Lab 1 report

Group B13: Aron(aroen488), Sergey(servo519), Shahin(mdpa888)

2025-11-15

Assignment 1. Handwritten digit recognition with K-nearest neighbors.

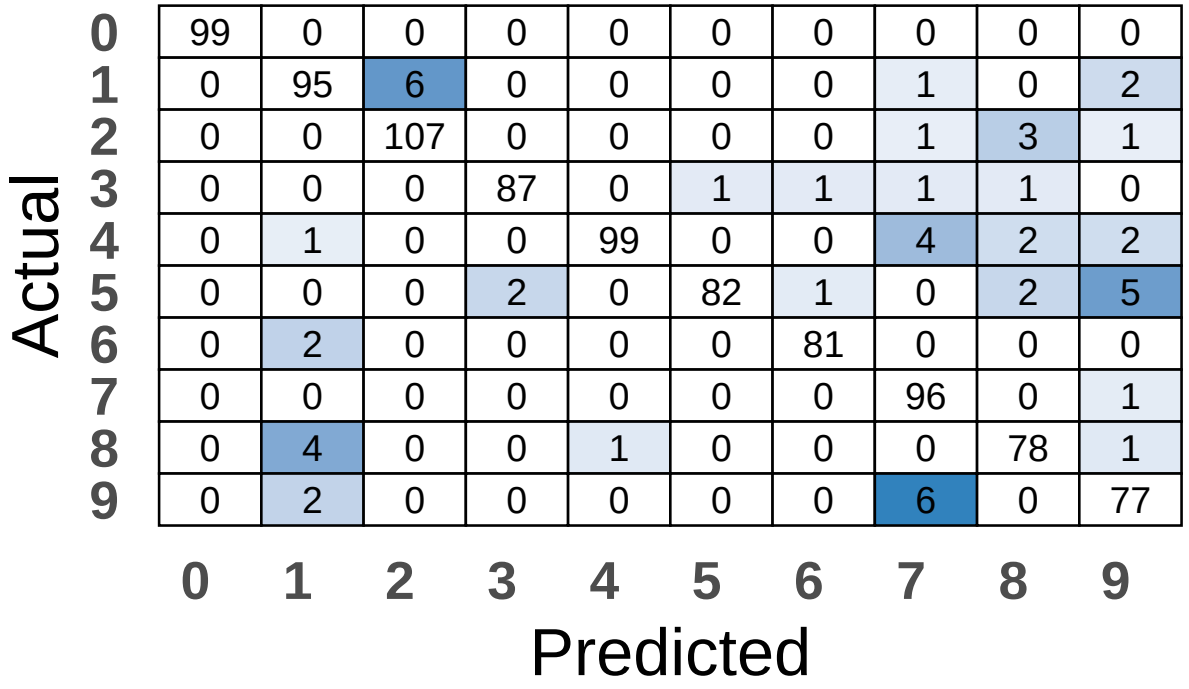
1. A dataset is given with handwritten numbers. These are originally 32x32. The information is compressed by doing a 4x4 convolution that sums the pixels. This results in 8x8 = 64 variables. The aim of this assignment is to classify the handwritten numbers by identifying clusters in these variables. Before analysis the data is divided into 50% training data, 25% validation data and 25% test data according to the slides.

2. KNN is fit using the following code:

Missclassification rate = 0.04

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	177	0	0	0	1	0	0	0	0	0
1	0	174	9	0	0	0	1	0	1	3
2	0	0	170	0	0	0	0	1	2	0
3	0	0	0	197	0	2	0	1	0	0
4	0	1	0	0	166	0	2	6	2	2
5	0	0	0	0	0	183	1	2	0	11
6	0	0	0	0	0	0	200	0	0	0
7	0	1	0	1	0	1	0	192	0	0
8	0	10	0	1	0	0	2	0	190	2
9	0	3	0	4	2	0	0	2	4	181

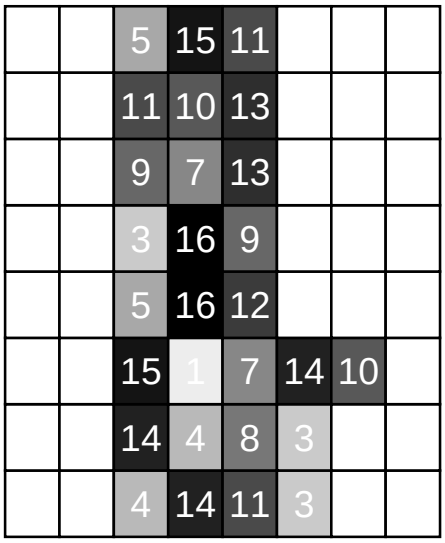
Missclassification rate = 0.06

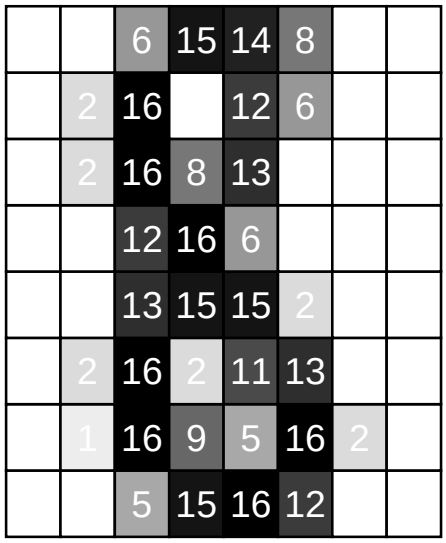


Overall the missclassification rate is low in both models, it is possible to do better and it would be context dependent. Say the number detection is for a banking application, then 6% error is unacceptable. In the training set the model predicts 1 and 9 too often, 1s were predicted when it was an 8 and 9s were predicted when it was a 5. The same errors are present in the validation set but here the model also predicts 7s when the actual number is a 9.

3. Here we look at examples visually of 8s in that were hard and easy for the model to classify. Keep in mind that the displayed images are after the convolution which makes it harder for humans to interpret perfectly.

