Machine Leaning Lab #1

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

2022-11-20

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

Assignment 1: Handwritten Digit Recognition Using KNN Models

For this assignment, we are given data regarding bitmaps of handwritten digits, and from this data we are going to build a model to recognize which digits are being written.

We begin by shuffling the data randomly and splitting it 50/25/25 into training, validation, and testing sets. We then use the training data to train a 30-nearest-neighbor classifier model. Below are the confusion matrices for this model with the training and testing data, respectively:

Table 1: Confusion matrix for 30-nn on train dataset

	0	1	2	3	4	5	6	7	8	9
0	202	0	0	0	0	0	0	0	0	0
1	0	179	11	0	0	0	0	1	1	3
2	0	1	190	0	0	0	0	1	0	0
3	0	0	0	185	0	1	0	1	0	1
4	1	3	0	0	159	0	0	7	1	4
5	0	0	0	1	0	171	0	1	0	8
6	0	2	0	0	0	0	190	0	0	0
7	0	3	0	0	0	0	0	178	1	0
8	0	10	0	2	0	0	2	0	188	2
9	1	3	0	5	2	0	0	3	3	183

Table 2: Confusion matrix for 30-nn on test dataset

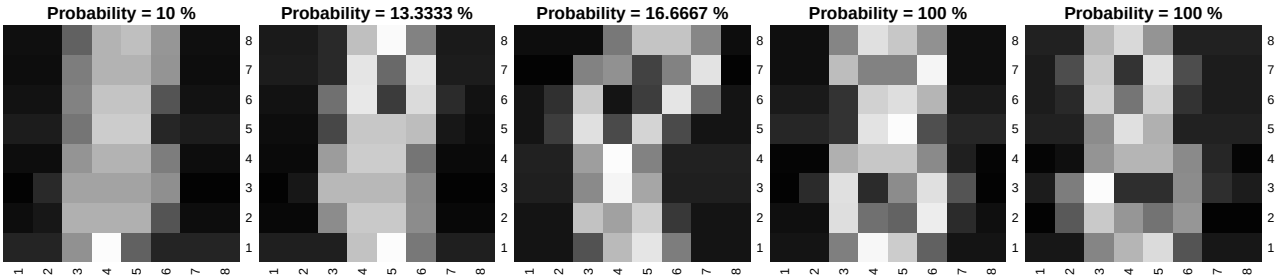
	0	1	2	3	4	5	6	7	8	9
0	77	0	0	0	1	0	0	0	0	0
1	0	81	2	0	0	0	0	0	0	3
2	0	0	98	0	0	0	0	0	3	0
3	0	0	0	107	0	2	0	0	1	1
4	0	0	0	0	94	0	2	6	2	5
5	0	1	1	0	0	93	2	1	0	5
6	0	0	0	0	0	0	90	0	0	0
7	0	0	0	1	0	0	0	111	0	0
8	0	7	0	1	0	0	0	0	70	0
9	0	1	1	1	0	0	0	1	0	85

In the training data, “9,” “1,” “4,” and “8” had the most misclassifications (17 for “9,” 16 for each of the others). The most common misclassifications were “8” being mistaken for “1” (10 instances), “5” being mistaken for “9” (8 instances), and “4” being mistaken for “7” (10 instances). “0” had the least misclassifications - in fact, it was predicted correctly every single time. Additionally, “2”, “6”, “3”, and “7” were all handled well by the model, with only 2, 2, 3, and 4 misclassifications, respectively.

For the test data, “4,” “5,” and “8” have the most misclassifications (15, 10, and 8, respectively), and we see the same three patterns of which numbers are getting misclassified the most: “8” for “1” (7 instances), “4” for “7” (6 instances), and “5” for “9” (5 instances). Once again, “0,” “2,” “6,” and “7” had the fewest errors (1, 3, 0, and 1, respectively).

The misclassification error for the training data is 4.5%, while the misclassification error for the testing data is 5.33%. These error rates suggest that there is significant room for improvement for our predictions.

