732A99/732A68/TDDE01 Machine Learning Computer Lab 1

Erik Lundqvist (erilu777)
Axel Strid (axest556)
Oliver Solvang Stoltz (oliso325)

December 13, 2024

Statement of Contribution

Erik Lundqvist, Axel Strid and Oliver Solvang Stoltz started working on the begining of task 1.1 and 1.2, Oliver Solvang Stoltz then completed assignment 1 and wrote the report about assignment 1.

Erik Lundqvist coded and wrote the part of the report for assignment 2.

Axel Strid coded and wrote the report regarding Assignment 3.

Oliver Solvang Stoltz did theory part for assignment 4 and wrote the report for this assignment .

1 Assignment 1: Handwritten Digit Recognition with K-nearest Neighbors

This assignment was about classifying handwritten numbers, 0-9 using the k-nearest neighbors method (KKNN). The dataset contained handwritten numbers from 43 different people and a validation number of the actual number.

1.1 Data Preparation and Initial Model

The data was imported and divided into train-, validation- and test by 50%, 25%, 25% respectively. The seed 12345 was used to ensure reproducibility in the random splitting process when splitting the data. After the data was split, the KKNN model was now fitted with a K value = 30 and a rectangular kernel. The model was fitted twice: first using the training data for both training and testing and second, using the training data for training and test data for testing. The resulting confusion matrices and misclassification errors were compared to analyze the differences.

1.2 Classification Results

The result of the confusion matrix for the training data can be seen in Table: 1 and the confusion matrix for the test data can be seen in Table: 2

Table 1: Confusion Matrix for Training Data

Truth \ Predicted	**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 Test Data

Truth \ Predicted	**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

The misclassification error for the training data was calculated to be $0.04500262 \approx 4.50\%$, and the misclassification error for the test data was calculated to be $0.05329154 \approx 5.33\%$. This implies that the overall accuracy of the model is about 95%. This level of accuracy might be acceptable for use cases where some error is tolerable, such as scanning documents, where occasional misclassification may not significantly impact the outcome. However, if the model is used for more critical tasks, such as grading exams, a higher accuracy would likely be required to minimize errors and ensure fairness. The model sometimes misclassifies 4 as 7, 5 as 9, and 8 as 1, this is likely since these numbers have some things in common. However, a high majority is still accurate which makes us believe that this model is accurate.

1.3 Analysis of Digit ”8” Classification

The next step was to identify two cases that the model found easiest to classify as eight and three cases it struggled the most to classify as eight. After arranging the probabilities for the prediction of the number eight, heatmaps were created to visualize different cases. The easiest cases to classify as an eight were at indices 129 and 195, with index 129 being the easiest to classify. For the hardest cases to classify as an eight, the indices were 520, 431, and 1294, with index 520 being the hardest to classify. In figure: 1, 2, 3, 4 and 5 the different heat maps can be seen.

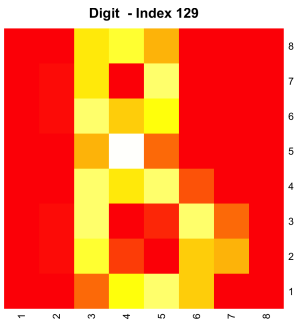

Figure 1: The easiest case for the model to predict as an eight.

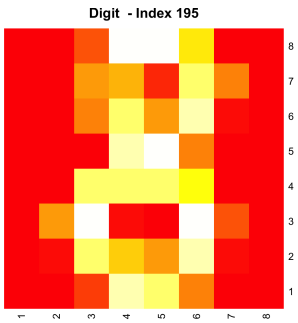

Figure 2: The second easiest case for the model to predict as an eight.

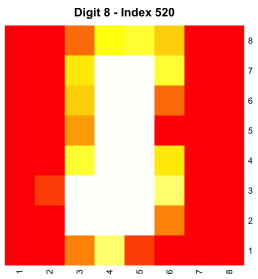

Figure 3: The hardest case for the model to predict as an eight.

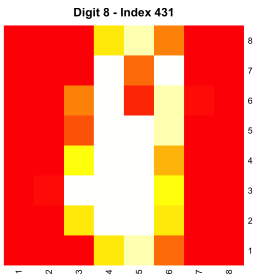

Figure 4: The second hardest case for the model to predict as an eight.

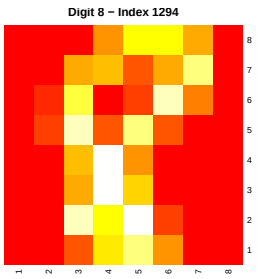

Figure 5: The third hardest case for the model to predict as an eight.

Visually, the easy cases can be identified as eights. The hard cases are indeed challenging to predict, and these numbers could be 8, 9, or even 1.

1.4 Different K-values and their effect on misclassification error

The next step was to calculate the misclassification error for different K-values for the training data and the validation data to see what value corresponded with the lowest misclassification error. This resulted in a graph with training error, validation error and an optimal k-value, this graph can be found in figure: 6.

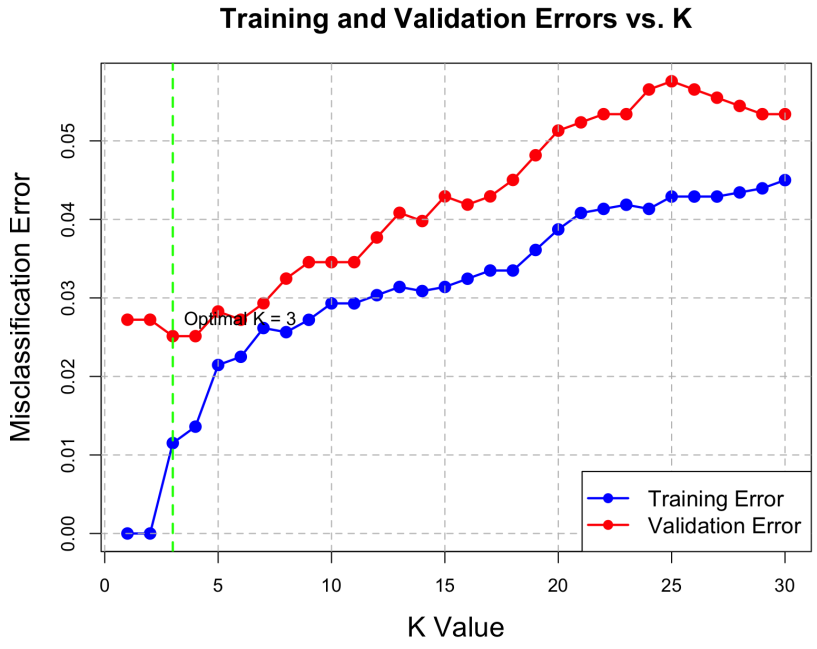

Figure 6: An graph for different K values and their corresponding misclassification error, for training- and validation data.