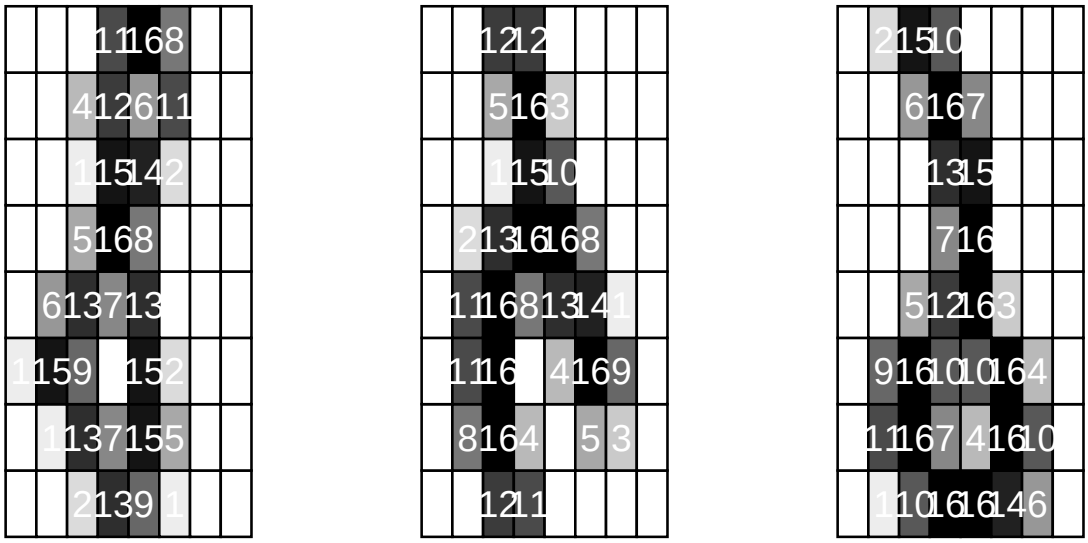
Easy 8s





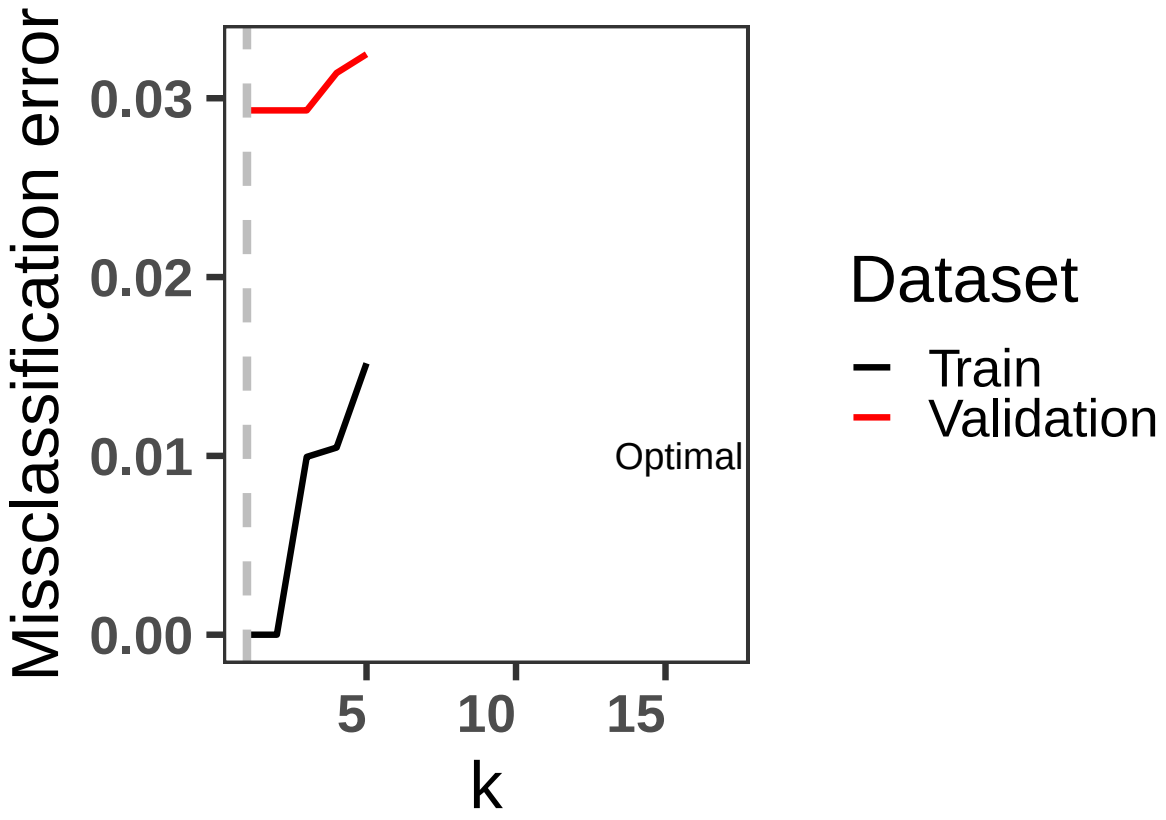
These look like eights, we clearly see two round shapes, on the right image we even see holes in the middle of the convolutions, the left image must have a smaller ring above.

Hard 8s



These had probability 0.33, 0.37 and 0.37 of being 8s, but 8 had the highest probability. We see that the above ring resembles more a line and that it could be close to a 6, all of these images look slightly rotated and the 2 rightmost images are hard to classify visually. Maybe the model could benefit from augmented data with rotations.

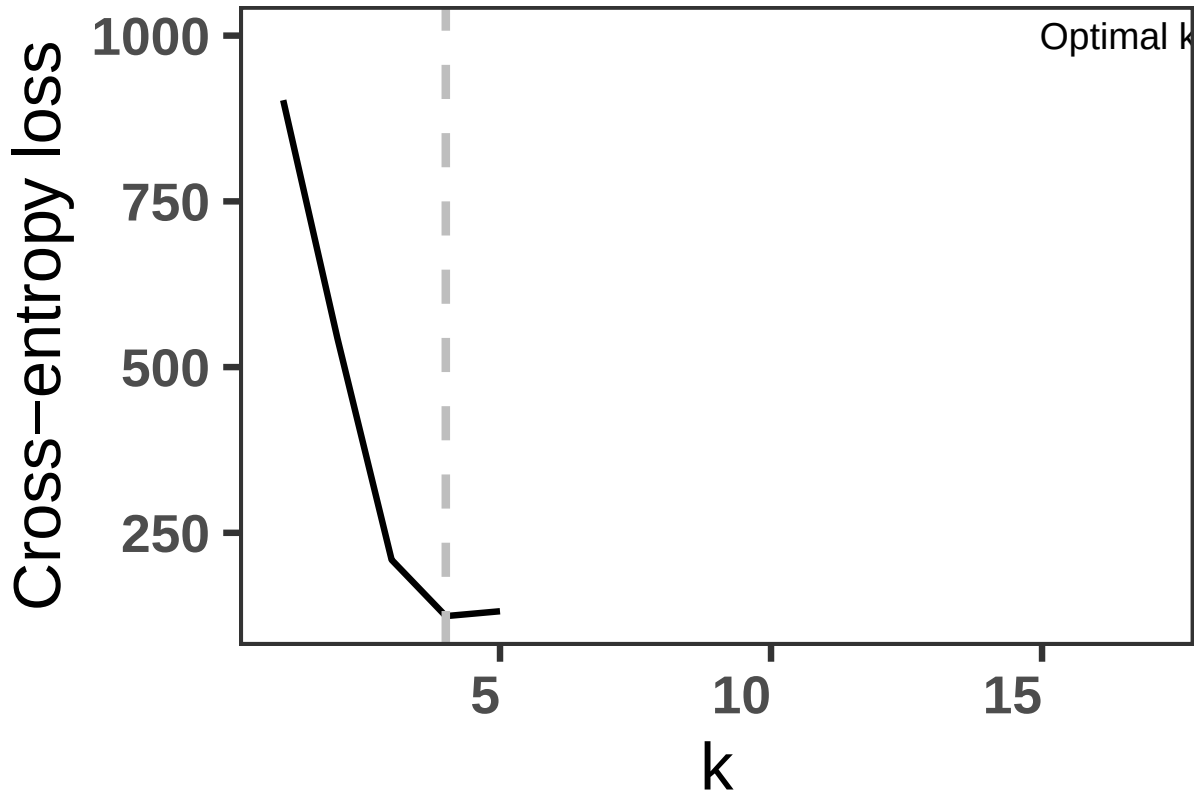
4. Lets try to optimize k by looking at the validation error.



For small K we see that the model is extremely overfit in the training set. If k=1 every point will be classified as itself. But the model won't generalize well. If K is too large the model will take a larger neighboorhood into consideration. The issue with that could be that the model cannot pick up the 5s that look similar to 9s that we mentioned earlier. We can't fit to small details like this, model complexity drops. Looking at the errors we see that the validation error is basically identical for $k < 10$. Even with $k = 1$ which is interesting, the prediction would be the closest number in the training set then. For higher k we start missclassifying numbers that are similar to eachother.

