Machine Learning Lab1

Lepeng Zhang, Xuan Wang, Priyarani Patil

2023-11-08

Assignment 1. Handwritten digit recognition with Knearest neighbors.

1.

Answer:

2.

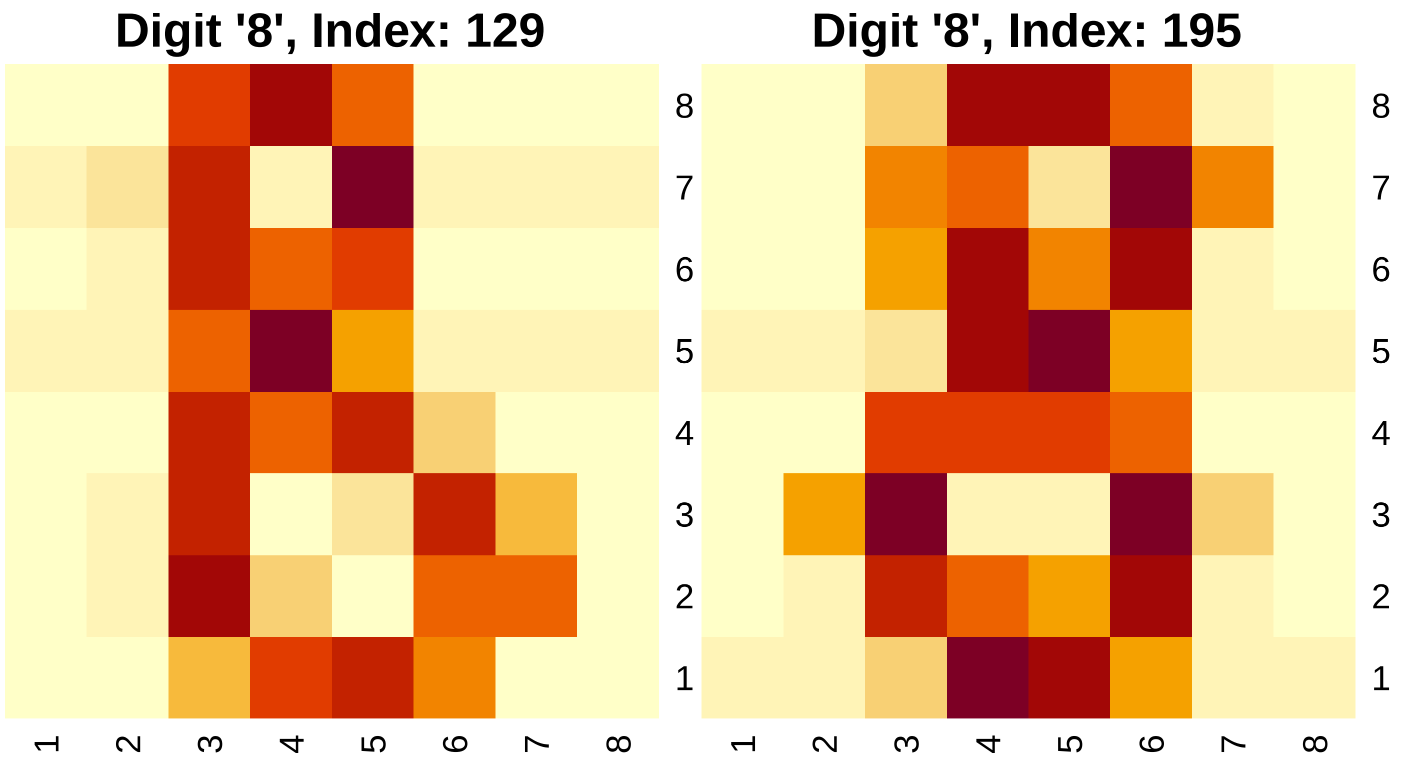
Answer:

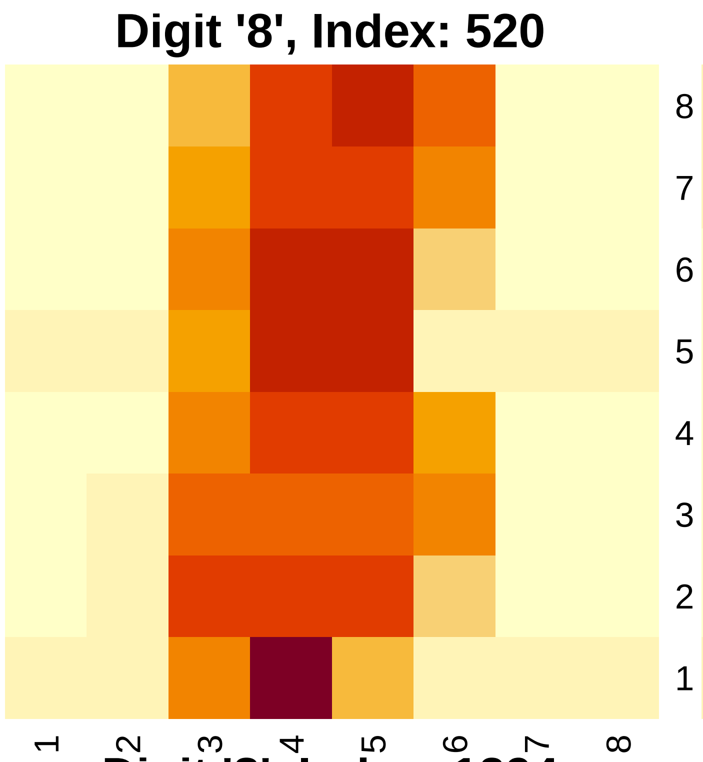
```
## Confusion Matrix for Training Data:

##
##       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

## Confusion Matrix for Test Data:

##
##      0  1  2  3  4  5  6  7  8  9
##   0 82  0  0  0  1  0  1  0  0  0
##   1  0 90  2  0  0  0  0  0  0  3
##   2  0  1 92  0  0  0  0  1  1  1
##   3  0  0  0 85  0  2  0  3  1  1
##   4  0  1  0  0 89  0  1  6  0  5
##   5  0  1  0  1  0 97  1  1  0  7
##   6  0  0  0  0  0  0 97  0  0  0
##   7  0  1  0  1  0  0  0 99  0  0
##   8  0  7  0  0  0  0  0  0 84  0
##   9  0  2  0  0  0  0  0  2  1 86

## Misclassification errors for the training data are:  0.04500262

## Misclassification errors for the test data are:  0.0585162

## The accuracy of prediction of for digit  0  is:  0.9761905
## The accuracy of prediction of for digit  1  is:  0.9473684
## The accuracy of prediction of for digit  2  is:  0.9583333
## The accuracy of prediction of for digit  3  is:  0.923913
## The accuracy of prediction of for digit  4  is:  0.872549
## The accuracy of prediction of for digit  5  is:  0.8981481
## The accuracy of prediction of for digit  6  is:  1
## The accuracy of prediction of for digit  7  is:  0.980198
## The accuracy of prediction of for digit  8  is:  0.9230769
## The accuracy of prediction of for digit  9  is:  0.9450549
```