In Figure 6, we can see that the optimal K value is 3, with a misclassification error of approximately 2.5%, as the validation error is the lowest. With a low K-value, the model becomes more complex and tries to fit every inconsistently, which will lead to overfitting. As K increases, the model becomes less complex because it generalizes more, leading to

underfitting. With a K-value $= 3$ the test error was calculated to $0.03239289 \approx 3.24\%$. The test error shows that the model would be useful on unseen data with a high accuracy.

1.5 Different K-values and their effect on Cross-entropy error

The final task was to use cross-entropy error instead of misclassification error and then see which K-value gave the optimal solution. In figure 7 the result of this is shown.

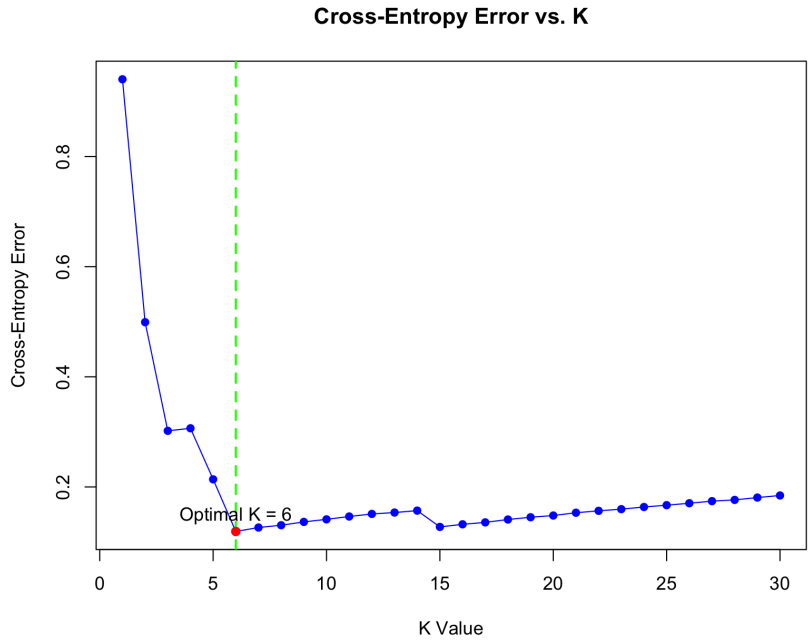

Figure 7: An graph for different K values and their corresponding cross-entropy error, for validation data.

We can see that the optimal solution was k=6 with a cross entropy error to be approximately 0.07. If you assume that the response has a multinational distribution, cross-entropy error could be more useful than misclassification error. This is because cross-entropy error takes into account the prediction probability, not just the predicted class like the misclassification error does, this makes it better to calibrate the prediction and penalize wrong predictions [1, pp. 109–119].

2 Assignment 2: Linear Regression and Ridge Regression

2.1 Introduction

This assignment focuses on analyzing Parkinson's disease symptom data using linear regression and ridge regression techniques. The main objective is to predict the motor UPDRS score (a measure of Parkinson's disease symptom severity) using various biomedical voice measurements. The dataset contains voice recordings from 42 people with early-stage Parkinson's disease, collected during a six-month trial of a telemonitoring device. The given data set consists of 5875 observations and 22 variables.

The assignment consists of four main tasks:

1. Data preparation and scaling
2. Initial linear regression analysis
3. Implementation of ridge regression functions
4. Analysis of ridge regression with different penalty parameters

2.2 Data Preparation

The data was split into 60% training data and 40% testing data. Then, the training data was scaled using caret (z-score standardization) so that every column has the same mean (=0) and standard deviation (=1) and using the same parameters, the test data was also scaled.

2.3 Linear Regression

A linear regression model was fitted using the scaled training data. Table 3 shows the coefficient estimates and their statistical significance for all predictor variables.

Table 3: Summary of linear regression coefficients and their statistical significance, ordered by absolute t-value

Variable	Estimate	Std. Error	t value	p-value
DFA	-0.280318	0.020136	-13.921	$< 2\times10^{-16}$ ***
PPE	0.226467	0.032881	6.887	6.70×10^{-12} ***
HNR	-0.238543	0.036395	-6.554	6.41×10^{-11} ***
Shimmer.APQ11	0.305546	0.061236	4.990	6.34×10^{-7} ***
Jitter.Abs.	-0.169609	0.040805	-4.157	3.31×10^{-5} ***
NHR	-0.185387	0.045567	-4.068	4.84×10^{-5} ***
Shimmer.APQ5	-0.387507	0.113789	-3.405	6.68×10^{-4} ***
Shimmer	0.592436	0.205981	2.876	4.05×10^{-3} **
Jitter...	0.186931	0.149561	1.250	2.11×10^{-1}
Shimmer.dB.	-0.172655	0.139316	-1.239	2.15×10^{-1}
Jitter.PPQ5	-0.074568	0.087766	-0.850	3.96×10^{-1}
Shimmer.DDA	-32.387241	77.158814	-0.420	6.75×10^{-1}
Shimmer.APQ3	32.070932	77.159242	0.416	6.78×10^{-1}
Jitter.DDP	5.249558	18.837525	0.279	7.81×10^{-1}
Jitter.RAP	-5.269544	18.834160	-0.280	7.80×10^{-1}
RPDE	0.004068	0.022664	0.179	8.58×10^{-1}
Significance codes: *** p < 0.001, ** p < 0.01, * p < 0.05				
$R^2 = 0.1212$, Adjusted $R^2 = 0.1172$, F-statistic = 30.25 on 16 and 3509 DF				
Residual standard error: 0.9394 on 3509 degrees of freedom				

The linear regression model shows an adjusted R-squared of 0.1172, indicating that about 12% of the variance in motor_UPDRS is explained by the voice characteristics. The most significant predictors are DFA (t = -13.921), PPE (t = 6.887), and HNR (t = -6.554).