The missclassification error in the test set is 4%, slightly higher than in the validation set. The model seems to generalize assuming the test set is a fair characterization of what the model will encounter in the real world.

5. Optimizing k using cross-entropy loss instead of missclassification error in the validation set.



The multinomial pdf is proportional in p_i to

$$\prod_{c=1}^{C} p_c^{x_c}$$

where C is the amount of classes and x_c is the amount of occurrences of the class c in the sample. If we maximize the likelihood for p, equivalent to maximizing the log-likelihood we get the following.

$$\ell(p) = constant + \sum_{c=1}^{C} x_c log(p_c)$$

Which is similar to minimizing the cross entropy

$$C.E. = -\sum_{i=1}^{n}\sum_{c=1}^{C} \mathbf{1}_{\{\text{observation i is of class c}\}} log(\hat{p_c}(data))$$

If we say Y|data is multinomial with $\vec{p} = p|\vec{data}$ then it maps one to one to cross entropy, as we will have n samples from the multinomial with x_c for each being 1. This is more reasonable in this case as the first example ignored all the covariates that are available.

Assignment 2. Linear regression and ridge regression

In the first step, we load and scale the data and perform checks to make sure the data has been scaled and normalized. We then split the data into train and test datasets at the ratio of 60/40.

```
## [1] "Train data size:  3525"
```

```
## [1] "Test data size:  2350"
```

We then fit a non-optimized linear regression model as a sanity check. We do use the intercept as one of the coefficients in our model as that represents the value of the target variable when all the input variables are at their mean values. We also extract the list of significant input variables from the summary of the lm function in R which are the coefficients with p-values below a certain arbitrary small threshold e.g. 0.05.

```
## [1] "Train MSE:  57.7021129525355"
```

```
## [1] "Test MSE:  61.4395689573022"
```

```
## [1] "Features contributing significantly to the model: "
```

```
## [1] "Jitter.Abs."   "Shimmer"       "Shimmer.APQ5"  "Shimmer.APQ11"
## [5] "NHR"           "HNR"           "DFA"           "PPE"
```

Based on the assumption that the residuals of the linear regression function are normally distributed (this might need to be checked if it turns out that the model provides a poor fit to the data i.e. high test MSE but is outside the scope of this assignment), the log-likelihood formula we use is as follows:

$$\mathcal{L}(\boldsymbol{\theta}, \sigma \mid X, y) = -\frac{n}{2}\log(2\pi) - n\log(\sigma) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left[y_i - (\theta_0 + \sum_{j=1}^{p} x_{ij}\theta_j)\right]^2$$

where

- y_i: The observed value of the target (dependent, or response) variable for observation i (e.g., for your data, this could be a specific `motor_UPDRS` score).
- x_{ij}: The value of the j^{th} predictor (independent variable, feature) for observation i. Each x_{ij} is a value drawn from your data table's predictor columns.
- θ_0: The intercept term, representing the expected value of y when all predictors are zero.
- θ_j: The coefficient for predictor j, quantifying the change in y per unit change in x_{ij}, holding other variables fixed.
- p: The number of predictor variables (not counting the intercept).
- n: The total number of observations in the data set (rows of your data).
- $\boldsymbol{\theta}$: The vector of all coefficients, including the intercept $(\theta_0, \theta_1, \ldots, \theta_p)$.
- X: The design matrix, where each row is an observation and each column is a predictor variable ($n \times p$ matrix; often with an added first column of ones for the intercept).
- σ: The standard deviation of the residuals (errors), representing the typical size of the difference between observed values (y_i) and model predictions.

Based on the results obtained from fitting a non-optimized linear regression model above, the test MSE does not seem to be sufficiently higher than the train MSE and therefore there is no sign of the model overfitting the train data. However, since we are specifically asked to do so, we also perform ridge regression to reduce the complexity of the model.

The formula we use is

$$\text{Loss}(\boldsymbol{\theta}, \sigma, \lambda) = -\log P(y \mid \boldsymbol{\theta}, \sigma, X) + \lambda \sum_{j=0}^{p} \theta_j^2$$

where: - y is the vector of observed response values. - $\boldsymbol{\theta} = (\theta_0, \theta_1, ..., \theta_p)$ is the full coefficient vector (including intercept). - X is the $n \times p$ predictor matrix (not including the intercept column). - σ is the residual standard deviation. - $\lambda \geq 0$ is the penalty parameter controlling the amount of shrinkage. - $P(y \mid \boldsymbol{\theta}, \sigma, X)$ is the likelihood of the data under the linear regression model with normal errors.

We then perform the optimization of our ridge regression function using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm and also calculate the degrees of freedom of the resulting ridge-optimized model for various values of lambda. The degrees of freedom in ridge regression measure the effective number of parameters being used by the fitted model, after accounting for the shrinkage effect of the ridge penalty. Unlike standard

linear regression, where the degrees of freedom equal the number of predictors, ridge regression shrinks coefficients, reducing the model's flexibility and thus its effective degrees of freedom. Given a data matrix X (with n rows and p columns) and a ridge penalty parameter $\lambda \geq 0$,
let $d_1, d_2, \ldots, d_p$ be the singular values of X, obtained from its singular value decomposition (SVD).

The effective degrees of freedom for ridge regression are calculated as:

$$df(\lambda) = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda}$$

where

- X: The (centered and/or scaled) data matrix of predictors, with n samples and p features.
- λ: The ridge penalty parameter, controlling the amount of coefficient shrinkage (higher λ means stronger penalization and lower model flexibility).
- d_j: The j-th singular value of X (from SVD), which characterizes how much variance in the data is associated with each principal direction.
- $df(\lambda)$: The effective degrees of freedom; the sum reflects the impact of ridge regularization on the true dimensionality of the model fit.

The results of fitting the ridge-optimized model are summarized below:

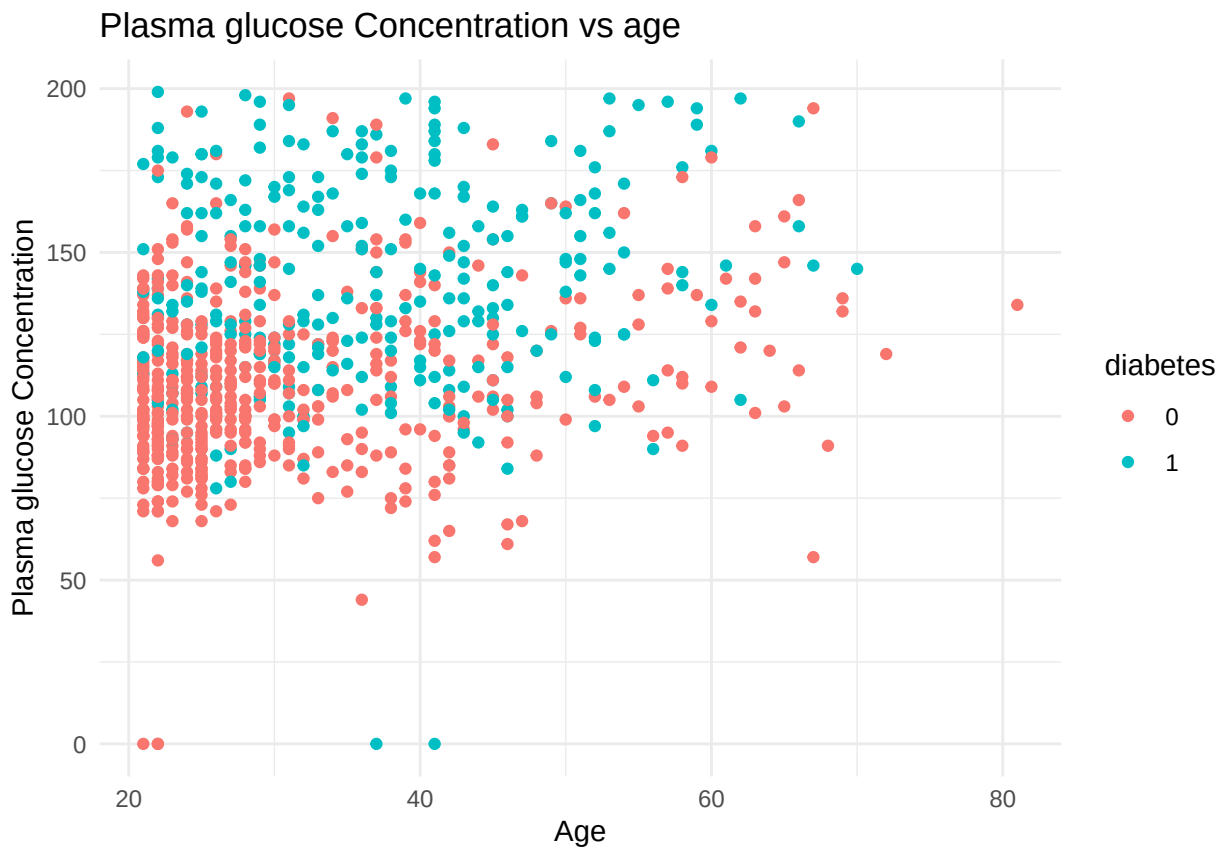
Table 1: Ridge Regression Results (Train/Test MSE and Degrees of Freedom)