To test how easy or difficult it is to differentiate the digits, we looked at 5 instances of the number “8,” which included the two rated most likely to be an “8” and the three rated least likely to be an “8.” These instances are shown below:

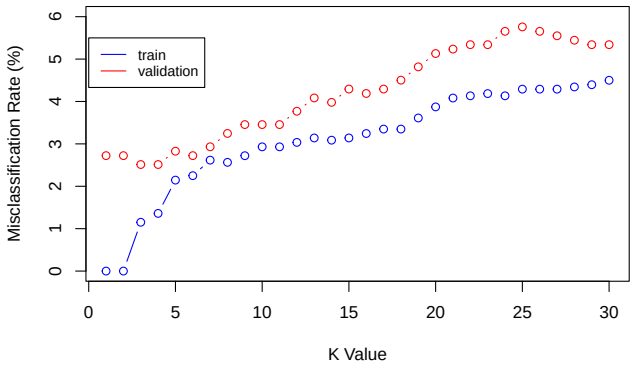


.

The last two plots are clearly resemble an 8, but the first three plots are barely similar to an 8. The first two plots in particular are hard to classify visually because the top and the bottom circles of the 8 figure are blurred out. This is what is leading these to be classified as “1” instead of “8.”

To try to improve these predictions, we set out to find the best number to use in our model for k. To do this, we created thirty different nearest-neighbor models based on the training data, from $k = 1$ to $k = 30$. We then recorded the misclassification error rates for the training and validation data for each model. The results are shown in Figure 1 below:

Figure 1: K−Value vs. Model Misclassification Rate



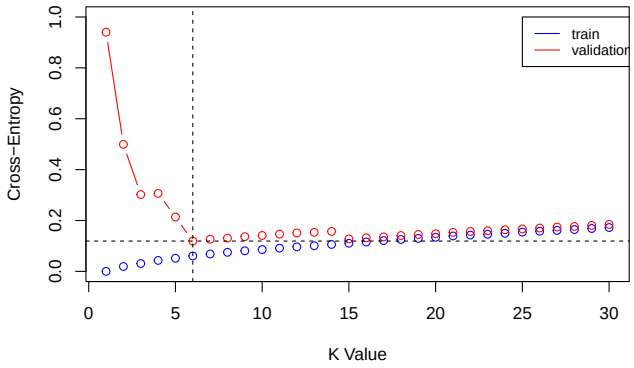
.

As k increases, the error rates for both the training and validation data go up. When $k = 1$ and $k = 2$, the error rate for the training data is basically 0, which indicates overfitting. This is consistent with the idea that a model with to few neighbors is overfitted, while a model with too many neighbors is too simple and underfitted. The error for the validation data is minimized when $k = 3$, which indicates that this would be our optimal k value. When $k = 3$, the misclassification rate for the testing data is 2.4%, while it is 2.51% and 1.15% for the validation and training data, respectively. These misclassification rates are much lower than the ones we received previously, which indicates that this is a better model than our $k = 30$ model from before. Additionally, the error for the testing data is close to the one for the validation data, which is a good sign that model will predict new data points with high accuracy. However, the low misclassification error for the training dataset leads us to be wary that this model may also be overfitted.

To test if this is truly the best value for k, we ran through the same process again, only this time we used cross-entropy as our error measurement rather than just the raw error rate. Figure 2 shows these results:

:

Figure 2: K−Value vs. Cross−Entropy



.

When we use cross-entropy as our error measurement, we see that the validation error is minimized when $k = 6$. Assuming our response has a multinomial distribution, this model may be a more suitable choice for our purposes because cross-entropy uses the probability values to calculate the quality of the model rather than just the results themselves, which leads to a more accurate calculation.

Assignment 2: Linear Regression and Ridge Regression

For this assignment, we are given a data set with 5,875 observations that contains biomedical voice measurements from 42 people with early-stage Parkinson's disease. From this data, we are going to build a linear regression model that measures symptom progression by predicting a disease symptom score called Motor UPDRS using a variety of voice characteristics.

To do this, we started by taking the data set, shuffling it randomly, and splitting it 60/40 into a training and testing data set. We then scaled the based on the training data and applied the same scaling transformation to both the training and testing data.

Next, we built our initial regression model. Since Motor UPDRS is a function of the subject's voice characteristics, we are using the 16 included characteristics in the data set to build our model. And since our data is scaled, there is no need to include a β_0 term for the intercept. Our linear model is as follows:

$$motor\_UPDRS = Jitter * x_1 + Jitter.Abs * x_2 + Jitter.RAP * x_3 + Jitter.PPQ5 * x_4 + Jitter.DDP * x_5 +$$

$$Shimmer * x_6 + Shimmer.dB * x_7 + Shimmer.APQ3 * x_8 + Shimmer.APQ5 * x_9 + Shimmer.APQ11 * x_{10} +$$

$$Shimmer.DDA * x_{11} + NHR * x_{12} + HNR * x_{13} + RPDE * x_{14} + DFA * x_{15} + PPE * x_{16} + \varepsilon_i$$