MSE values:

- Training data: 0.8785
- Testing data: 0.9354

2.4 Ridge Regression Implementation

To perform ridge regression analysis, four core functions that work together to find optimal parameters while incorporating a ridge penalty were implemented. Each function serves a specific purpose in the implementation:

2.4.1 Log-likelihood Function

A function to compute the log-likelihood for normally distributed data was implemented:

- **Input**: Parameter vector θ (value of the coefficients for how much each feature contributes to the target variable), dispersion σ, and training data
- **Implementation**:
  - Constructs design matrix X with removed target value column
  - Makes predictions using $X\theta$

– Computes log-likelihood using the normal distribution formula:

$$-\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{\sum(y - X\theta)^2}{2\sigma^2}$$

- **Purpose**: Measures how well our parameters explain the observed data, want to maximize this

2.4.2 Ridge Function

Built upon the log-likelihood function to create the ridge regression, it adds a ridge penalty to parameters:

- **Input**: Parameters θ, σ (the same θ and σ are used as input to the log-likelihood function), penalty parameter λ (bigger lambda -¿ coefficients will be smaller, and training data
- **Implementation**:
  - Computes negative log-likelihood
  - Adds ridge penalty $\lambda \sum \theta_{(i)}^2$
- **Purpose**: Creates the complete objective function to be minimized

2.4.3 Ridge Optimization Function

This function finds the optimal parameters for the ridge regression model:

- **Input**: Penalty parameter λ and training data
- **Implementation**:
  - Creates a helper function that uses the ridge function (which already combines log-likelihood and ridge penalty)
  - Uses R's optimization function `optim()` with BFGS method
  - Starts with initial values (zeros for coefficients, 1 for σ)
  - Iteratively finds parameters that minimize the ridge function value
- **Purpose**: Automatically finds the best parameter values (θ and σ) that:
  - Minimize the ridge function value for a given λ
  - Return optimal coefficients and sigma for the model
- **Output**: Returns the optimal values for all parameters $(\theta_0, \theta_1, ..., \theta_p, \sigma)$

2.4.4 Degrees of Freedom Function

This function calculates how flexible the ridge regression model is, tells us "how many effective parameters" we're using with the current values of λ:

- **Input**: Penalty parameter λ and training data
- **Implementation**:
  - Uses the formula $trace(X(X'X + \lambda I)^{-1}X')$ to calculate effective degrees of freedom
  - Larger λ leads to fewer effective degrees of freedom
  - Represents how much freedom the model has to fit the data
- **Purpose**: Measures how complex or flexible our model is after ridge penalty is applied

These functions work together in sequence to perform ridge regression:

1. Log-likelihood measures fit of parameters
2. Ridge function adds penalty for complexity
3. Optimization finds best parameters
4. Degrees of freedom quantifies model complexity

2.5 Ridge Regression Analysis

To evaluate the effect of different ridge penalties, we tested the model with three different values of λ (1, 100, and 1000) and compared them with the original linear regression model ($\lambda = 0$). For each model, we calculated:

- Training and test Mean Squared Error (MSE) for both the training and test data
- Effective degrees of freedom to measure model complexity

Table 4 shows the comparison of MSE values and degrees of freedom for different λ values:

Table 4: Comparison of model performance with different ridge penalties (λ)

λ	Training MSE	Test MSE	Degrees of Freedom
0 (Initial Linear Regression)	0.8785	0.9354	16.0000
1	0.8786	0.9350	13.8607
100	0.8844	0.9323	9.9249
1000	0.9211	0.9539	5.6439

Among the tested values, λ=100 appears most appropriate as it achieves the lowest test MSE (0.9323). As λ increases, we can see a trade-off: the degrees of freedom decrease significantly (from 16 to 5.6), indicating reduced model complexity, while the training MSE gradually increases. The results suggest that moderate regularization (λ=100) provides slightly better generalization performance compared to both the unregularized model

(λ=0) and strong regularization (λ=1000). The test MSE improves slightly from 0.9354 (λ=0) to 0.9323 (λ=100), while training MSE gets slightly worse (0.8785 to 0.8844). This suggests there was a small amount of overfitting that regularization helped with, but not much.

3 Assignment 3: Logistic Regression and Basis Function Expansion

3.1 Initial Data Visualization

The first task of assignment 3 was to create a scatterplot showing Plasma glucose concentration versus Age, with points colored based on Diabetes status. Based on the figure, diabetes is not easy to classify using a standard logistic regression model that uses these two variables because the data shows significant overlap between the classes.

3.2 Basic Logistic Regression

The code trains a logistic regression model using Plasma Glucose Concentration and Age as features to predict Diabetes. It calculates predicted probabilities, and then classifies observations using a threshold of 0.5. It also extracts the probabilistic equation of the estimated model, indicating how the target depends on the features and the estimated model parameters probabilistically. The training misclassification error has also been calculated. Finally, it visualizes the predicted classifications on a scatterplot, similar as the one in assignment 3.1, but now showing the predicted classifications of diabetes.

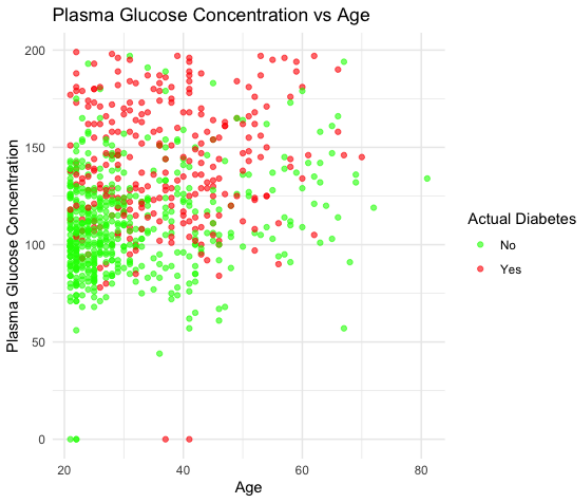

Figure 8: Scatterplot with points colored based on actual Diabetes status.

Figure 9: Scatterplot of trained logistic regression model with points based on predicted Diabetes status.

Logistic Regression - Probabilistic Equation:

$$\text{logit}(P(y = 1)) = -5.8979 + 0.0356 \cdot \text{Glucose} + 0.0245 \cdot \text{Age}$$

Training Misclassification Error: 26.60%

The model indicates that higher glucose and age increase the probability of diabetes, but the 26.60% misclassification error suggests moderate accuracy, implying room for improvement in feature selection or model complexity.

3.3 Decision Boundary Analysis

The decision boundary between the two classes has been calculated for the model used in assignment 3.2. This line has been implemented in the scatterplot.