Lambda	Train MSE	Test MSE	Degrees of Freedom
0.00	57.702	61.440	16.000
0.01	57.707	61.416	14.123
0.10	57.741	61.315	14.000
1.00	58.434	61.475	13.863
10.00	87.707	88.514	12.932
100.00	331.600	328.503	9.939
1000.00	494.413	490.146	5.643

From the table above, we can see that at λ equal to 0, we obtain the same values as for the non-optimized model which is used as a sanity check as in this case there is no coefficient shrinkage. We the observe a small improvement in test MSE at small values of λ such as 0.1 which indicates that a slight coefficient shrinkage can be beneficial. However, at high values of λ, we observe a significant growth of both the train and the test MSE which indicates that we have oversimplified our model and it can no longer adequately describe the data (underfitting).

Assignment 3. Logistic regression and basis function

3.1 Scatterplot showing a Plasma glucose concentration on Age



Comments

The scatterplot shows that glucose and age do relate to diabetes, but the two classes overlap a lot. This means it is not easy to classify diabetes using only these two variables with a simple logistic regression model.

3.2 Train a logistic regression model

```
## Misclassification rate (r = 0.5): 0.263
```

Probabilistic Model

$$p(x) = \frac{1}{1 + e^{-(-5.912449 + 0.035644 x_1 + 0.024778 x_2)}}$$

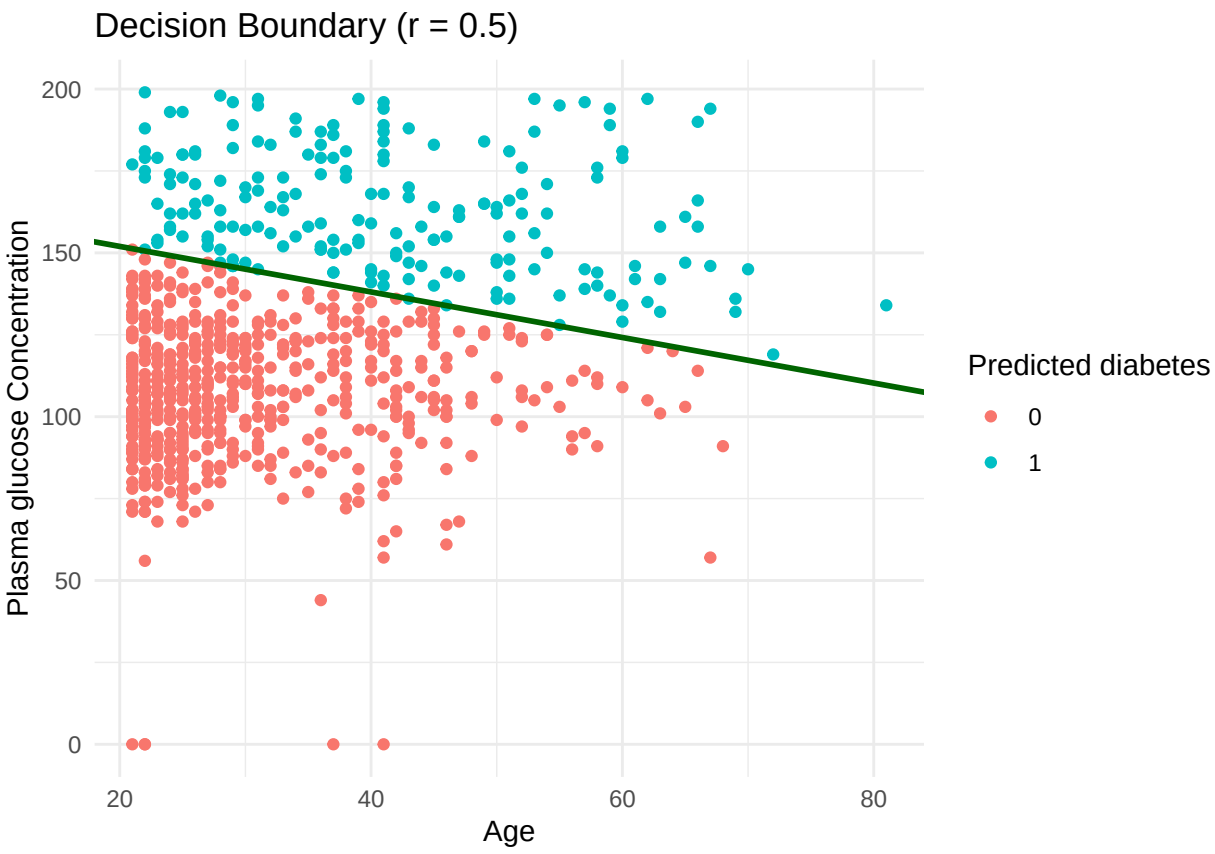
where x_1 = Plasma glucose concentration and x_2 = age.

Comments

The logistic regression model gives a misclassification rate of 0.263, meaning it gets about 74% of the predictions right. The model works reasonably well but makes many mistakes because the classes overlap. This shows that glucose and age alone do not fully separate diabetic and non-diabetic individuals.

