people under the poverty level) positive

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



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

Task 3

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

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

Then we fit the model and use it for prediction

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

The MSE is defined as

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

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

The R^2 score is computed as

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

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

Table 2: Model Performance on Training and Test Sets

Dataset	MSE	R2
Train	0.2752	0.7245
Test	1.8139	-0.6155

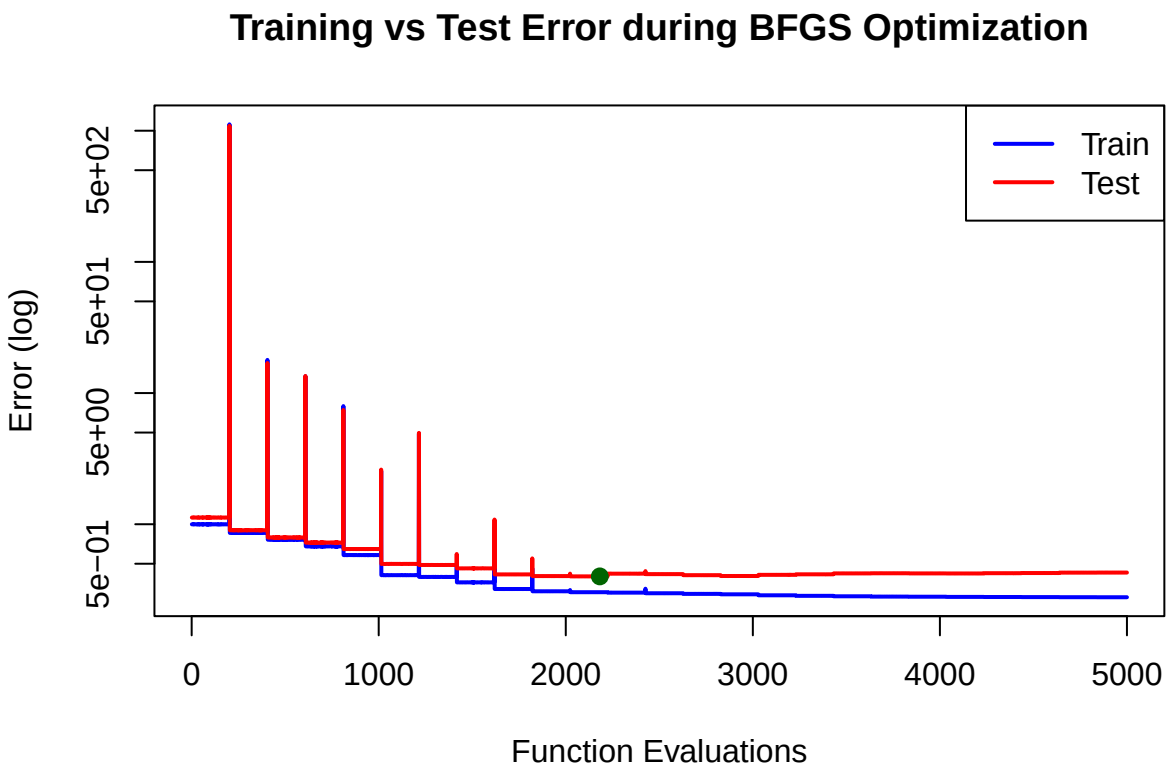
The linear regression model shows good performance on the training set. However, the test set performance is very poor (R^2 = -0.62), meaning the model predicts worse than a simple mean-based model. The large discrepancy between training and test errors indicates overfitting. This concludes that the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

Task 4

The cost function for linear regression without intercept is:

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

Plot the errors



Find optimal iteration with minumum error

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

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

4. Theory

Part 3

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

APPENDIX 3

First we have to load the data from communities.csv

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

Task 1

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

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

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

y <- data$ViolentCrimesPerPop

X_scaled <- scale(X)
```

Implement PCA using eigen() and computing variation explained

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

eigenvalues <- eigen_decomp$values


var_explained <- eigenvalues / sum(eigenvalues)

cum_var <- cumsum(var_explained)

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

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

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

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

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

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

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

The proportion of variation explained by PC1 and PC2 are

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

Task 2

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

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

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

The five variables with the highest loadings on PC1 are:

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

Plot

```r
pca2 <- princomp(X_scaled)

scores <- pca2$scores

library(ggplot2)

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

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

Task 3

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

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

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

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

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

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

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

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

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

Then we fit the model and use it for prediction

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

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

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

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

cat(sprintf("Train MSE: %.4f, R²: %.4f\n", train_mse, train_r2))
cat(sprintf("Test MSE: %.4f, R²: %.4f\n", test_mse, test_r2))
```

Task 4

The cost function for linear regression without intercept is:

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

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

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

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

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

  return(train_err)
}

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

Plot the errors

```r
optimal_iter <- which.min(test_errors)
start <- 500

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

lines(start:length(test_errors),
      test_errors[start:length(test_errors)],
      col = "red", lwd = 2)

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

points(optimal_iter, test_errors[optimal_iter],
       col = "darkgreen", pch = 19, cex = 1.1)
```

Find optimal iteration with minumum error

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