Using the lm() function in R, we used the training data to find initial values for the relevant coefficients in our regression. The results are seen in the table below:

Table 3: Initial Regression Coefficient Estimates

term	estimate	std.error	statistic	p.value
Jitter...	0.1869308	0.1495605	1.2498674	0.2114314
Jitter.Abs.	-0.1696093	0.0408055	-4.1565332	0.0000331
Jitter.RAP	-5.2695439	18.8341599	-0.2797865	0.7796578
Jitter.PPQ5	-0.0745680	0.0877659	-0.8496247	0.3955917
Jitter.DDP	5.2495583	18.8375252	0.2786756	0.7805102
Shimmer	0.5924358	0.2059810	2.8761675	0.0040496
Shimmer.dB.	-0.1726554	0.1393162	-1.2393066	0.2153149
Shimmer.APQ3	32.0709324	77.1592423	0.4156460	0.6776945
Shimmer.APQ5	-0.3875068	0.1137892	-3.4054787	0.0006679
Shimmer.APQ11	0.3055462	0.0612360	4.9896520	0.0000006
Shimmer.DDA	-32.3872408	77.1588144	-0.4197478	0.6746954
NHR	-0.1853867	0.0455674	-4.0684029	0.0000484
HNR	-0.2385435	0.0363954	-6.5542224	0.0000000
RPDE	0.0040681	0.0226637	0.1794988	0.8575564
DFA	-0.2803175	0.0201360	-13.9211999	0.0000000
PPE	0.2264671	0.0328813	6.8874171	0.0000000

Based on the model shown above, we see that the most significant variables are Jitter.Abs, Shimmer, Shimmer.APQ5, Shimmer.APQ11, NHR, HNR, DFA, and PPE, all of which are significant at at least a 99% confidence level. Additionally, predicting Motor UPDRS from the training data on this model gives us a mean squared error (MSE) of 0.8785 and a residual standard error of 0.9394.

From there, we generated a negative-log-likelihood function based on the regression model, built a ridge model that manages the complexity of the regression to avoid overfitting, and wrote an optimization function that provides the optimal parameter estimates for the training data and a ridge coefficient λ. This optimization function takes a vector of initial values for each of the parameters and the dispersion σ, as well as a scalar value for λ, and generates optimal values for each parameter given λ. For our purposes, we used the coefficient estimates from the earlier regression as our initial parameter values, and the residual standard error as the initial σ. We used this optimization function to create three different models: one with $\lambda = 1$, one with $\lambda = 100$ and one with $\lambda = 1000$. Below is a summary of the MSE generated by each model for the training and testing data, as well as the degrees of freedom calculated for each model:

Table 4: MSE and Degrees of Freedom for Different Lambda Values

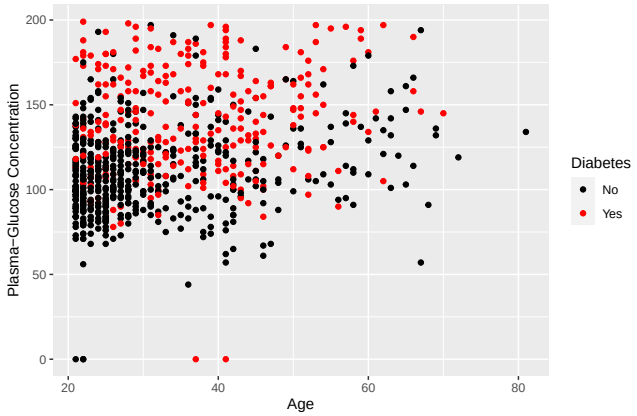
	Lamba = 1	Lamba = 100	Lamba = 1000
Train MSE	0.8786	0.8844	0.9211
Test MSE	0.9350	0.9323	0.9539
Degrees of Freedom	13.8607	9.9249	5.6439

Given the choice of these three options, we would choose to use the model where $\lambda = 100$. The comparison of the MSE values shows that the model appears to be just as good at predicting the test values as it is the training values. The MSE for both sets of data is lower for $\lambda = 100$ than it is for $\lambda = 1000$. And even though the MSE values are about the same for $\lambda = 1$, it is still wiser to use the $\lambda = 100$ model, as there is a lower risk of overfitting from the less complex model.