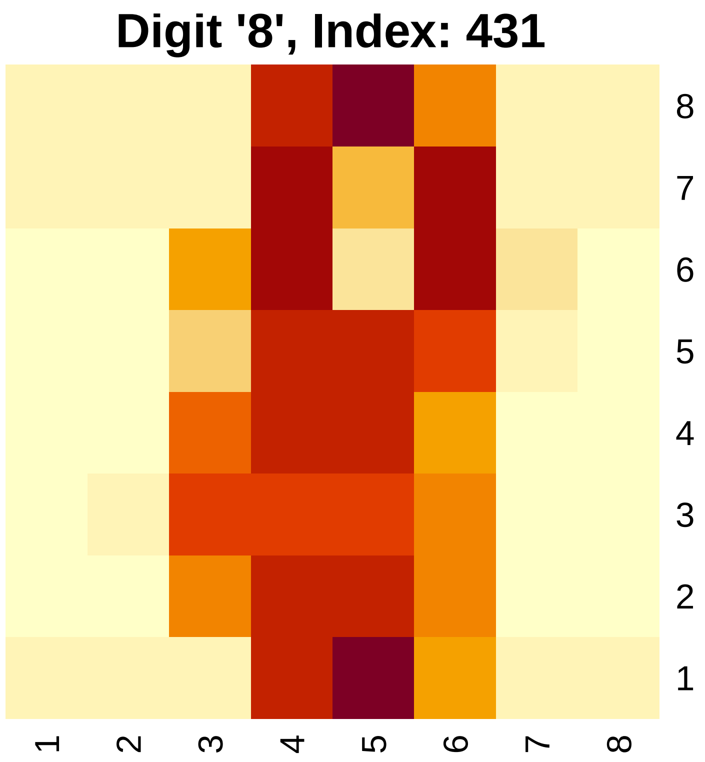
```
## The overall accuracy of prediction is: 0.9414838
```

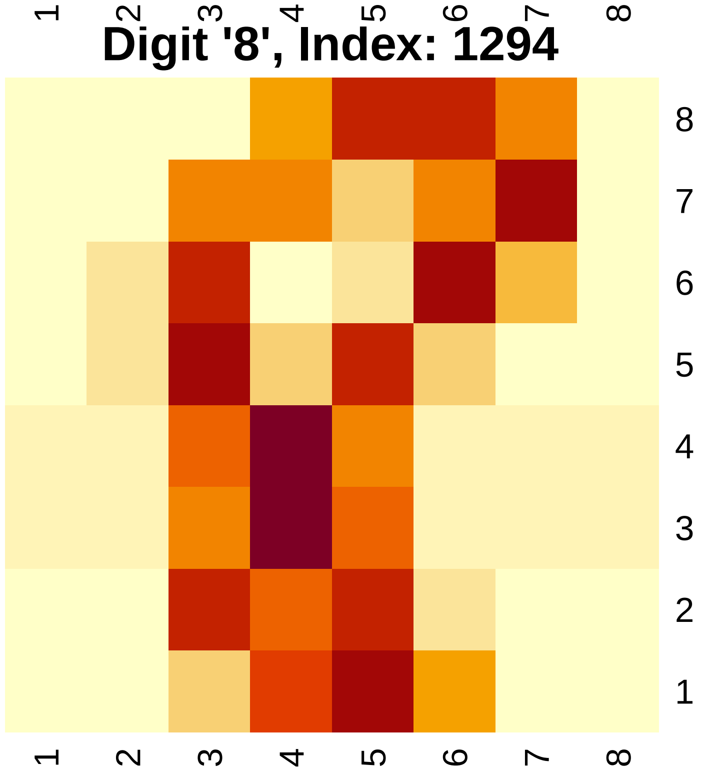
3.

Answer:



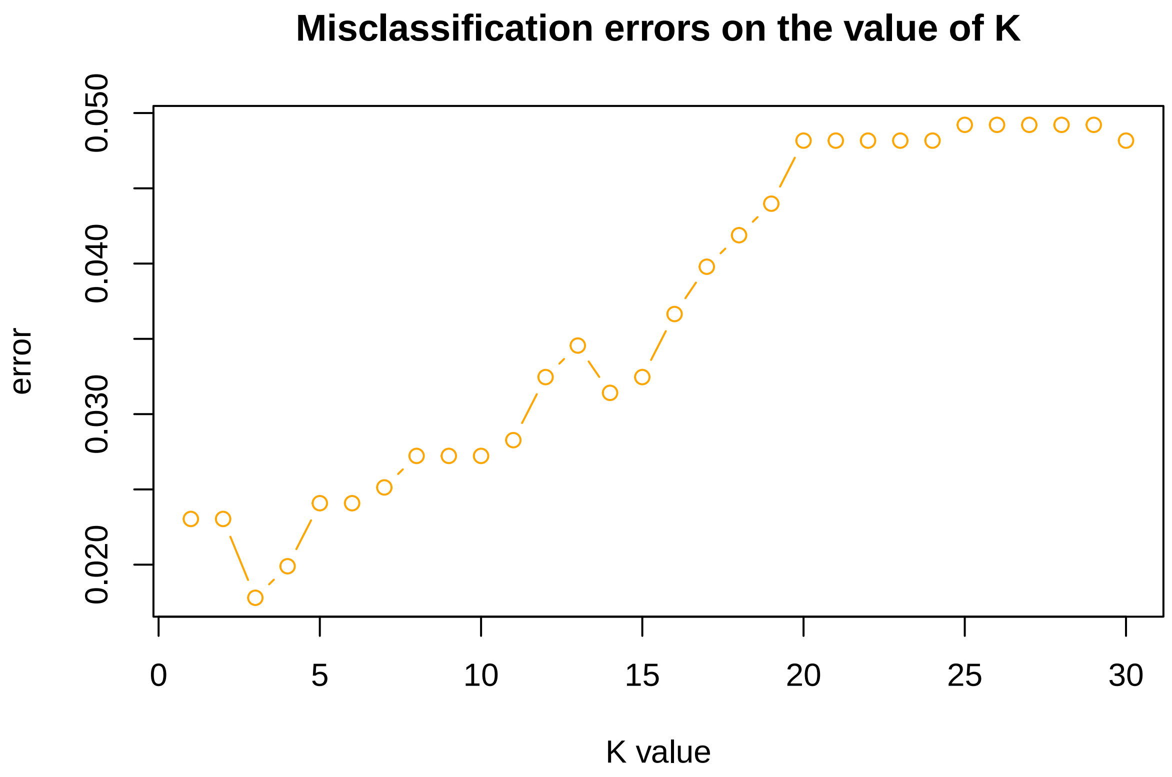






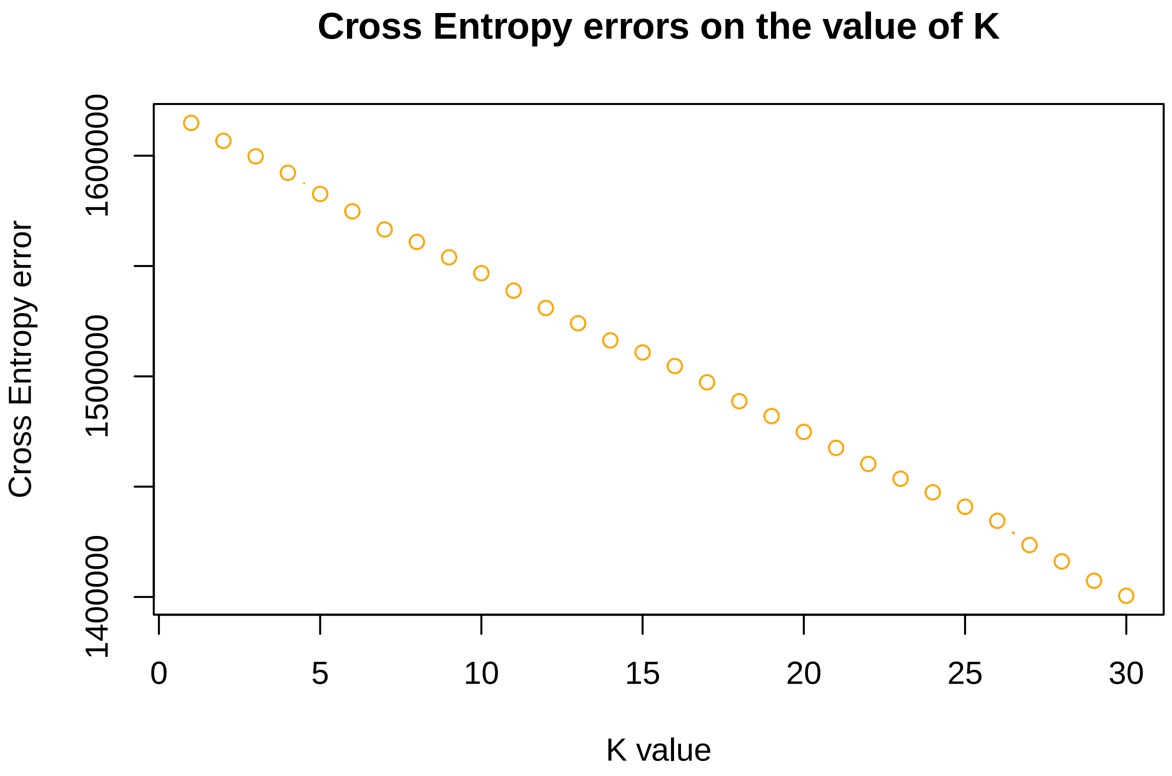
4.

Answer:



5.

Answer:



```
## The optimal k is: 30
```

Assignment 2. Linear regression and ridge regression

1.

Answer:

2.

Answer:

```
## The training data's MSE is:  6.348095
```

```
## The training data's MSE is:  6.087681
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ ., data = scaled_train_data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.654 -1.373  0.173  1.697  7.062
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    21.34657    0.04257 501.449  < 2e-16 ***
## subject.       -0.21493    0.04934  -4.356 1.36e-05 ***
## age            -0.28601    0.04640  -6.164 7.89e-10 ***
## sex             0.47954    0.05242   9.149  < 2e-16 ***
## test_time      -0.02043    0.04302  -0.475 0.634817
## total_UPDRS     7.78037    0.04958 156.933  < 2e-16 ***
## Jitter...       1.37390    0.40291   3.410 0.000657 ***
## Jitter.Abs.    -0.64396    0.11742  -5.484 4.45e-08 ***
## Jitter.RAP      5.95170   50.69756   0.117 0.906553
## Jitter.PPQ5    -0.28759    0.23807  -1.208 0.227127
## Jitter.DDP     -6.75744   50.70512  -0.133 0.893988
## Shimmer         1.35532    0.55653   2.435 0.014929 *
## Shimmer.dB.    -0.22604    0.37564  -0.602 0.547387
## Shimmer.APQ3   77.03110  207.68721   0.371 0.710735
## Shimmer.APQ5   -1.15985    0.30801  -3.766 0.000169 ***
## Shimmer.APQ11   0.62652    0.16665   3.760 0.000173 ***
## Shimmer.DDA   -77.49708  207.68684  -0.373 0.709064
## NHR             0.01178    0.12627   0.093 0.925651
## HNR            -0.03483    0.09940  -0.350 0.726048
## RPDE           -0.24771    0.06150  -4.028 5.75e-05 ***
## DFA            -0.05569    0.05672  -0.982 0.326256
## PPE             0.44229    0.08899   4.970 7.02e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.527 on 3503 degrees of freedom
## Multiple R-squared:  0.9033, Adjusted R-squared:  0.9027
## F-statistic:  1559 on 21 and 3503 DF,  p-value: < 2.2e-16
```

3.

Answer:

4.

Answer:

```
## Ridge Training MSE: 2.16166e+29 7.940084e+28 8.990485e+31
```

```
## Ridge Test MSE: 2.154747e+29 7.778365e+28 8.745325e+31
```

```
## Degrees of Freedom: 18.8572 14.70172 9.321473
```