Decision Boundary Equation: The decision boundary is determined where the probability of $y = 1$ equals 0.5, which corresponds to the logit function being equal to 0:

$$\text{logit}(P(y = 1)) = w_0 + w_1 \cdot \text{Glucose} + w_2 \cdot \text{Age} = 0$$

Rearranging for Age:

$$\text{Age} = -\frac{w_0}{w_2} - \frac{w_1}{w_2} \cdot \text{Glucose}$$

Substituting the coefficients from the model ($w_0 = -5.8979$, $w_1 = 0.0356$, $w_2 = 0.0245$):

$$\text{Age} = -\frac{-5.8979}{0.0245} - \frac{0.0356}{0.0245} \cdot \text{Glucose}$$

Simplifying:

$$\text{Age} = 240.73 - 1.453 \cdot \text{Glucose}$$

Thus, the **decision boundary equation is:**

$$\text{Age} = 240.73 - 1.453 \cdot \text{Glucose}$$

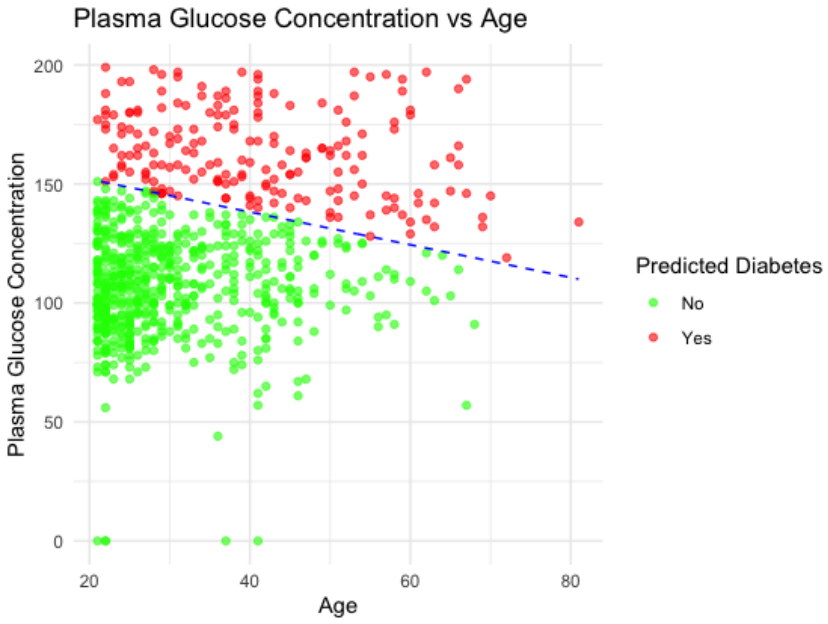

Figure 10: Scatterplot from assignment 3.2 with decision boundary line

The decision boundary in the scatterplot generally separates the two classes (diabetes vs. no diabetes) based on glucose and age. It captures the data distribution reasonably well, with most of the red points (diabetes) above the line and green points (no diabetes) below. However, there is some overlap, particularly near the boundary, indicating some misclassified points

3.4 Threshold Analysis

The code script adjusts the decision threshold (r) for classifying diabetes and generates scatterplots for $r = 0.2$ and $r = 0.8$.

When $r = 0.2$, most predictions are classified as diabetes (red) since any probability greater than 0.2 is classified as positive (=1). Conversely, when $r = 0.8$, most predictions are classified as no diabetes (green), as only probabilities above 0.8 are classified as positive. This demonstrates the sensitivity of predictions to the chosen threshold.

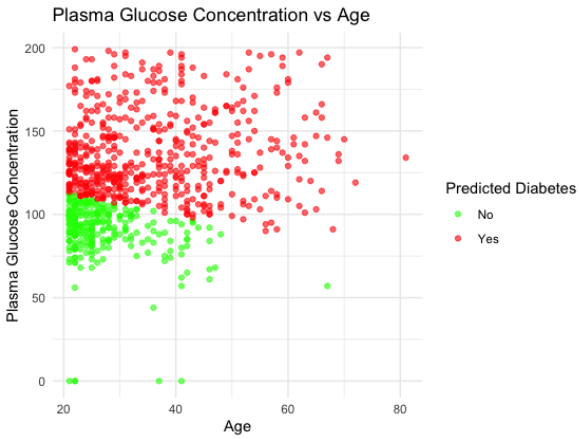

Figure 11: Scatterplot with $r = 0.2$: Majority classified as diabetes.

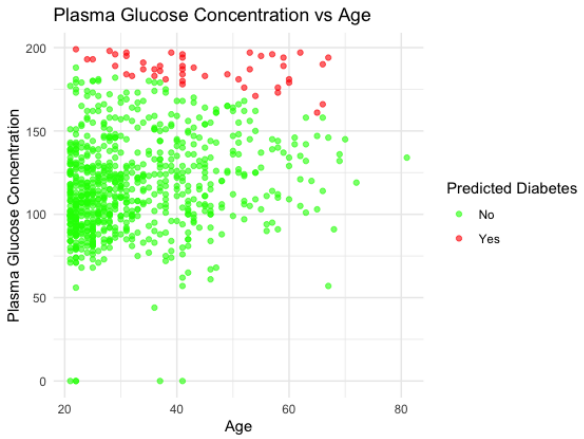

Figure 12: Scatterplot with $r = 0.8$: Majority classified as no diabetes.

3.5 Basis Function Expansion

The task involves applying a basis function expansion to the logistic regression model by creating new polynomial interaction features: $z_1 = x_1^4$, $z_2 = x_1^3 \cdot x_2$, $z_3 = x_1^2 \cdot x_2^2$, $z_4 = x_1 \cdot x_2^3$, and $z_5 = x_2^4$. Here, $x_1 =$ PlasmaGlucoseConcentration and $x_2 =$ Age. These features were added to the dataset and used to train an extended logistic regression model.

Training Misclassification Error (Expanded Model): 24.64%

The basis function expansion slightly reduces the training misclassification error from 26.60% in the previous model to 24.64% in the extended model. While the new model fits the data better and captures more complexity, it risks overfitting due to its complexity. This confirms that predicting diabetes based on Plasma Glucose Concentration and Age alone is challenging, and the initial model may not be the most reliable.

The decision boundary for the basis expansion model is determined by solving the equation:

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^4 + w_4 x_1^3 x_2 + w_5 x_1^2 x_2^2 + w_6 x_1 x_2^3 + w_7 x_2^4 = 0$$

Given the complexity of the equation, numerical methods are necessary to plot the decision boundary. The basis expansion trick has transformed the decision boundary from linear to a curve, reflecting the added polynomial interaction terms.