Assignment 3: Logistic Regression and Basis Function Expansion

For this assignment, we are given measurements taken from the Pima tribe, and using this data we are going to create a model to predict the onset of diabetes. For this assignment, the data will not be split, it will all be used to train the model. The initial data is shown below in Figure 3, with age on the x-axis, plasma-glucose concentration on the y-axis, and individuals with diabetes shown in red:

Figure 3: Raw Training Data for Diabetes Model



.

From this data alone, it seems difficult to classify diabetes using a standard logistic regression model using just these two variables as our features. For one thing, there is no clear pattern in the data, as those with diabetes appear to be dispersed somewhat randomly. Additionally, there is a large group of those without diabetes concentrated on the lower left corner of the graph, which would likely skew the findings of the regression. When we create such a model with a classification threshold of $r = 0.5$, we get the following coefficient model:

Table 5: Initial Logistic Regression Coefficient Estimates

term	estimate	std.error	statistic	p.value
(Intercept)	-5.9124491	0.4626197	-12.780366	0.0000000
plasma_glucose_concentration	0.0356440	0.0032900	10.834088	0.0000000
age	0.0247784	0.0073737	3.360364	0.0007784

According to these coefficients, the probabilistic equation is as follows:

$$p = \frac{1}{1 + e^{5.9124 + 0.0356 * plasma\ glucose\ concentration + 0.0247 age}}$$

The decision boundary is defined as:

$$plasma\ glucose\ concentration = \frac{5.912}{0.03565} + \frac{-0.0247}{0.0356} age = 165.8345 - 0.6938 age$$

and the misclassification rate is 26.3%.

The results of this initial model based on the training data are shown in Figure 4 below, where this time the people *predicted* to have diabetes by the model are shown in red:

Figure 4: Initial Logistic Regression Model

.

Based on the data, it appears that the quality of these classifications are mediocre. The overall misclassification rate of 26.3% seems high, and in particular the predictions for older people is not ideal compared to what we see in the original data. Because the decision boundary is based on the model's predictions and not the original data, it separates the predictions well but does not reflect the original data.

When we change the classification threshold to $r = 0.2$ and $r = 0.8$, we get the scatterplots shown in Figures 5 and 6 below:

Figure 5: Logistic Regression Model wtih r = 0.2

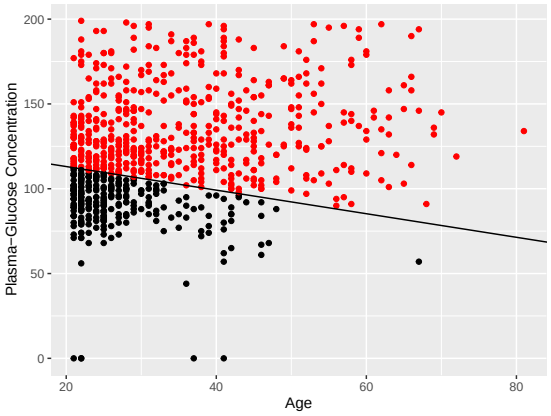
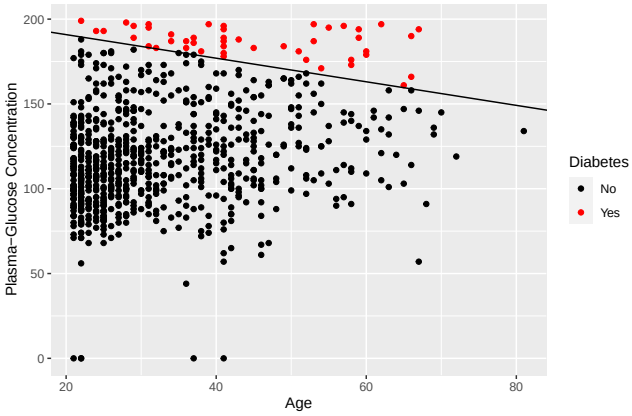


Figure 6: Logistic Regression Model wtih r = 0.8

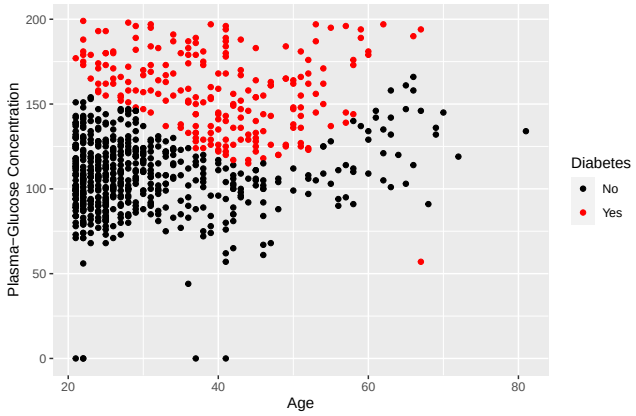


.