3.3 Compute boundary line (r = 0.5)

```
## Equation of the decision boundary: glucose =  -0.6951611  * age +  165.8748
```

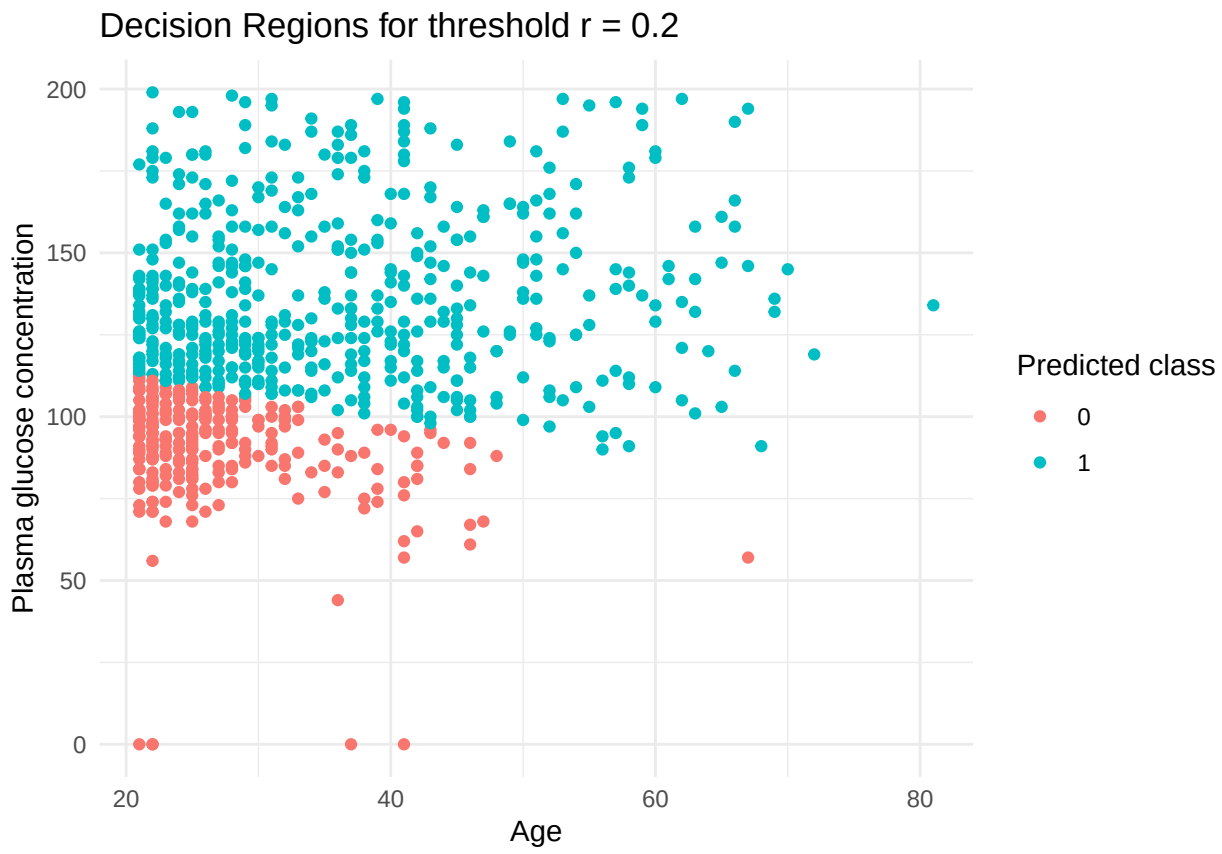


Comments

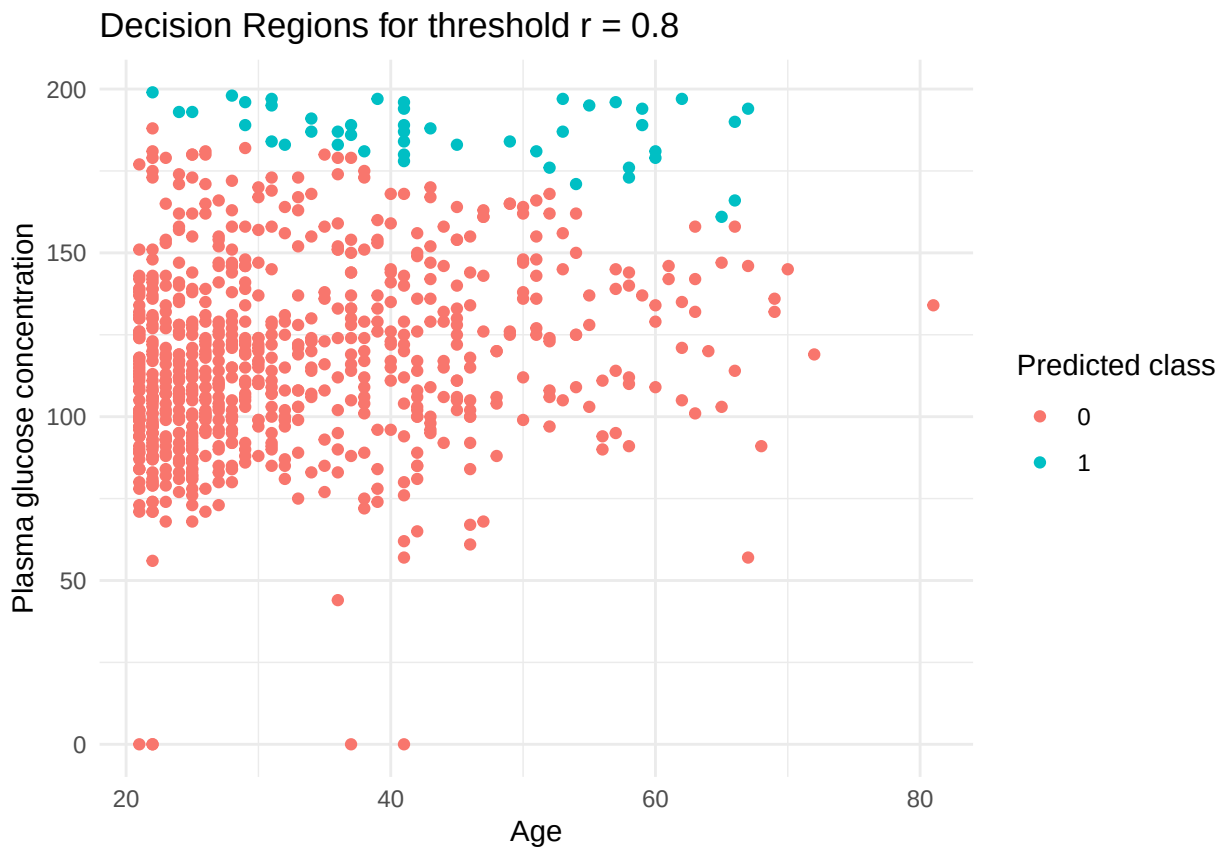
The decision boundary is a straight line, because logistic regression is linear. The line does not perfectly follow how the data is spread out. Many points fall on the wrong side of the boundary, which explains the moderate error rate.

3.4 Thresholds r = 0.2 and r = 0.8

```
## Misclassification rate (r = 0.2 ): 0.372
```




```
## Misclassification rate (r = 0.8 ): 0.315
```