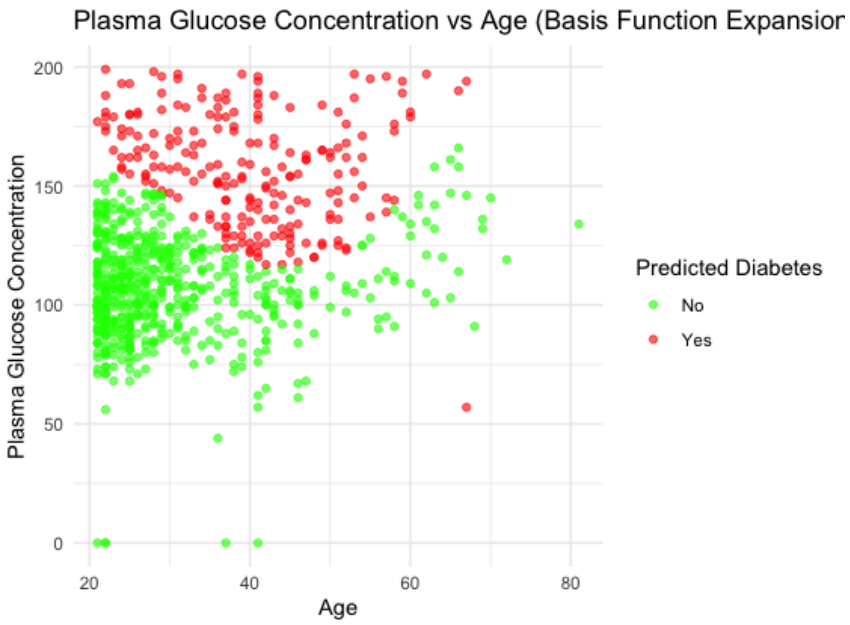

Figure 13: Scatterplot of predicted Diabetes status with Basis Expansion Model.

4 Assignment 4: Theory

In this assignment, three questions will be answered using the course book, *Machine Learning: A First Course for Engineers and Scientists* [1].

4.1 Probability Thresholds in Classification

Why can it be important to consider various probability thresholds in the classification problems, according to the book? In the book, it is stated that it could be a good idea to consider different probability thresholds since this could let the classifier achieve different things. The standard probability threshold is 0.5 which will make the misclassification error the smallest but real-world problems might not be the most important factor, by changing the threshold we can better predict asymmetric events. For example, in the medical field, it is more important to not miss a diagnosis of a sick person rather than false classifying a healthy person with a disease [1, pp. 49–50].

4.2 Target Variable Collection Methods

What ways of collecting correct values of the target variable for the supervised learning problems are mentioned in the book? In the book two different methods to get the target variable are mentioned: Joint recording and manually labeling the output. Joint recording is when one input gives an output that is easier to obtain than manually labeling the output. To manually label the output a branch expert needs to examine every input and then classify the most likely output [1, pp. 13–14].

4.3 Linear Regression Cost Function Matrix Form

How can one express the cost function of the linear regression in the matrix form, according to the book? In the book [1, pp. 41], one can find the formula 3.11 Which expresses the

cost function in linear regression in matrix form. In 3.11 we have

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}(x_i;\boldsymbol{\theta}) - y_i)^2 = \frac{1}{n}\|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 = \frac{1}{n}\|\boldsymbol{\epsilon}\|_2^2, \tag{3.11}$$

where $\|\cdot\|_2$ denotes the usual Euclidean vector norm and $\|\cdot\|_2^2$ its square. Note that this is directly taken from the book.

References

[1] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning: A First Course for Engineers and Scientists*. Cambridge University Press, Jul. 2022. Pre-publication version. Accessed online 24 Nov 2024.

A Code

```r
#### INSTALL NECESSARY PACKAGES ####
install.packages("kknn")
library(kknn)

#### DIVIDE THE DATA ####

# Load the data into a variable
data <- read.csv("optdigits.csv", header=F)

# Get the number of rows in the dataset
n <- dim(data)[1]

# Set a random seed for reproducibility
set.seed(12345)

# Partition 50% of the data for the training set
id <- sample(1:n, floor(n * 0.5))
train <- data[id, ]

# Partition 25% of the data for the validation set
id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n * 0.25))
valid <- data[id2, ]

# Use the rest for the test set
id3 <- setdiff(id1, id2)
test <- data[id3, ]


#### FIT A K-NEAREST NEIGHBOR MODEL TO TRAIN DATA ####

# Fitting = Learning = Training...?

# Fit the model on training data and test on training data
model_train <- kknn(as.factor(V65) ~ ., train = train, test =
    ↪ train, k = 30, kernel = "rectangular")
train_pred <- fitted(model_train)
confusion_matrix_train <- table(Truth = train$V65, Predicted =
    ↪ train_pred)
confusion_matrix_train

# Define the misclassification error function
missclass <- function(truth, predicted) {
  n <- length(truth)
  return(1 - sum(diag(table(truth, predicted))) / n)
}

# Misclassification errors for the training data
```

```r
train_misclass_error <- missclass(train$V65, train_pred)
print(train_misclass_error) #0.04500262

# Fit the model on training data and test on test data
model_test <- kknn(as.factor(V65) ~ ., train = train, test = test
   ↪ , k = 30, kernel = "rectangular")
test_pred <- fitted(model_test)
confusion_matrix_test <- table(Truth = test$V65, Predicted = test
   ↪ _pred)
confusion_matrix_test

# Misclassification errors for the training data
test_misclass_error <- missclass(test$V65, test_pred)
print(test_misclass_error) #0.05329154

#### TASK 1.3 ####
  # Get probabilities of each class for each case in the training
     ↪  data
train_probs <- model_train$prob
  # Filter cases where the true label is "8"
eight_indices <- which(train$V65 == 8)

  # Get the probabilities for the correct class (8) for each case
eight_probs <- train_probs[eight_indices, "8"]

# Sort by probability: highest for easiest, lowest for hardest
sorted_indices <- order(eight_probs, decreasing = TRUE)
easiest_indices <- eight_indices[sorted_indices[1:2]]   # Two
   ↪ highest probabilities
hardest_indices <- eight_indices[sorted_indices[(length(sorted_
   ↪ indices)-2):length(sorted_indices)]]  # Three lowest
   ↪ probabilities

# Function to visualize an 8x8 matrix of a digit with index
visualize_digit <- function(data_row, label, index) {
  # Reshape to 8x8 matrix
  digit_matrix <- matrix(as.numeric(data_row), nrow = 8, ncol =
     ↪ 8, byrow = TRUE)
  # Plot heatmap
  heatmap(digit_matrix, Rowv = NA, Colv = NA, scale = "none", col
     ↪  = heat.colors(256),
          main = paste("Digit", label, "- Index", index))
}

# Visualize easiest cases for digit "8" with indices
for (i in easiest_indices) {
  visualize_digit(train[i, -ncol(train)], train$XV65[i], i)  #
     ↪ Include index
}

# Visualize hardest cases for digit "8" with indices
```

```r
for (i in hardest_indices) {
  visualize_digit(train[i, -ncol(train)], train$V65[i], i)  # Include index
}


#### TASK 1.4 ####

Kvalues <- 1:30
train_errors <- numeric(length(Kvalues)) #creates a empty set of vectors
valid_errors <- numeric(length(Kvalues)) #creates a empty set of vectors

for (k in Kvalues) {
  # Fit KNN model on training data and predict on training data
  model_train <- kknn(as.factor(V65) ~ ., train = train, test = train, k = k, kernel = "rectangular")
  train_pred <- fitted(model_train)
  train_errors[k] <- missclass(train$V65, train_pred)  # Store training error

  # Fit KNN model on training data and predict on validation data
  model_valid <- kknn(as.factor(V65) ~ ., train = train, test = valid, k = k, kernel = "rectangular")
  valid_pred <- fitted(model_valid)
  valid_errors[k] <- missclass(valid$V65, valid_pred)  # Store validation error
}

# Set up plot for training errors with improved readability
plot(Kvalues, train_errors, type = "o", col = "blue", pch = 16, lwd = 2, cex = 1.5,
     xlab = "K Value", ylab = "Misclassification Error",
     main = "Training and Validation Errors vs. K",
     ylim = range(c(train_errors, valid_errors)), cex.lab = 1.5, cex.main = 1.5)

# Add validation errors with a thicker line and larger points
lines(Kvalues, valid_errors, type = "o", col = "red", pch = 16, lwd = 2, cex = 1.5)

# Add grid lines for readability
grid(nx = NULL, ny = NULL, lty = 2, col = "gray")

# Add a legend with larger font size
legend("bottomright", legend = c("Training Error", "Validation Error"),
       col = c("blue", "red"), pch = 16, lty = 1, lwd = 2, cex = 1.2)