When $r = 0.8$, the decision boundary for predicting diabetes moves toward to the top of the scatterplot, which means that fewer people are predicted to have diabetes. When $r = 0.2$, the opposite happens – the decision boundary of having diabetes moves toward to the bottom of the plot, and the model predicts more people having diabetes. Both cases resulted in higher misclassification rates, which means that the predictions in both cases became less accurate.

In order to improve the prediction accuracy of our model, we trained a new model using the same data – except this time, we used a basis expansion to add new features and more complexity to the model. The prediction results for this new model are shown below in Figure 7:

Figure 7: Logistic Regression with Basis Function Expansion



.

The misclassification rate for this model is 24.48%, which is still far from perfect, but better that all of the other models tested in this assignment. The basis expansion changed the shape of the decision boundary to account for variations in the data as age increases. Because of this, the prediction is closer to the original data. Due to higher dimension of the features, the decision boundary is hard to visualize on a 2-D graph, but we can still observe by looking at the color difference. Overall, using the basis expansion improved our prediction accuracy compared to the basic model.

Appendix 1: Code for Assignment 1

```
# 1
library(tidyverse)
library(kknn)
optdigits <- read.csv("./dataset/optdigits.csv", header = FALSE)
optdigits <- optdigits %>%
    mutate(y = as.factor(V65)) %>%
    select(!V65)
n <- nrow(optdigits)
set.seed(12345)
train_idx <- sample(seq_len(n), floor(n * 0.5))
train_data <- optdigits[train_idx, ]
remainder_idx <- setdiff(seq_len(n), train_idx)
set.seed(12345)
valid_idx <- sample(remainder_idx, floor(n * 0.25))
valid_data <- optdigits[valid_idx, ]
test_idx <- setdiff(remainder_idx, valid_idx)
test_data <- optdigits[test_idx, ]

# 2
knn_model_train <- kknn(y ~ ., train = train_data, test = train_data,
    k = 30, kernel = "rectangular")
g_hat_train <- knn_model_train$fitted.values
cm_train <- table(train_data$y, g_hat_train)
knitr::kable(cm_train, caption = "Confusion matrix for 30-nn on train dataset")
error_train <- 1 - (sum(diag(cm_train))/length(g_hat_train))
knn_model_test <- kknn(y ~ ., train = train_data, test = test_data,
    k = 30, kernel = "rectangular")
g_hat_test <- knn_model_test$fitted.values
cm_test <- table(test_data$y, g_hat_test)
knitr::kable(cm_test, caption = "Confusion matrix for 30-nn on test dataset")
error_test <- 1 - (sum(diag(cm_test))/length(g_hat_test))

# 3
prob_g8 <- predict(knn_model_train, type = "prob")
prob_g8 <- prob_g8[, "8"]
observations <- train_data %>%
    mutate(prob = prob_g8) %>%
    filter(y == "8") %>%
    arrange(prob) %>%
    slice(c(1:3, length(y) - 1, length(y)))
for (i in 1:5) {
    fig_matrix <- matrix(as.numeric(observations[i, 1:64]),
        nrow = 8, byrow = T)
    fig_title <- paste("Probability =", round(observations[i,
        "prob"] * 100, 4), "%")
    heatmap(fig_matrix, Colv = NA, Rowv = NA, col = paste0("gray",
        1:99), main = fig_title)
}

# 4
results <- map(1:30, ~{
    model <- kknn(y ~ ., train = train_data, test = train_data,
```

```
        k = .x, kernel = "rectangular")
    train_sum <- sum(diag(table(train_data$y, model$fitted.values)))
    model <- kknn(y ~ ., train = train_data, test = valid_data,
        k = .x, kernel = "rectangular")
    valid_sum <- sum(diag(table(valid_data$y, model$fitted.values)))
    c(train_sum, valid_sum)
})
results <- as.data.frame(results)
names(results) <- 1:30
lengths <- c(nrow(train_data), nrow(valid_data))
errors <- (lengths - results)/lengths * 100
plot(t(errors)[, 1], type = "b", col = "blue", ylim = c(0,
    6), xlab = "Vlaue of K", ylab = "Misclassification Error")
points(t(errors)[, 2], type = "b", col = "red")
legend(0, 5.5, legend = c("train", "validation"), col = c("blue",
    "red"), lty = 1, cex = 0.8)

# 5
cross_entropy <- function(y, prob) {
    one_hot <- model.matrix(~0 + y)
    result <- sum(-one_hot * log(prob + 1e-15))
    return(result/length(y))
}
results <- map(1:30, ~{
    model <- kknn(y ~ ., train = train_data, test = train_data,
        k = .x, kernel = "rectangular")
    ce_train <- cross_entropy(train_data$y, model$prob)
    model <- kknn(y ~ ., train = train_data, test = valid_data,
        k = .x, kernel = "rectangular")
    ce_valid <- cross_entropy(valid_data$y, model$prob)
    c(ce_train, ce_valid)
})
results <- as.data.frame(results)
names(results) <- 1:30
best_k <- which.min(t(results)[, 2])
ce_lowest <- t(results)[best_k, 2]
plot(t(results)[, 1], type = "b", col = "blue", ylim = c(0,
    1), xlab = "Vlaue of K", ylab = "Misclassification Error")
points(t(results)[, 2], type = "b", col = "red")
abline(v = 6, h = ce_lowest, lty = 2)
legend(25, 1, legend = c("train", "validation"), col = c("blue",
    "red"), lty = 1, cex = 0.8)
```