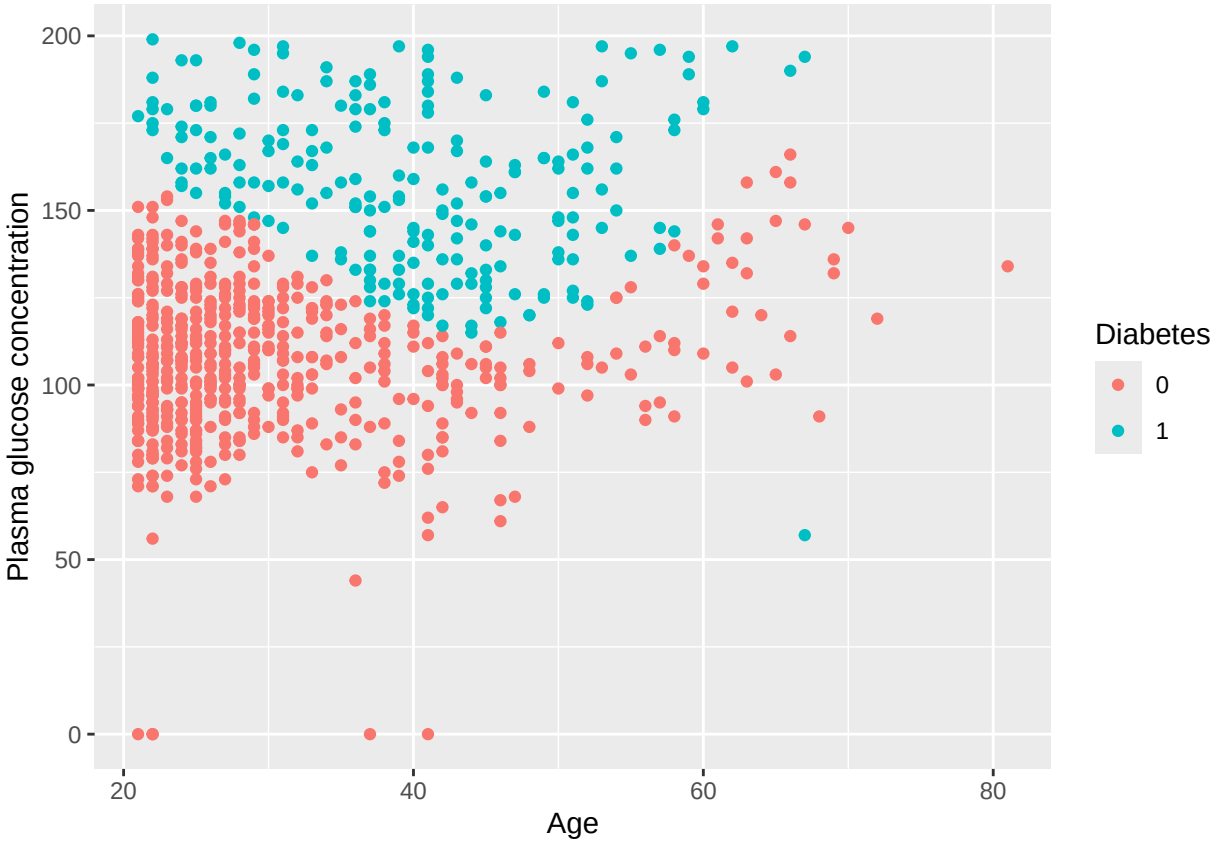
Comments

- With r = 0.2, the model predicts more diabetics (many false positives).
- With r = 0.8, the model predicts fewer diabetics (many false negatives).

Changing the threshold does not change the boundary shape but only how strict the model is when deciding between class 0 and class 1.

3.5 Basis Expansion (Nonlinear Logistic Regression)

```
## Misclassification rate (expanded model, r = 0.5): 0.245
```



Comments

After adding polynomial features, the misclassification rate improves slightly to 0.245. This model is better because it can create a curved, nonlinear decision boundary, which fits the data more closely. The improvement is small, but it shows that the relationship between age, glucose, and diabetes is not purely linear.

Assignment 4. Theory

Q1 Why can it be important to consider various probability thresholds in the classification problems, according to the book? The book (p.50) says that using the usual threshold of 0.5 only works well when both types of mistakes are equally bad and both classes appear equally often. In many real situations, one type of mistake is worse than the other (for example, missing a disease is worse than a false alarm). Sometimes one class is much rarer than the other, and 0.5 does not work well for that. Changing the threshold helps you decide which type of error you want to avoid more.

Q2 What ways of collecting correct values of the target variable for the supervised learning problems are mentioned in the book? The book (p.6) explains that in supervised learning, the training labels come from a "supervisor." This supervisor can be a human domain expert, such as a doctor labeling ECG data as normal or abnormal. But the book also says that the supervisor does not always have to be a human. Labels may come from automated computations, such as physics simulations (DFT) used to compute formation energies. In other situations, the labels come naturally from real-world outcomes, such as the result of a soccer match or the sale price of an apartment.

Q3 How can one express the cost function of the linear regression in the matrix form, according to the book?

The book *(p.41)* shows that the cost of linear regression can be written in matrix form as:

$$\frac{1}{n}\|X\theta - y\|^2.$$

This means that all input data are placed into one large matrix X, all outputs are placed into a vector y, and the model parameters are stored in a vector θ. Writing the model this way makes it easier to compute the best parameters using linear algebra. It also leads to the normal equation:

$$X^T X\theta = X^T y,$$

which provides a direct solution for the optimal θ.

Statement of Contribution

Assignment 1 was mainly contributed by Aron. Assignment 2 was mainly contributed by Sergey. Assignment 3 was mainly contributed by Shahin. The theory was mainly done by Shahin and Sergey. Formatting the report was mainly done by Aron.