```

```r
# Add a vertical line at the optimal K value
optimal_k <- which.min(valid_errors)
abline(v = Kvalues[optimal_k], col = "green", lty = 2, lwd = 2)
text(Kvalues[optimal_k], valid_errors[optimal_k] + 0.002, labels = paste("Optimal K =", Kvalues[optimal_k]), col = "black", pos = 4)

#Estimate the test error for k=optimal_k
model_test <- kknn(as.factor(V65) ~ ., train = train, test = test , k = optimal_k, kernel = "rectangular")
test_pred <- fitted(model_test)
test_misclass_error <- missclass(test$V65, test_pred)
print(test_misclass_error) #0.03239289


#### TASK 1.5 ####

# Define the cross-entropy error function
CE <- function(truth, probability_matrix) {
  epsilon <- 1e-15  # Small constant to avoid log(0)
  cross_entropy <- 0

  # Loop over each sample
  for (i in 1:length(truth)) {
    # Get the true class for the i-th sample
    true_class <- as.integer(truth[i])+1

    # Compute cross-entropy for this sample
    cross_entropy <- cross_entropy - log(probability_matrix[i, true_class] + epsilon)
  }

  # Return the average cross-entropy error
  return(cross_entropy / length(truth))
}

# Define the range of K values
Kvalues <- 1:30
cross_entropy_errors <- numeric(length(Kvalues))# Create an empty vector for cross-entropy errors
epsilon <- 1e-15  # Small constant to avoid log(0)

# Loop over each K value
for (k in Kvalues) {
  # Fit KNN model on training data and predict on validation data
  model_valid <- kknn(as.factor(V65) ~ ., train = train, test = valid, k = k, kernel = "rectangular")

  # Extract predicted probabilities for each class
  probs <- model_valid$prob
  valid_labels <- valid$V65
```

```
174
175
176
177   # Calculate cross-entropy error for the validation set and
          ↪ store it
178   cross_entropy_errors[k] <- CE(valid_labels, probs)
179 }
180
181 # Plot Cross-Entropy Error vs. K
182 plot(Kvalues, cross_entropy_errors, type = "o", col = "blue", pch
       ↪  = 16,
183      xlab = "K Value", ylab = "Cross-Entropy Error", main = "
           ↪ Cross-Entropy Error vs. K")
184
185 # Identify and mark the optimal K (minimum cross-entropy)
186 optimal_k <- which.min(cross_entropy_errors)
187 abline(v = Kvalues[optimal_k], col = "green", lty = 2, lwd = 2)
188 points(Kvalues[optimal_k], cross_entropy_errors[optimal_k], col =
       ↪  "red", pch = 19)
189 text(Kvalues[optimal_k], cross_entropy_errors[optimal_k], labels
       ↪ = paste("Optimal K =", Kvalues[optimal_k]), pos = 3)
```

```
1  # ASSIGNMENT 2
2
3  # Package imports
4  library(caret)
5
6  ### TASK 1 ###
7
8  # read the data
9  data <- read.csv("parkinsons.csv")
10
11 voice_features <- c("Jitter...", "Jitter.Abs.", "Jitter.RAP", "
       ↪ Jitter.PPQ5", "Jitter.DDP",
12                    "Shimmer", "Shimmer.dB.", "Shimmer.APQ3", "
                          ↪ Shimmer.APQ5", "Shimmer.APQ11", "Shimmer
                          ↪ .DDA",
13                    "NHR", "HNR",
14                    "RPDE",
15                    "DFA",
16                    "PPE")
17
18 model_data <- data[, c(voice_features, "motor_UPDRS")]
19
20 # Get the number of rows in the data set
21 n <- dim(model_data)[1]
22
23 # set random seed for reproducibility
24 set.seed(12345)
25
26 # Partition 60% of data for the training set
```

```r
id <- sample(1:n, floor(n * 0.6))
train <- model_data[id, ]

# Partition 40% of the data for the testing set
id1 <- setdiff(1:n, id)
test <- model_data[id1, ]

# Scale training data with caret
scaler <- preProcess(train)  # Learns scaling from training data
trainScaled <- predict(scaler, train)  # Applies scaling to
    ↪ training data

# Scale test data using same parameters from training data
testScaled <- predict(scaler, test)

### TASK 2, Linear Regression Model ###

# Fit linear regression model using scaled training data
# motor_UPDRS is the target variable, use all other columns as
    ↪ predictors
# motor_UPDRS means we're trying to predict motor_UPDRS
fit <- lm(formula = motor_UPDRS ~ . - 1, data=trainScaled)  # -1
    ↪ removes intercept

# Get summary to see which variables are significant
summary(fit) # DFA has a t-value of -13.921, the predictor with
    ↪ the highest absolute t-value

# Calculate MSE for training data
pred_train <- predict(fit, trainScaled)  # get predictions for
    ↪ training data
mse_train <- mean((trainScaled$motor_UPDRS - pred_train)^2)  #
    ↪ calculate training MSE

# Calculate MSE for test data
pred_test <- predict(fit, testScaled)  # get predictions for test
    ↪  data
mse_test <- mean((testScaled$motor_UPDRS - pred_test)^2)  #
    ↪ calculate test MSE

# Print MSE results
print(paste("Training MSE:", round(mse_train, 4)))
print(paste("Test MSE:", round(mse_test, 4)))

### TASK 3 ###

# 3a #
loglikelihood <- function(theta, sigma, traindata){
  # Get X matrix (predictors) from training data
  X <- as.matrix(traindata[, -which(names(traindata) == "motor_
      ↪ UPDRS")])
```

```r

  # Get y (actual values) from training data
  y <- traindata$motor_UPDRS

  # Calculate predicted values (X * theta)
  y_pred <- X %*% theta

  # Calculate log-likelihood using normal distribution formula
  n <- length(y) # Number of observations
  loglik <- -n/2 * log(2*pi) - n/2 * log(sigma^2) - sum((y - y_
    ↪ pred)^2)/(2*sigma^2) # Minus for minimization

  return(loglik)
}

# 3b #
ridge <- function(theta, sigma, lambda, traindata){
  # Get negative log-likelihood (minus because we want to
    ↪ minimize)
  neg_loglik <- -loglikelihood(theta, sigma, traindata)

  # Calculate ridge penalty using: lambda * sum(theta^2)
  # We exclude first theta (intercept) from penalty
  ridge_penalty <- lambda * sum(theta^2)

  # Return total negative log-likelihood + ridge penalty
  return(neg_loglik + ridge_penalty)
}

# 3c #
# Function that takes lambda and finds optimal theta and sigma
ridgeOpt <- function(lambda, traindata) {
  # Number of parameters we need (number of columns - 1 for motor
    ↪ _UPDRS)
  n_params <- ncol(traindata) - 1

  # Function that optim will minimize
  # Takes parameters and returns ridge value
  ridge_value <- function(params) {
    theta <- params[1:n_params]  # first n_params values are
      ↪ theta
    sigma <- params[n_params + 1]  # last value is sigma
    return(ridge(theta, sigma, lambda, traindata))
}

  # Starting values for optim: zeros for theta, 1 for sigma
  start_values <- c(rep(0, n_params), 1)

  # Run optimization
  result <- optim(par = start_values,       # starting values
                  fn = ridge_value,           # function to
```

```
                    ↪ minimize
                  method = "BFGS")          # optimization method

  # Return optimized parameters
  return(result$par)  # returns optimal theta and sigma
}

# 3d #
DF <- function(lambda, traindata){
  # Remove motor_UPDRS
  X <- as.matrix(traindata[, -which(names(traindata) == "motor_
     ↪ UPDRS")])

  # Matrix Magic
  p <- ncol(X)
  XtX <- t(X) %*% X
  ridge_matrix <- XtX + lambda * diag(p)
  df <- sum(diag(X %*% solve(ridge_matrix) %*% t(X)))
  return(df)
}

### 4 ###

# ridgeOpt returns the optimal parameters using the ridge
   ↪ function

result1 <- ridgeOpt(lambda=1, trainScaled)        # lambda=1
result100 <- ridgeOpt(lambda=100, trainScaled)    # lambda=100
result1000 <- ridgeOpt(lambda=1000, trainScaled)  # lambda=1000

# Function to calculate MSE
calculate_mse <- function(theta, sigma, data) {
  # Make X matrix without intercept
  X <- as.matrix(data[, -which(names(data) == "motor_UPDRS")])
  pred <- X %*% theta # Calculate predictions
  mse <- mean((data$motor_UPDRS - pred)^2)
  return(mse)
}

#### Calculate MSE for each lambda ####

# Calculate how many predictors we have
n_params <- ncol(model_data) - 1  # -1 for motor_UPDRS

# Calculate MSE for each lambda
# Lambda = 1
mse_train1 <- calculate_mse(result1[1:n_params], result1[n_params
   ↪  + 1], trainScaled)
mse_test1 <- calculate_mse(result1[1:n_params], result1[n_params
   ↪ + 1], testScaled)