Appendix 2: Code for Assignment 2

```
# Split and Prepare Data
# ------------------------------------------------

# Split the Data Into Training and Testing Data
# (60/40):
parkinsons <- read.csv("parkinsons.csv")
set.seed(0)
n = nrow(parkinsons)
id = sample(1:n, floor(n * 0.6))
train = parkinsons[id, ]
test = parkinsons[-id, ]


# Scale the Data Appropriately:
scaler = preProcess(train)
scaledtrain = predict(scaler, train)
scaledtest = predict(scaler, test)


# Initial Regression and Values
# ------------------------------------------

# Create the Regression Model and Calculate MSE:
regmodel <- lm(motor_UPDRS ~ Jitter... + Jitter.Abs. + Jitter.RAP +
    Jitter.PPQ5 + Jitter.DDP + Shimmer + Shimmer.dB. + Shimmer.APQ3 +
    Shimmer.APQ5 + Shimmer.APQ11 + Shimmer.DDA + NHR + HNR +
    RPDE + DFA + PPE - 1, data = scaledtrain)
trainsigma <- summary(regmodel)$sigma
fitvalues <- predict(regmodel)
initmse <- mean((scaledtrain$motor_UPDRS - fitvalues)^2)

# Create Vectors, Matrices, and Values to Use in
# Calculations:
trainestimates <- as.matrix(regmodel[["coefficients"]])
trainxvalues <- as.matrix(scaledtrain[, 7:22])
trainyvalues <- as.matrix(scaledtrain[, "motor_UPDRS"])
testxvalues <- as.matrix(scaledtest[, 7:22])
testyvalues <- as.matrix(scaledtest[, "motor_UPDRS"])
trainsigma <- summary(regmodel)$sigma
thetasigma <- c(regmodel[["coefficients"]], stderr = trainsigma)


# Log Likelihood
# ------------------------------------------------------------

loglikelihood <- function(thetas, sigma) {
    n = nrow(trainxvalues)
    likelihood <- ((-n/2) * log(2 * pi * (sigma^2))) - (1/(2 *
        (sigma^2)) * sum(((trainxvalues %*% thetas) - trainyvalues)^2))
    return(unname(likelihood))
}
```

```r
# Ridge Regression
# ---------------------------------------------------------

ridge <- function(parameters, lambda) {
    thetas <- parameters[1:16]
    sigma <- parameters[17]
    penalty <- lambda * sum(thetas^2)
    ridgelikelihood <- -loglikelihood(thetas = thetas, sigma = sigma) +
        penalty
    return(ridgelikelihood)
}


# Ridge Optimization
# ---------------------------------------------------------

ridgeopt <- function(lambda) {
    optim(par = thetasigma, fn = ridge, lambda = lambda,
        method = "BFGS")
}


# Degrees of Freedom
# ---------------------------------------------------------

df <- function(lambda) {
    hatmat <- trainxvalues %*% solve((t(trainxvalues) %*%
        trainxvalues) + diag(lambda, length(trainestimates))) %*%
        t(trainxvalues)
    degrees <- sum(diag(hatmat))
    return(degrees)
}


# Computing and Testing Optimal Parameters
# --------------------------------

# Obtain Optimal Parameters:
lambdaone <- as.matrix(ridgeopt(1)$par[1:16])
lambdahundred <- as.matrix(ridgeopt(100)$par[1:16])
lambdathousand <- as.matrix(ridgeopt(1000)$par[1:16])


# Calculate Estimated Values for Training Data, Test
# Data and DF:
trainestone <- as.vector(trainxvalues %*% lambdaone)
trainesthundred <- as.vector(trainxvalues %*% lambdahundred)
trainestthousand <- as.vector(trainxvalues %*% lambdathousand)

testestone <- as.vector(testxvalues %*% lambdaone)
testesthundred <- as.vector(testxvalues %*% lambdahundred)
testestthousand <- as.vector(testxvalues %*% lambdathousand)
```

```r
# Calculate MSE for Each Set of Y-hats:
returnmse <- function(data, yhats) {
    if (data == "train") {
        mse <- mean((scaledtrain$motor_UPDRS - yhats)^2)
    } else if (data == "test") {
        mse <- mean((scaledtest$motor_UPDRS - yhats)^2)
    }
}

# Generate MSE and Degrees of Freedom Table:
trainmseone <- returnmse(data = "train", yhats = trainestone)
trainmsehundred <- returnmse(data = "train", yhats = trainesthundred)
trainmsethousand <- returnmse(data = "train", yhats = trainestthousand)

testmseone <- returnmse(data = "test", yhats = testestone)
testmsehundred <- returnmse(data = "test", yhats = testesthundred)
testmsethousand <- returnmse(data = "test", yhats = testestthousand)

dfone <- df(1)
dfhundred <- df(100)
dfthousand <- df(1000)

one <- c(trainmseone, testmseone, dfone)
hundred <- c(trainmsehundred, testmsehundred, dfhundred)
thousand <- c(trainmsethousand, testmsethousand, dfthousand)

modeldata <- data.frame(one, hundred, thousand)
modeldata <- round(modeldata, 4)
colnames(modeldata) <- c("Lamba = 1", "Lamba = 100", "Lamba = 1000")
rownames(modeldata) <- c("Train MSE", "Test MSE", "Degrees of Freedom")
```