Assignment 3. Logistic regression and basis function expansion

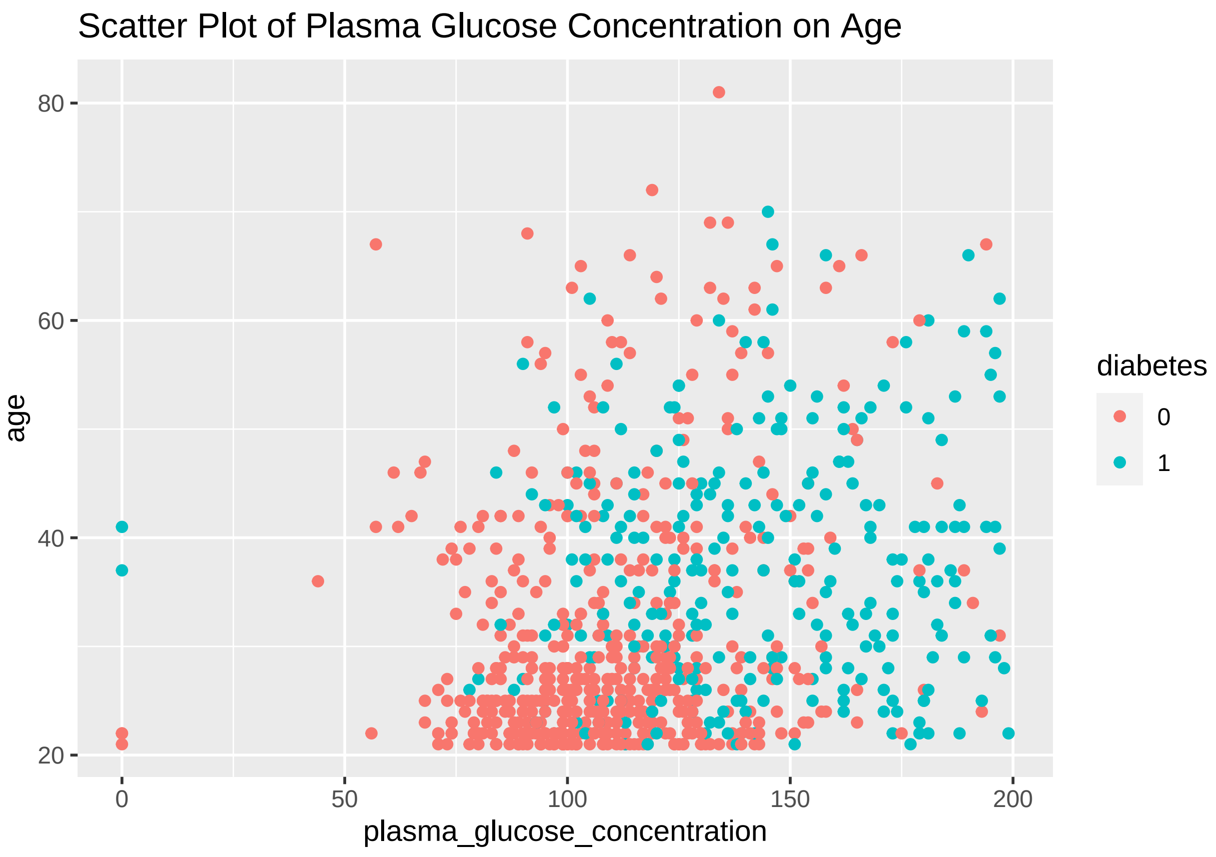
1.

Answer:

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

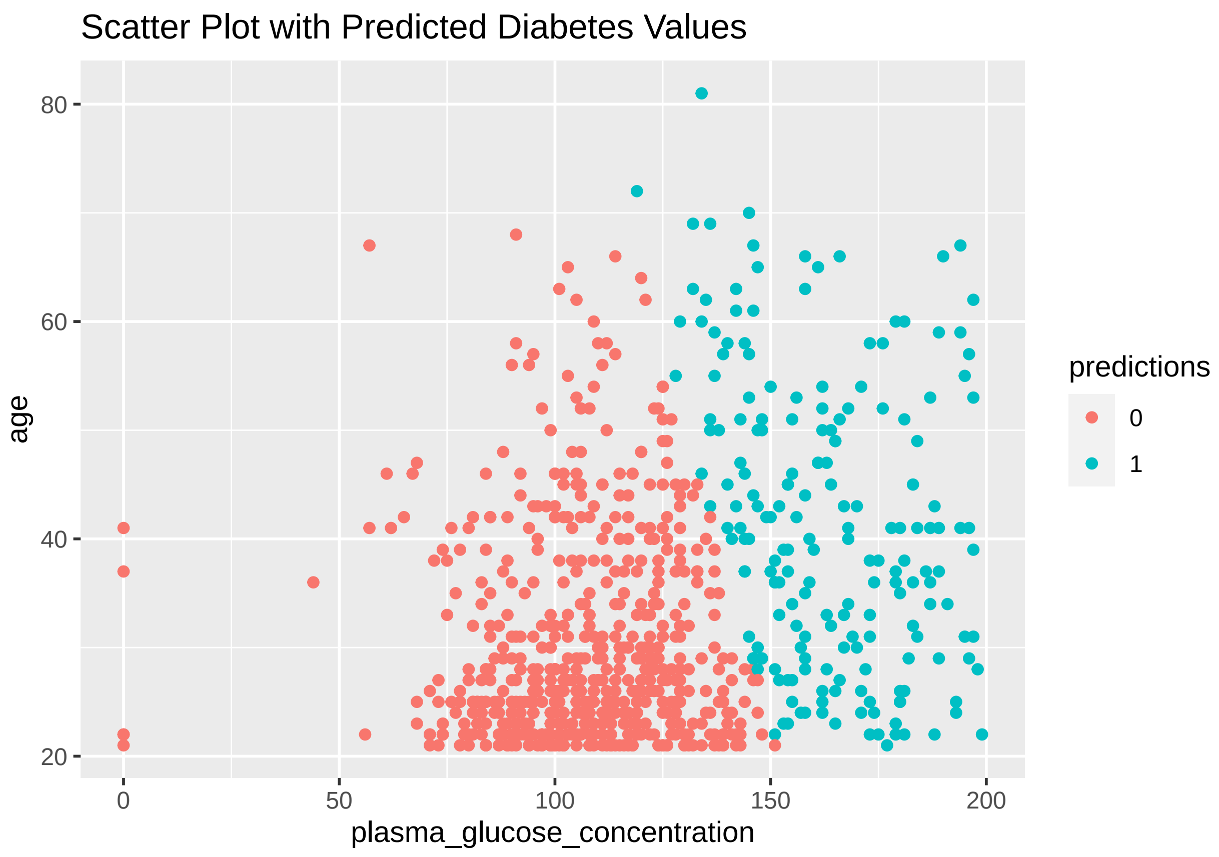


2.

Answer:

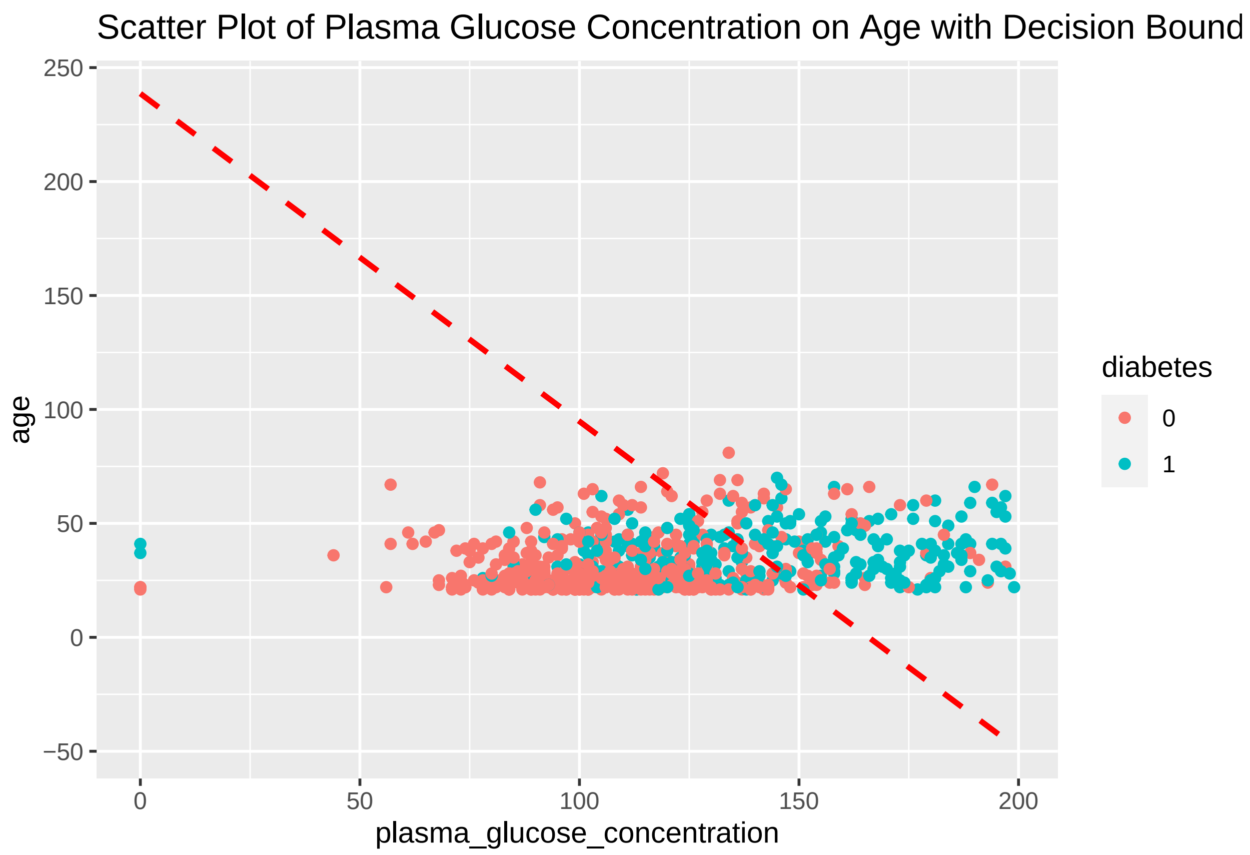
```
##
## Call:
## glm(formula = diabetes ~ plasma_glucose_concentration + age,
##     family = binomial, data = pima_data)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3367  -0.7775  -0.5087   0.8367   3.1630
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -5.912449   0.462620  -12.78  < 2e-16 ***
## plasma_glucose_concentration  0.035644   0.003290   10.83  < 2e-16 ***
## age                           0.024778   0.007374    3.36 0.000778 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.48  on 767  degrees of freedom
## Residual deviance: 797.36  on 765  degrees of freedom
## AIC: 803.36
##
## Number of Fisher Scoring iterations: 4

## Probability(Diabetes=1) = 1 / (1 + exp(-(intercept + coef1*x1 + coef2*x2)))

## Misclassification Error: 0.2630208
```



3.

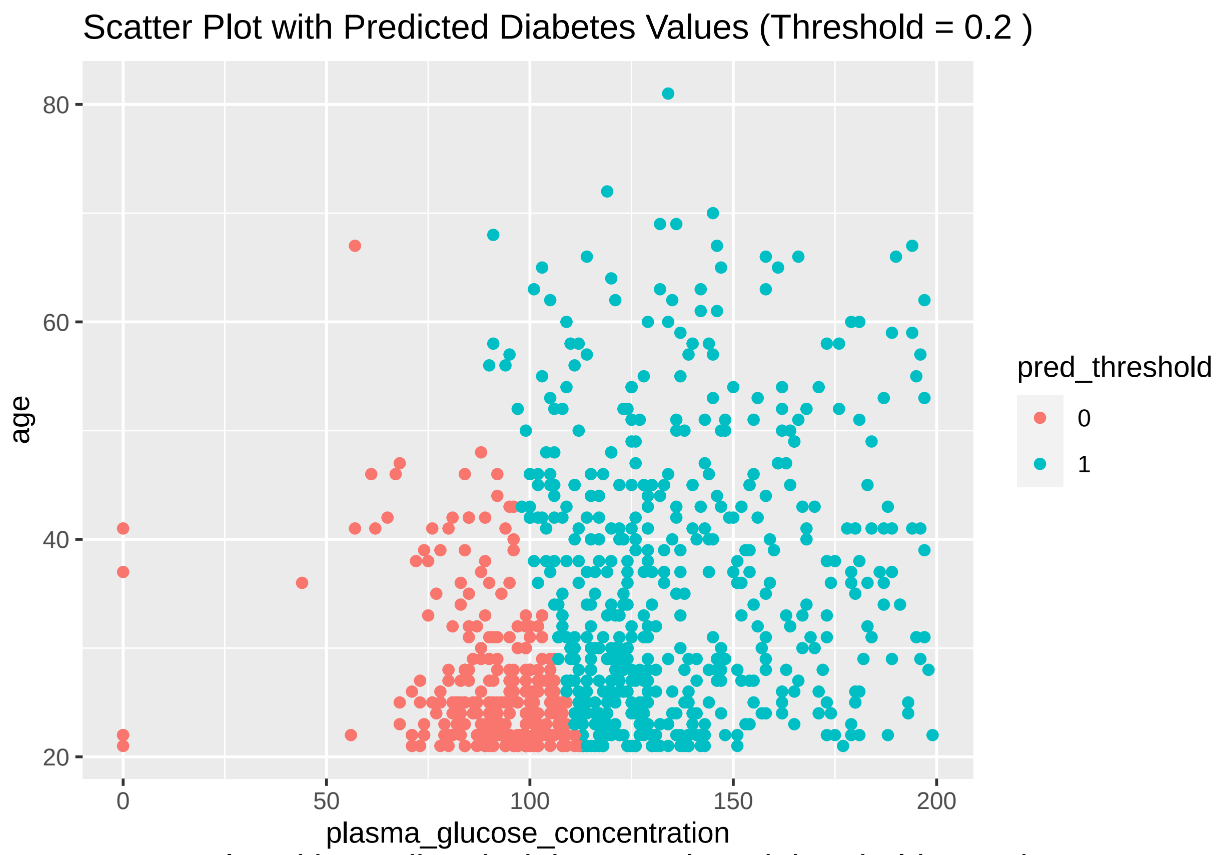
Answer:

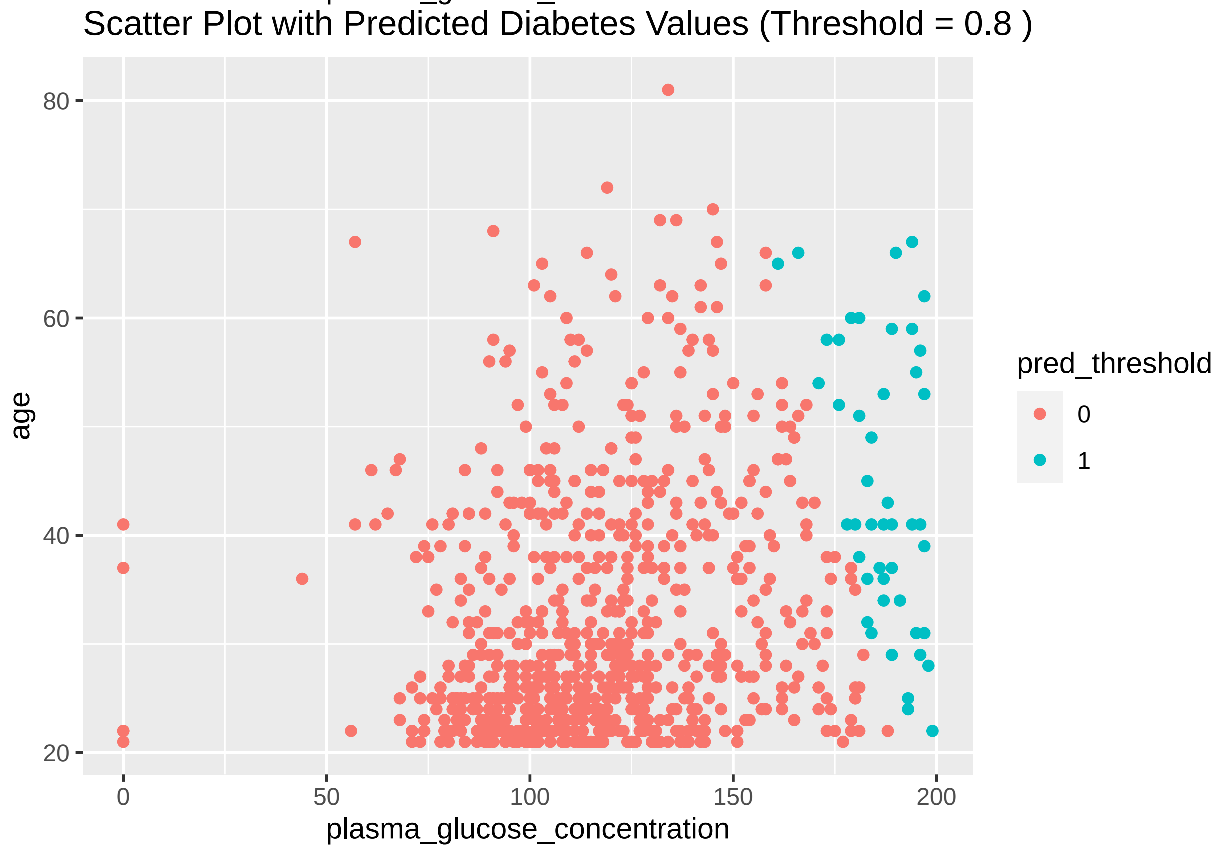


Scatter Plot of Plasma Glucose Concentration on Age with Decision Bound
250
200
150
100
50
0
−50
age
0
50
100
150
200
plasma_glucose_concentration
diabetes
0
1

4.

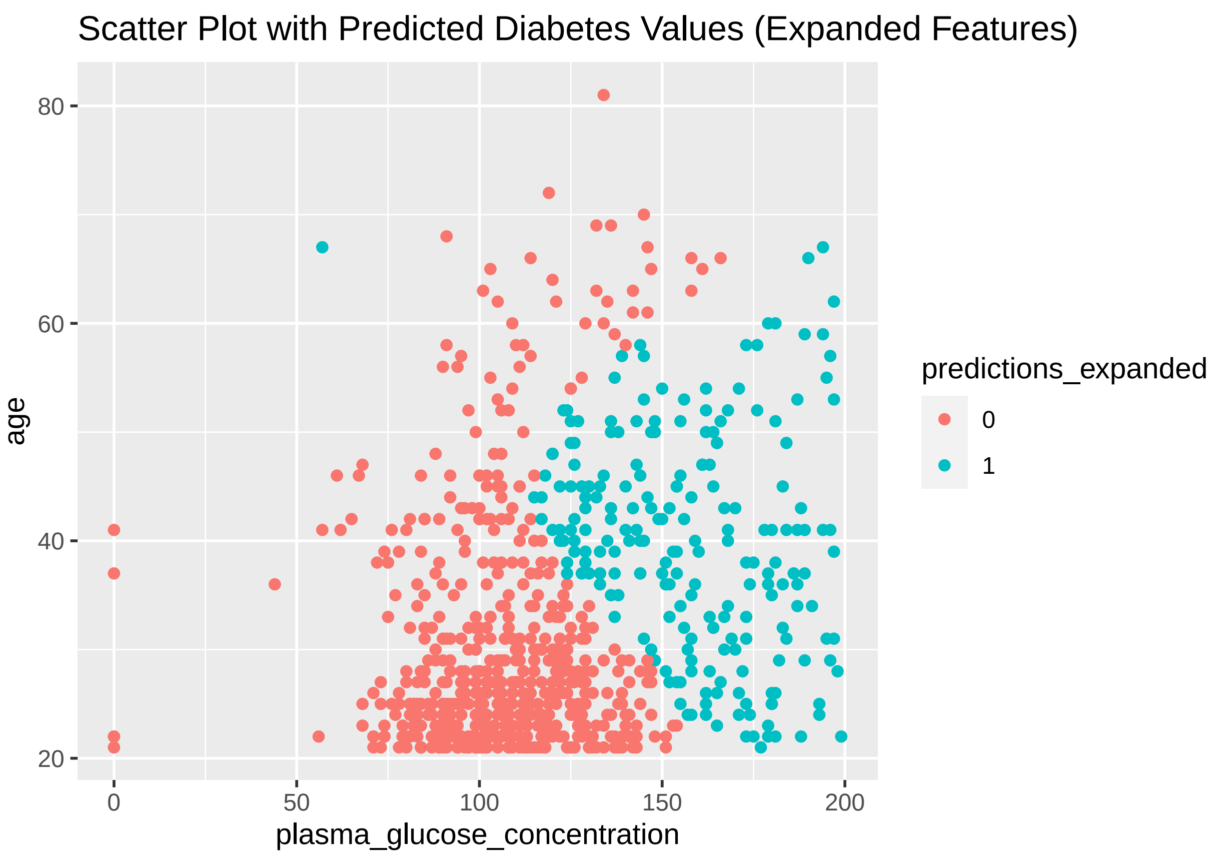
Answer:


Scatter Plot with Predicted Diabetes Values (Threshold = 0.2)
age
80
60
40
20
0
50
100
150
200
plasma_glucose_concentration
pred_threshold
0
1


Scatter Plot with Predicted Diabetes Values (Threshold = 0.8)
age
80
60
40
20
0
50
100
150
200
plasma_glucose_concentration
pred_threshold
0
1

5.