```

```r
# Lambda = 100
mse_train100 <- calculate_mse(result100[1:n_params], result100[n_params + 1], trainScaled)
mse_test100 <- calculate_mse(result100[1:n_params], result100[n_params + 1], testScaled)

# Lambda = 1000
mse_train1000 <- calculate_mse(result1000[1:n_params], result1000[n_params + 1], trainScaled)
mse_test1000 <- calculate_mse(result1000[1:n_params], result1000[n_params + 1], testScaled)

# Calculate degrees of freedom (DF) for each lambda
df1 <- DF(1, trainScaled)
df100 <- DF(100, trainScaled)
df1000 <- DF(1000, trainScaled)

# Print the MSE results and degrees of freedom
print(paste("Lambda = 0 (without ridge function):    Train MSE:", round(mse_train, 4),
            "Test MSE:", round(mse_test, 4),
            "DF:", n_params))  # DF is number of predictors (no intercept)
print(paste("Lambda = 1:    Train MSE:", round(mse_train1, 4),
            "Test MSE:", round(mse_test1, 4),
            "DF:", round(df1, 4)))
print(paste("Lambda = 100:  Train MSE:", round(mse_train100, 4),
            "Test MSE:", round(mse_test100, 4),
            "DF:", round(df100, 4)))
print(paste("Lambda = 1000: Train MSE:", round(mse_train1000, 4),
            "Test MSE:", round(mse_test1000, 4),
            "DF:", round(df1000, 4)))


# Higher lambda -> Higher MSE -> Predictions get worse with increasing lambda
# This suggests that original model wasn't overfitting, no need for regularization

# Data frame of coefficients for comparison
coef_comparison <- data.frame(
  Variable = colnames(trainScaled)[-which(names(trainScaled) == "motor_UPDRS")],
  Lambda0 = fit$coefficients,  # All coefficients (no intercept to exclude)
  Lambda1 = result1[1:n_params],
  Lambda100 = result100[1:n_params],
  Lambda1000 = result1000[1:n_params]
)

# Print to look at the coefficients
```

```r
print("Coefficient comparison:")
print(coef_comparison)



# ASSIGNMENT 3
# Logistic regression and basis function expansion

# Load necessary library
library(ggplot2)

# Load the data
data <- read.csv("pima-indians-diabetes.csv")

# Rename the columns
colnames(data) <- c("Pregnancies", "PlasmaGlucoseConcentration", "BloodPressure", "SkinfoldThickness",
                    "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Diabetes")


#### Assignment 1.1 ####
# Scatterplot of diabetes observations in a Glucose vs Age graph

# Create the scatterplot
ggplot(data, aes(x = Age, y = PlasmaGlucoseConcentration, color = factor(Diabetes))) + # Define x/y-axes
  geom_point(alpha = 0.6) +  # Add points with 40% transparency
  labs( # Labels for the plot
    title = "Plasma Glucose Concentration vs Age",
    x = "Age",
    y = "Plasma Glucose Concentration",
    color = "Actual Diabetes"
  ) +
  theme_minimal() +  # Clean theme
  scale_color_manual(values = c("green", "red"), labels = c("No", "Yes")) # Customize color/label for Diabetes


#### Assignment 1.2 ####
# Predictions of Diabetes based on a logistic regression model trained with Glucose & Age as features

# Train the logistic regression model
logistic_model <- glm(Diabetes ~ PlasmaGlucoseConcentration + Age , data = data, family = binomial)

# Make predictions for all observations!

# Predicted probabilities for target variable Diabetes
data$Predicted_Probabilities <- predict(logistic_model, type = "
```

```r
   ↪ response") # Values between 0 and 1
# Convert predicted probabilities into binary classifications
   ↪ based on threshold = 0.5
data$Predicted_Classifications <- ifelse(data$Predicted_
   ↪ Probabilities > 0.5, 1, 0) # Values either 0 or 1

# Probabilistic equation of the estimated model
coefficients <- coef(logistic_model) # Extract the coefficients
intercept <- coefficients[1]
coef_glucose <- coefficients[2]
coef_age <- coefficients[3]
cat("Logistic Regression - Probabilistic Equation:\n")
cat(sprintf("logit(P(y = 1)) = %.4f + %.4f * Glucose + %.4f * Age
   ↪ \n",
            intercept, coef_glucose, coef_age)) # Print the
               ↪ result

# Training misclassification error
error <- mean(data$Predicted_Classifications != data$Diabetes)
cat(sprintf("\nTraining Misclassification Error: %.2f%%\n", error
   ↪  * 100)) # Print the result

# Plot the predicted Diabetes values based on Glucos & Age
ggplot(data, aes(x = Age, y = PlasmaGlucoseConcentration, color =
   ↪  factor(Predicted_Classifications))) + # Define x/y-axes
  geom_point(alpha = 0.6) +  # Add points with 40% transparency
  labs( # Labels for the plot
    title = "Plasma Glucose Concentration vs Age",
    x = "Age",
    y = "Plasma Glucose Concentration",
    color = "Predicted Diabetes"
  ) +
  theme_minimal() +  # Clean theme
  scale_color_manual(values = c("green", "red"), labels = c("No",
     ↪  "Yes")) # Customize color/label for Diabetes


#### Assignment 1.3 ####
# Decision boundary of the estimated logistic regression model

# The line where probability of y=1 is equal to 0.5 <==> logit
   ↪ function = 0
# logit(P(y=1)) = w0 + w1*Glucose + w2*Age = 0 <==> Age = -(w0/w2
   ↪ )-(w1*Glucose)/w2

# Function to calculate decision boundary between classes Glucose
   ↪  & Age
decision_boundary <- function(glucose) {
  -(intercept/coef_age)-(coef_glucose/coef_age)*glucose
}