Appendix 3: Code for Assignment 3

```
# =====Assignment 3=====

# =====Set Up=====
diabetes_data <- read.csv("D:/pima-indians-diabetes.csv",
    header = FALSE)
colnames(diabetes_data) <- c("number_of_times_pregnant",
    "plasma_glucose_concentration", "blood_pressure", "triceps_skinfold_thickness",
    "serum_insulin", "bmi", "diabetes_pedigree_function",
    "age", "diabetes")
library(ggplot2)

# ===== ÉX 3.1===== create 'diabetes_data_1' so the
# original data wont be affect
diabetes_data_1 <- diabetes_data

# use 'as.factor' so 1 means has diabetes, 0 means no
# diabetes
diabetes_data_1$diabetes <- as.factor(ifelse(diabetes_data_1$diabetes ==
    1, "Yes", "No"))  # use 'as.factor' so 1 means has diabetes, 0 means no diabetes
plot_assigment3_q1 <- ggplot(diabetes_data_1, aes(x = age,
    y = plasma_glucose_concentration, color = diabetes)) +
    geom_point() + labs(title = "Assigment 3 Question 1, the original data",
    colour = "Diabetes") + scale_color_manual(values = c("#000000",
    "#ff0000"))

plot_assigment3_q1

# ===== ÉX 3.2===== =====1=====
model_1 <- glm(diabetes ~ plasma_glucose_concentration +
    age, data = diabetes_data_1, family = binomial)
summary(model_1)$coef
diabetes_data_1$probabilities <- predict(model_1, diabetes_data_1,
    type = "response")
# The type='response' option tells R to output
# probabilities of the form P(Y = 1|X), as opposed to
# other information such as the logit.

diabetes_data_1$predicted_classes_0.5 <- as.factor(ifelse(diabetes_data_1$probabilities >
    0.5, "Yes", "No"))

# =====2=====
missclass = function(X, X1) {
    n = length(X)
    return(1 - sum(diag(table(X, X1)))/n)
}
missclassification_ex2 <- missclass(diabetes_data_1$diabetes,
    diabetes_data_1$predicted_classes_0.5)
missclassification_ex2
# =====3=====
plot_assigment3_q2 <- ggplot(diabetes_data_1) + geom_point(aes(x = age,
    y = plasma_glucose_concentration, color = predicted_classes_0.5)) +
    labs(title = "Assigment 3 Question 2, r=0.5", colour = "Diabetes") +
```

```
    scale_color_manual(values = c("#000000", "#ff0000"))

plot_assigment3_q2

# ===== ÉX 3.3===== To correct the intercept on the
# plot if the threshold is not 0.5
inverse_logit <- function(threshold) {
    return(-log((1 - threshold)/threshold))

}
decision_boundary <- function(a, b, c, ...) {
    # function to plot decision boundary
    slope <- -a/b
    intercept <- -c/b
    geom_abline(slope = slope, intercept = intercept, ...)

}
plot_assigment3_q3 <- ggplot(diabetes_data_1) + geom_point(aes(x = age,
    y = plasma_glucose_concentration, color = predicted_classes_0.5)) +
    labs(title = "Assigment 3 Question 3, r=0.5, with decision boundary",
        colour = "Diabetes") + scale_color_manual(values = c("#000000",
    "#ff0000")) + decision_boundary(model_1$coefficients[3],