Answer:



```
## Misclassification Error (Expanded Features): 0.2447917
```

Appendix:

knearest.R

```
# load necessary libraries
library(ggplot2)
library(kknn)

# import data set
optdigits_data <- read.csv('optdigits.csv', header = FALSE)
colnames(optdigits_data) <- c(paste0("a",1:64),"digit")
optdigits_data$digit <- as.factor(optdigits_data$digit)
#head(optdigits_data, 5)

n=dim(optdigits_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train_data=optdigits_data[id,]
id1=setdiff(1:n, id)
id2=sample(id1, floor(n*0.25))
valid_data=optdigits_data[id2,]
id3=setdiff(id1,id2)
test_data=optdigits_data[id3,]
```

```r
# 30-nearest neighbor classification
k_fit_train <- kknn(formula = digit ~ ., train_data, train_data, k = 30, kernel = "rectangular")
k_fit_test <- kknn(formula = digit ~ ., train_data, test_data, k = 30, kernel = "rectangular")

# Confusion matrices for the training and test data
train_confusion <- table(train_data$digit, fitted(k_fit_train))
cat("Confusion Matrix for Training Data:\n")
print(train_confusion)

test_confusion <- table(test_data$digit, fitted(k_fit_test))
cat("Confusion Matrix for Test Data:\n")
print(test_confusion)

# Misclassification errors for the training and test data
train_error <- 1 - sum(diag(train_confusion)) / sum(train_confusion)
cat("Misclassification errors for the training data are: ", train_error, "\n" )

test_error <- 1 - sum(diag(test_confusion)) / sum(test_confusion)
cat("Misclassification errors for the test data are: ", test_error, "\n" )


# the quality of predictions for different digits
for ( i in 1:nrow(test_confusion)) {
  digit_accuracy <- test_confusion[i,i] / sum(test_confusion[i,])
  cat("The accuracy of prediction of for digit ", i-1, " is: ", digit_accuracy, "\n")
}

overall_accuracy <- sum(diag(test_confusion)) / sum(test_confusion)
cat("The overall accuracy of prediction is:", overall_accuracy, "\n")

# Get probabilities of class "8"
probabilities <- k_fit_train$prob[,"8"]

# Get indices of training data for class "8"
indices_8 <- which(train_data$digit == "8")

# Get probabilities for class "8"
probabilities_8 <- probabilities[indices_8]

# Find 2 easiest (highest probability) and 3 hardest (lowest probability) to classify cases
easiest_indices <- indices_8[order(probabilities_8, decreasing = TRUE)[1:2]]
hardest_indices <- indices_8[order(probabilities_8)[1:3]]

# Reshape features as 8x8 matrix and visualize
for (index in c(easiest_indices, hardest_indices)) {
  digit_8 <- matrix(as.numeric(train_data[index, 1:64]), nrow = 8, byrow = TRUE)
  heatmap(digit_8, Colv = NA, Rowv = NA, main = paste("Digit '8', Index:", index))
}

# Fit KNN for different K values and plot errors
errors <- data.frame()
for (k in 1:30) {
  fit <- kknn(digit ~ ., train_data, valid_data, k = k, kernel = "rectangular")
  pred <- fit$fitted.values
  confusion_matrix <- table(valid_data$digit, pred)
  error <- 1 - sum(diag(confusion_matrix)) / sum(confusion_matrix)
  errors <- rbind(errors, data.frame(K = k, Error = error))
}
# Plot the misclassification errors on the value of K
plot(errors$K, errors$Error, type = "b", col='orange', main="Misclassification errors on the value of K

# Initialize a data frame to store the results
ce_errors <- data.frame(k_value = integer(), cross_entropy_error = numeric())

for (k in 1:30) {
  # Fit K-nearest neighbor classifier
  fit <- kknn(digit ~ ., train_data, valid_data, k = k, kernel = "rectangular")

  # Get predicted probabilities
  predicted_probabilities <- fit$prob

  # Compute cross-entropy error
  #actual_probabilities <- ifelse(valid_data$digit == "8", 1, 0)
  #cross_entropy <- -sum(actual_probabilities * log(predicted_probabilities + 1e-15))
  cross_entropy <- -sum(as.numeric(valid_data$digit) * log(predicted_probabilities + 1e-15))

  ce_errors <- rbind(ce_errors, data.frame(k_value = k, cross_entropy_error = cross_entropy))
}

# Plot the results
plot(ce_errors$k_value, ce_errors$cross_entropy_error, type = "b", col='orange', main="Cross Entropy er

# Find the optimal K
optimal_k <- ce_errors$k_value[which.min(ce_errors$cross_entropy_error)]
cat("The optimal k is:", optimal_k, "\n")
```