Appendix

```
########################
#
# TASK 1 CODE
#
########################
data <- read.csv("data/optdigits.csv")

library(dplyr)
library(ggplot2)
library(kknn)
library(reshape2)
library(tidyverse)
library(ggpubr)


#sapply(1:ncol(data), function(x) data[, x] %>% unique() %>% length())


colnames(data)[65] <- "y"
data$y <- data$y %>% as.factor()


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

# the ids not in id
id1=setdiff(1:n, id)

set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=data[id2,]
```

```r
id3=setdiff(id1,id2)
test=data[id3,]
# the rest is test

model <- kknn(y ~ ., train = train, test = valid, k = 30, kernel = "rectangular")
y_hat_v <- model$fitted.values

model <- kknn(y ~ ., train = train, test = train, k = 30, kernel = "rectangular")
y_hat_t <- model$fitted.values


conf_table <- function(y, y_hat, decimals = 2, p_or_r) {

  cm <- table(Number = y, Predict = y_hat)

  C <- nrow(cm)
  # I wanna do relative coloring, based on deviation
  rowsums <- sapply(1:C, function(x) sum(cm[x, ]))
  perfect_pred <- diag(rowsums)

  diff <- cm - perfect_pred

  # If one row has more examples than another we need to scale to see devation
  for (i in 1:nrow(diff)) {
    diff[i, ] <- diff[i, ] / rowsums[i]

  }

  # I dont care about the diagonal, the most interesting is that the errors pop out
  diff[diff < 0] <- 0


  cm_long <- melt(cm)
  cm_long$Number <- cm_long$Number[nrow(cm_long):1] %>% as.factor()
  cm_long$Predict <- cm_long$Predict %>% as.factor()

  cm_long$color <- melt(diff)$value



  ggplot(cm_long, aes(x = Predict, y = Number, fill = color)) +
    geom_tile(color = "black") +
    geom_text(aes(label = value), color = "black", size = 5) +
    scale_fill_gradient(low = "white", high = "#3182bd", guide = "none") +
    theme_minimal(base_size = 25) +
    scale_y_discrete(labels = levels(cm_long$Number) %>% rev() ) +
    labs(x = "Predicted", y = "Actual", title = paste0("Missclassification rate = ",
                                                       (1 - mean(y == y_hat)) %>%
                                                         round(decimals)) ) +
    theme(
      axis.text.x = element_text(hjust = 1, face = "bold"),
      axis.text.y = element_text(face = "bold"),
      plot.title = element_text(hjust = 0.5, face = "bold"),
```

```
      panel.grid = element_blank(),
    )
}

conf_table(train$y, y_hat_t)

conf_table(valid$y, y_hat_v)


model <- kknn(y ~ ., train = train, test = train, k = 30, kernel = "rectangular")
# ------------------------------------------------
# Ex 3
prob_eight <- model$prob[, 9][y_hat_t == 8]
ordering <- prob_eight[order(prob_eight)]
hard <- ordering[1:3]
easy <- ordering[(length(ordering)-1):length(ordering)]

hard_index <- which(prob_eight %in% hard)
easy_index <- which(prob_eight %in% easy)[1:2]


viz_number <- function(index, title, df = train, which_num = 8) {
  df <- train[y_hat_t == which_num, ]
  vec <- df[index, 1:64]
  viz_matrix <- matrix(0, 8, 8)
  for (i in 1:8) {
    viz_matrix[i, ] <- vec[(8*(i-1) + 1):(8*i)] %>% as.matrix()
  }

  M_long <- melt(viz_matrix)

  ggplot(M_long, aes(x = Var2, y = Var1, fill = value)) +
    geom_tile(color = "black") +
    geom_text(aes(label = round(value, 2)), color = "white", size = 5) +
    scale_fill_gradient(low = "white", high = "black", guide = "none") +
    theme_minimal(base_size = 25) +
    scale_y_discrete(limits = rev(levels(M_long$Var1))) +
    labs(x = "", y = "", title = title) +
    theme(
      axis.text.x = element_blank(),
      axis.text.y = element_blank(),
      plot.title  = element_text(hjust = 0.5, face = "bold"),
      panel.grid  = element_blank()
    )


}

ggarrange(
  viz_number(easy_index[1], "Easy 8s"),
  viz_number(easy_index[2], "")
)
```

```r
ggarrange(
  viz_number(hard_index[1], ""),
  viz_number(hard_index[2], "Hard 8s"),
  viz_number(hard_index[3], ""),
  ncol = 3
)


train_error <- c()
validation_error <- c()

for (k in 1:5) {
  if (k%%5 == 0) {
    #print(k)
  }
  model <- kknn(y ~ ., train = train, test = valid, k = k, kernel = "rectangular")
  y_hat_v <- model$fitted.values

  model <- kknn(y ~ ., train = train, test = train, k = k, kernel = "rectangular")
  y_hat_t <- model$fitted.values

  train_error <- c(train_error, 1 - (train$y == y_hat_t) %>% mean())
  validation_error <- c(validation_error, 1 - (valid$y == y_hat_v) %>% mean())

}

plot_df <- data.frame(Validation = validation_error, Train = train_error,
                      k = 1:length(train_error))
ggplot(plot_df) + aes(x = k) + geom_line(aes(y = Validation, color = "Validation")) +
  geom_line(aes(y = Train, color = "Train")) + theme_bw(base_size = 25) +
  scale_color_manual(values = c("Validation" = "red", "Train" = "black")) +
  geom_vline(xintercept = which.min(validation_error), linetype = "dashed", color = "grey", linewidth =
  labs(y = "Missclassification error", color = "Dataset") + theme(
    axis.text.x = element_text(hjust = 1, face = "bold"),
    axis.text.y = element_text(face = "bold"),
    axis.title.y = element_text(size = 25),
    plot.title = element_text(hjust = 0.5, face = "bold"),
    panel.grid = element_blank(),
  ) + annotate("text", x = 17, y = 0.01,
               label = paste0("Optimal k = ", which.min(validation_error)), size = 5)


model <- kknn(y ~ ., train = train, test = test,
              k = which.min(validation_error), kernel = "rectangular")
y_hat_test <- model$fitted.values


# Shows prescision and recall, only used for interest, not in the report:
missclass_table <- function(y, y_hat) {

  cm <- table(Number = y, Predict = y_hat)
  FN <- sapply(1:nrow(cm), function(x) (sum(cm[x, ]) - cm[x, x]))
  TP <- diag(cm)
```

```r
  FP <- sapply(1:ncol(cm), function(x) (sum(cm[, x]) - cm[x, x]))

  # when predicting class X how many times was right?
  Prescision <- TP / (TP + FP)

  # when the class was X how often was X predicted?
  Recall <- TP / (TP + FN)

  miss_table <- rbind(Prescision, Recall) %>% round(2)

  print("Prescision: when predicting class X how many times was right?")
  print("Recall: when the class was X how often was X predicted?")
  return(miss_table)

}

#missclass_table(test$y, y_hat_test)
#conf_table(test$y, y_hat_test)



cross_error <- c()
for (k in 1:5) {
  if (k%%5 == 0) {
    #print(k)
  }
  model <- kknn(y ~ ., train = train, test = valid, k = k, kernel = "rectangular")
  y_hat_ <- model$fitted.values


  # x + 1 as the indexing starts at 1
  loss_per_class <- sapply(0:9, function(x) {
    neg_loss <- (model$prob[valid$y == x, x + 1]  + 10e-15) %>%
      log() %>% sum()
    return(-neg_loss)
  }
  )


  cross_error <- c(cross_error, sum(loss_per_class))

}




plot_df <- data.frame(Validation = cross_error,
                      k = 1:length(cross_error))
ggplot(plot_df) + aes(x = k) + geom_line(aes(y = Validation), color = "black") +
  theme_bw(base_size = 25) +
  geom_vline(xintercept = which.min(cross_error), linetype = "dashed", color = "grey", linewidth = 1.5)
  labs(y = "Cross-entropy loss") + theme(
    axis.text.x = element_text(hjust = 1, face = "bold"),
```

```
    axis.text.y = element_text(face = "bold"),
    axis.title.y = element_text(size = 25),
    plot.title = element_text(hjust = 0.5, face = "bold"),
    panel.grid = element_blank(),
  ) + annotate("text", x = 17, y = 1000,
               label = paste0("Optimal k = ", which.min(cross_error)), size = 5)



#######################
#
# TASK 2 CODE
#
#######################
# Set all variables to null to avoid possible overlap between unrelated parts of the lab
rm(list = ls())

Loglikelihood <- function(theta, sigma, X, y) {

  stopifnot(length(theta) == ncol(X) + 1)
  stopifnot(is.numeric(sigma) && sigma > 0)

  lm_model <- function(x, theta) {
    x <- as.matrix(x)
    X <- cbind(1, x)