    model_1$coefficients[2], model_1$coefficients[1] - inverse_logit(0.5))
plot_assigment3_q3

# ===== ÉX 3.4===== ===== r=0.2 =====
diabetes_data_1$predicted_classes_0.2 <- as.factor(ifelse(diabetes_data_1$probabilities >
    0.2, "Yes", "No"))

plot_assigment3_q4_r0.2 <- ggplot(diabetes_data_1) + geom_point(aes(x = age,
    y = plasma_glucose_concentration, color = predicted_classes_0.2)) +
    labs(title = "Assigment 3 Question 4, r=0.2, with decision boundary",
        colour = "Diabetes") + scale_color_manual(values = c("#000000",
    "#ff0000")) + decision_boundary(model_1$coefficients[3],
    model_1$coefficients[2], model_1$coefficients[1] - inverse_logit(0.2))

plot_assigment3_q4_r0.2

# ===== r=0.8 =====
diabetes_data_1$predicted_classes_0.8 <- as.factor(ifelse(diabetes_data_1$probabilities >
    0.8, "Yes", "No"))

plot_assigment3_q4_r0.8 <- ggplot(diabetes_data_1) + geom_point(aes(x = age,
    y = plasma_glucose_concentration, color = predicted_classes_0.8)) +
    labs(title = "Assigment 3 Question 4, r=0.8, with decision boundary",
        colour = "Diabetes") + scale_color_manual(values = c("#000000",
    "#ff0000")) + decision_boundary(model_1$coefficients[3],
    model_1$coefficients[2], model_1$coefficients[1] - inverse_logit(0.8))
plot_assigment3_q4_r0.8

# ===== ÉX 3.5=====
diabetes_data_ex5 <- diabetes_data
# Create new data frame so it won't affect the
```

```r
# previous data frame

# Add new features
diabetes_data_ex5$z1 <- (diabetes_data_ex5$plasma_glucose_concentration)^4
diabetes_data_ex5$z2 <- (diabetes_data_ex5$plasma_glucose_concentration)^3 *
    diabetes_data_ex5$age
diabetes_data_ex5$z3 <- (diabetes_data_ex5$plasma_glucose_concentration)^2 *
    (diabetes_data_ex5$age)^2
diabetes_data_ex5$z4 <- (diabetes_data_ex5$plasma_glucose_concentration)^1 *
    (diabetes_data_ex5$age)^3
diabetes_data_ex5$z5 <- (diabetes_data_ex5$age)^4

# Do the model using glm with new features
model_2 <- glm(diabetes ~ plasma_glucose_concentration +
    age + z1 + z2 + z3 + z4 + z5, data = diabetes_data_ex5,
    family = binomial)

diabetes_data_ex5$probabilities <- predict(model_2, diabetes_data_ex5,
    type = "response")
diabetes_data_ex5$predicted_classes_0.5 <- as.factor(ifelse(diabetes_data_ex5$probabilities >
    0.5, "Yes", "No"))
plot_assigment3_q5 <- ggplot(diabetes_data_ex5, aes(x = age,
    y = plasma_glucose_concentration)) + geom_point(aes(x = age,
    y = plasma_glucose_concentration, color = predicted_classes_0.5)) +
    scale_color_manual(values = c("#000000", "#ff0000")) +
    labs(title = "Assigment 5, r=0.5", colour = "Diabetes")
plot_assigment3_q5

missclassification_ex5 <- missclass(diabetes_data_ex5$diabetes,
    diabetes_data_ex5$predicted_classes_0.5)
cat("The missclassification is", missclassification_ex5)
```