linear_ridge.R

```
# import data set
parkinson_data <- read.csv('parkinsons.csv', header = TRUE)
#head(parkinson_data, 5)

# Split the data into training and test sets
n=dim(parkinson_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train_data=parkinson_data[id,]
test_data=parkinson_data[-id,]

# Scale the data
scaled_train_data <- as.data.frame(cbind(scale(subset(train_data, select = -c(motor_UPDRS))), motor_UPD
scaled_test_data <- as.data.frame(cbind(scale(subset(test_data, select = -c(motor_UPDRS))), motor_UPDRS

# linear regression model
lm_model <- lm(motor_UPDRS ~ ., data = scaled_train_data)

# Training and test MSE
```

```r
train_pred <- predict(lm_model, newdata = scaled_train_data)
train_mse <- mean((scaled_train_data$motor_UPDRS - train_pred)^2)
cat("The training data's MSE is: ", train_mse)

test_pred <- predict(lm_model, newdata = scaled_test_data)
test_mse <- mean((scaled_test_data$motor_UPDRS - test_pred)^2)
cat("The training data's MSE is: ", test_mse)

# Significant variables
summary(lm_model)

# Log-likelihood function
log_likelihood_fun <- function(theta, sigma, y, X) {
  n <- length(y)
  log_likelihood <- -n/2 * log(2*pi*sigma^2) - 1/(2*sigma^2) * sum((y - X%*%theta)^2)
  return(log_likelihood)
}

# Ridge log-likelihood function
ridge_log_Likelihood_fun <- function(theta, sigma, lambda, y, X) {
  log_like <- log_likelihood_fun(theta, sigma, y, X)
  ridge_penalty <- lambda * sum(theta^2)
  return(-log_like + ridge_penalty)
}

# Ridge log-likelihood optimization function
ridge_log_likelihood_opt <- function(lambda, y, X) {
  start <- c(rep(0, ncol(X)), sd(y)) # combine theta and sigma into one numeric vector
  theta_length <- ncol(X)

  # Define a function for the optim() call
  fn_to_optim <- function(params) {
    theta <- params[1:theta_length]
    sigma <- params[theta_length + 1]
    return(-ridge_log_Likelihood_fun(theta, sigma, lambda, y, X))
  }

  # Call optim()
  opt_res <- optim(start, fn = fn_to_optim, method = "BFGS")

  # Return theta and sigma separately
  return(list(theta = opt_res$par[1:theta_length], sigma = opt_res$par[theta_length + 1]))
}

# Degrees of freedom function
df_fun <- function(lambda, X) {
  H <- solve(t(X) %*% X + lambda * diag(ncol(X))) %*% t(X) %*% X
  return(sum(diag(H)))
}

y <- scaled_train_data$motor_UPDRS
X <- as.matrix(subset(scaled_train_data, select = -c(motor_UPDRS)))
# Ridge optimization for different lambdas
```

```r
lambdas <- c(1, 100, 1000)
opt_params <- lapply(lambdas, function(lambda) ridge_log_likelihood_opt(lambda, y, X))


# Predictions and MSE for different lambdas
ridge_train_preds <- lapply(1:length(opt_params), function(i) {
  theta <- opt_params[[i]]$theta
  as.matrix(subset(scaled_train_data, select = -c(motor_UPDRS))) %*% theta
})

#train_mses <- colMeans((scaled_train_data$motor_UPDRS - ridge_train_preds)^2)
ridge_train_mse <- sapply(ridge_train_preds, function(pred) mean((pred - scaled_train_data$motor_UPDRS)
cat("Ridge Training MSE:", ridge_train_mse, "\n")

ridge_test_preds <- lapply(1:length(opt_params), function(i) {
  theta <- opt_params[[i]]$theta
  as.matrix(subset(scaled_test_data, select = -c(motor_UPDRS))) %*% theta
})

ridge_test_mse <- sapply(ridge_test_preds, function(pred) mean((pred - scaled_test_data$motor_UPDRS)^2)
cat("Ridge Test MSE:", ridge_test_mse, "\n")

# Degrees of freedom for different lambdas
degrees_of_freedom <- sapply(lambdas, function(lambda) df_fun(lambda, X))
cat("Degrees of Freedom:", degrees_of_freedom, "\n")
```

logistic.R

```r
library(ggplot2)
library(caret)

# import data set
pima_data <- read.csv('pima-indians-diabetes.csv', header = FALSE)
colnames(pima_data) <- c("num_of_pregnant", "plasma_glucose_concentration", "blood_pressure",
                         "skinfold_thickness", "serum_insulin", "bmi",
                         "diabetes_predigree", "age",  "diabetes")
#head(pima_data, 5)


# Scatterplot
ggplot(pima_data, aes(x=plasma_glucose_concentration, y=age, color=as.factor(diabetes))) +
  geom_point() +
  labs(color="diabetes", title='Scatter Plot of Plasma Glucose Concentration on Age')

# Logistic Regression
logsitic_model <- glm(diabetes ~ plasma_glucose_concentration + age, data=pima_data, family=binomial)
summary(logsitic_model)

# Predict probabilities
probabilities <- predict(logsitic_model, type="response")
cat("Probability(Diabetes=1) = 1 / (1 + exp(-(intercept + coef1*x1 + coef2*x2)))")

# Classify observations
predictions <- ifelse(probabilities >= 0.5, 1, 0)
```

```r
# Compute misclassification error
mis_error <- mean(predictions != pima_data$diabetes)
cat("Misclassification Error:", mis_error, "\n")

# Scatter plot with predicted values
ggplot(data.frame(pima_data, predictions), aes(x=plasma_glucose_concentration, y=age, color=as.factor(pr
  geom_point() +
  labs(color="predictions",title='Scatter Plot with Predicted Diabetes Values')

# Decision boundary equation
# Decision boundary is where the logistic function equals 0.5
# 0 = intercept + coef1*x1 + coef2*x2
# x2 = -(intercept + coef1*x1) / coef2

#decision_boundary <- -(coef(logsitic_model)[1] + coef(logsitic_model)[2]*pima_data$age) / coef(logsiti
decision_boundary_x1 <- pima_data$plasma_glucose_concentration
decision_boundary_x2 <- -(coef(logsitic_model)[1] + coef(logsitic_model)[2]*decision_boundary_x1) / coe

# Add decision boundary to scatter plot
ggplot(pima_data, aes(x=plasma_glucose_concentration, y=age, color=as.factor(diabetes))) +
  geom_point() +
  geom_line(aes(x=decision_boundary_x1, y=decision_boundary_x2), color='red', linetype='dashed', linewi
  labs(color="diabetes", title='Scatter Plot of Plasma Glucose Concentration on Age with Decision Bound


# Predictions with different thresholds
thresholds <- c(0.2, 0.8)

for ( threshold in thresholds) {
  pred_threshold <- ifelse(predict(logsitic_model, newdata=pima_data, type='response') >= threshold, 1,

  # Scatter plot with predicted values and threshold
  scatter_plot <- ggplot(data.frame(pima_data, pred_threshold), aes(x=plasma_glucose_concentration, y=a
    geom_point() +
    labs(color="pred_threshold", title=paste('Scatter Plot with Predicted Diabetes Values (Threshold ='
  print(scatter_plot)
}



# Create new features
pima_data$z1 <- pima_data$plasma_glucose_concentration^4
pima_data$z2 <- pima_data$plasma_glucose_concentration^3 * pima_data$age
pima_data$z3 <- pima_data$plasma_glucose_concentration^2 * pima_data$age^2
pima_data$z4 <- pima_data$plasma_glucose_concentration * pima_data$age^3
pima_data$z5 <- pima_data$age^4

# Logistic Regression with new features
model_expanded <- glm(diabetes ~ plasma_glucose_concentration + age + z1 + z2 + z3 + z4 + z5, data=pima

# Predict probabilities and classify observations
probabilities_expanded <- predict(model_expanded, type="response")
predictions_expanded <- ifelse(probabilities_expanded >= 0.5, 1, 0)
```

```
# Scatter plot for the model with expanded features
ggplot(data.frame(pima_data, predictions_expanded), aes(x=plasma_glucose_concentration, y=age, color=as
  geom_point() +
  labs(color="predictions_expanded",title='Scatter Plot with Predicted Diabetes Values (Expanded Featur

# Compute misclassification error
mis_error_expanded <- mean(predictions_expanded != pima_data$diabetes)
cat("Misclassification Error (Expanded Features):", mis_error_expanded, "\n")
```