    mu <- X %*% theta
    return(as.numeric(mu))
  }

  # Calculate fitted mean
  mu <- lm_model(X, theta)

  # Compute log-likelihood for N(y | mu, sigma^2) for each row
  # logL = -1/2 * n * log(2*pi) - n * log(sigma) - 1/(2*sigma^2) * sum((y - mu)^2)
  n <- length(y)
  residuals <- y - mu
  logL <- -0.5 * n * log(2 * pi) - n * log(sigma) - (0.5 / sigma^2) * sum(residuals^2)

  return(logL)
}


Ridge <- function(theta, sigma, lambda, X, y) {

  stopifnot(length(theta) == ncol(X) + 1)
  stopifnot(is.numeric(sigma) && sigma > 0)
  stopifnot(is.numeric(lambda) && lambda >= 0)

  logL <- Loglikelihood(theta, sigma, X, y)

  ridge_penalty <- lambda * sum(theta^2)
```

```r
  neg_loglik_with_penalty <- -logL + ridge_penalty
  return(neg_loglik_with_penalty)
}

RidgeOpt <- function(theta, sigma, lambda, X, y) {
  optim(par=theta, fn=Ridge, gr=NULL, method = "BFGS", sigma=sigma, lambda=lambda, X=X, y=y)
}

DF <- function(lambda, X) {
  # Calculate the singular values of the training data matrix X.
  d <- svd(X)$d

  # Calculate the degrees of freedom using the formula:
  # sum(d^2 / (d^2 + lambda))
  # This formula is a more computationally stable way to
  # find the trace of the hat matrix.
  # See https://stats.stackexchange.com/questions/220243/the-proof-of-shrinking-coefficients-using-ridg
  # for details
  df <- sum(d^2 / (d^2 + lambda))

  # Return the calculated degrees of freedom.
  return(df)
}
# Read and clean data
#setwd("C:/Users/arone/Desktop/MasterML")
data <- read.csv("data/parkinsons.csv", header = TRUE)
data <- subset(data, select = -c(subject., age, sex, test_time, total_UPDRS))

# Scale
# Exclude non-feature columns from scaling
exclude_cols <- c("motor_UPDRS")
feature_cols <- setdiff(names(data), exclude_cols)
data[feature_cols] <- scale(data[feature_cols])
feature_means <- colMeans(data[feature_cols])
feature_sds <- apply(data[feature_cols], 2, sd)

#print("Scaled data sample")
#print(head(data))
#print("Scaled data means")
#print(feature_means)
#print("Scaled data SDs")
#print(feature_sds)
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=data[id,]
test=data[-id,]
print(paste("Train data size: ", dim(train)[1]))
print(paste("Test data size: ", dim(test)[1]))
fit <- lm(motor_UPDRS ~ ., data = train)
#print(summary(fit))
print(paste("Train MSE: ", mean(fit$residuals^2)))
test_predictions <- predict(fit, newdata = test)
```

```r
test_mse <- mean((test$motor_UPDRS - test_predictions)^2)
print(paste("Test MSE: ", test_mse))

# Checking significance of each predictor
summary_fit <- summary(fit)
coefficients <- summary_fit$coefficients[-1,]
# Columns: Estimate, Std. Error, t value, Pr(>|t|)
# Variables with small p-values (below threshold, e.g. 0.05)
# contribute significantly and are likely good predictors
significant_vars <- coefficients[coefficients[,4] < 0.05, ]
print("Features contributing significantly to the model: ")
print(rownames(significant_vars))
sigma <- summary(fit)$sigma
theta <- coef(fit)

# y <- data$motor_UPDRS
# X <- as.matrix(subset(data, select = -c(motor_UPDRS)))
# result <- Loglikelihood(theta, sigma, X, y)
# print(paste("Loglikelihood result: ", result))
# lambda <- 0.1                    # example ridge penalty
# ridge_result <- Ridge(theta, sigma, lambda, X, y)
# print(paste("Ridge result: ", ridge_result))

lambdas <- c(0, 0.01, 0.1, 1, 10, 100, 1000)
train_X <- as.matrix(subset(train, select = -motor_UPDRS))
train_y <- train$motor_UPDRS
results <- data.frame(
  lambda = numeric(),
  train_MSE = numeric(),
  test_MSE = numeric(),
  DF = numeric()
)

for (lambda in lambdas) {
  opt <- RidgeOpt(theta = theta, sigma = sigma, lambda = lambda, X = train_X, y = train_y)
  theta_hat <- opt$par

  train_X_pred <- cbind(1, train_X)
  test_X <- as.matrix(subset(test, select = -motor_UPDRS))
  test_X_pred <- cbind(1, test_X)

  train_preds <- as.numeric(train_X_pred %*% theta_hat)
  test_preds <- as.numeric(test_X_pred %*% theta_hat)

  train_MSE <- mean((train_y - train_preds)^2)
  test_MSE <- mean((test$motor_UPDRS - test_preds)^2)

  df_val <- DF(lambda, train_X)

  results <- rbind(results, data.frame(lambda = lambda, train_MSE = train_MSE, test_MSE = test_MSE, DF =
}

table_caption <- "Ridge Regression Results (Train/Test MSE and Degrees of Freedom)"
```

```r
knitr::kable(
  results,
  digits = 3,
  format.args = list(scientific = FALSE),
  col.names = c("Lambda", "Train MSE", "Test MSE", "Degrees of Freedom"),
  caption = table_caption,
  align = c("c", "r", "r", "r")
)
########################
#
# TASK 3 CODE
#
########################
# Set all variables to null to avoid possible overlap between unrelated parts of the lab
rm(list = ls())
library(ggplot2)
df <- as.data.frame(read.csv("data/pima-indians-diabetes.csv", header = FALSE))

colnames(df) <- c("pregnancies", "glucose", "blood_pressure", "skin_thickness", "insulin", "BMI", "diabe

ggplot(df, aes(x = age, y = glucose, color = factor(diabetes))) +
  geom_point() +
  labs(title = "Plasma glucose Concentration vs age",
       x = "Age",
       y = "Plasma glucose Concentration",
       color = "diabetes") +
  theme_minimal()

model <- glm(diabetes ~ glucose + age, data = df, family = binomial)

# Show summary
#summary(model)

# Predicted probabilities
probs <- predict(model, type = "response")

# Predicted classes using threshold r = 0.5
pred_class <- ifelse(probs > 0.5, 1, 0)

# Compute misclassification error
misclassification_func <- function(predicted, actual) {
  n <- length(actual)
  confusion <- table(predicted, actual)
  accuracy <- sum(diag(confusion)) / n
  misclassification <- 1 - accuracy
  return(misclassification)
}


error_rate <- misclassification_func(pred_class, df$diabetes)
cat("Misclassification rate (r = 0.5):", round(error_rate, 3), "\n")

coeff <- coef(model)
```

```r
slope <- -coeff["age"] / coeff["glucose"]
intercept <- -coeff["(Intercept)"] / coeff["glucose"]
cat("Equation of the decision boundary: glucose = ", slope, " * age + ", intercept, "\n")

# Plot with boundary line
ggplot(df, aes(x = age, y = glucose, color = factor(pred_class))) +
  geom_point() +
  geom_abline(intercept = intercept, slope = slope,  color = "darkgreen", linetype = "solid", linewidth
  labs(title = "Decision Boundary (r = 0.5)",
       x = "Age",
       y = "Plasma glucose Concentration",
       color = "Predicted diabetes") +
  theme_minimal()

r_values <- c(0.2, 0.8)

for (r in r_values) {
  pred_class_r <- ifelse(probs > r, 1, 0)

  # Misclassification rate
  err <- misclassification_func(pred_class_r, df$diabetes)
  cat("Misclassification rate (r =", r, "):", round(err, 3), "\n")

  # Plot
  ggplot(df, aes(x = age, y = glucose, color = factor(pred_class_r))) +
    geom_point() +
    labs(title = paste("Decision Regions for threshold r =", r),
         x = "Age",
         y = "Plasma glucose concentration",
         color = "Predicted class") +
    theme_minimal() ->
    p

  print(p)
}

# Create basis expansion features
df$z1 <- df$glucose^4
df$z2 <- df$glucose^3 * df$age
df$z3 <- df$glucose^2 * df$age^2
df$z4 <- df$glucose * df$age^3
df$z5 <- df$age^4

# Fit expanded model
model_expanded <- glm(diabetes ~ glucose + age + z1 + z2 + z3 + z4 + z5,
                      data = df, family = binomial)

#summary(model_expanded)

# Predicted probabilities
probs_expanded <- predict(model_expanded, type = "response")
pred_expanded <- ifelse(probs_expanded > 0.5, 1, 0)
```

```r
# Misclassification rate
error_rate_expanded <- misclassification_func(pred_expanded, df$diabetes)
cat("Misclassification rate (expanded model, r = 0.5):",
    round(error_rate_expanded,3), "\n")

# Plot
ggplot(df, aes(y = glucose, x = age )) +
  geom_point(aes(color = factor(pred_expanded))) +
  labs(
    x = "Age",
    y = "Plasma glucose concentration",
    color = "Diabetes"
  )
```