```

```r
# Define Glucose range for boundary line
glucose_range <- data$PlasmaGlucoseConcentration # Use all Glucose values
age_boundary <- decision_boundary(glucose_range)

# Add the decision boundary in predicted Diabetes plot
ggplot(data, aes(x = Age, y = PlasmaGlucoseConcentration, color = factor(Predicted_Classifications))) + # Define x/y-axes
  geom_point(alpha = 0.6) +  # Add points with 40% transparency
  # Decision boundary line
  geom_line(aes(x = age_boundary, y = glucose_range), color="blue", linetype="dashed", size=0.5) +
  labs( # Labels for the plot
    title = "Plasma Glucose Concentration vs Age",
    x = "Age",
    y = "Plasma Glucose Concentration",
    color = "Predicted Diabetes"
  ) +
  xlim(min(data$Age), max(data$Age)) + # Restrict the x-axis to the observed range of Age
  theme_minimal() +  # Clean theme
  scale_color_manual(values = c("green", "red"), labels = c("No", "Yes")) # Customize color/label for Diabetes


#### Assignment 1.4 ####
# Plotted graphs based on different r-values

# Convert predicted probabilities into binary classifications based on threshold = 0.2
data$Predicted_Classifications_02 <- ifelse(data$Predicted_Probabilities > 0.2, 1, 0) # Values either 0 or 1
# Convert predicted probabilities into binary classifications based on threshold = 0.8
data$Predicted_Classifications_08 <- ifelse(data$Predicted_Probabilities > 0.8, 1, 0) # Values either 0 or 1

# Plot r = 0.2
ggplot(data, aes(x = Age, y = PlasmaGlucoseConcentration, color = factor(Predicted_Classifications_02))) + # Define x/y-axes
  geom_point(alpha = 0.6) +  # Add points with 40% transparency
  labs( # Labels for the plot
    title = "Plasma Glucose Concentration vs Age",
    x = "Age",
    y = "Plasma Glucose Concentration",
    color = "Predicted Diabetes"
  ) +
  theme_minimal() +  # Clean theme
  scale_color_manual(values = c("green", "red"), labels = c("No", "Yes")) # Customize color/label for Diabetes

```

```r
# Plot r = 0.8
ggplot(data, aes(x = Age, y = PlasmaGlucoseConcentration, color = factor(Predicted_Classifications_08))) + # Define x/y-axes
  geom_point(alpha = 0.6) +  # Add points with 40% transparency
  labs( # Labels for the plot
    title = "Plasma Glucose Concentration vs Age",
    x = "Age",
    y = "Plasma Glucose Concentration",
    color = "Predicted Diabetes"
  ) +
  theme_minimal() +  # Clean theme
  scale_color_manual(values = c("green", "red"), labels = c("No", "Yes")) # Customize color/label for Diabetes


#### Assignment 1.5 ####
# Basis function expansion to introduce polynomial interaction terms into the logistic regression model
# x1 = PlasmaGlucoseConcentration, x2 = Age

# Compute new features for basis expansion and add these to the dataset as new columns
data$z1 <- data$PlasmaGlucoseConcentration^4
data$z2 <- data$PlasmaGlucoseConcentration^3 * data$Age
data$z3 <- data$PlasmaGlucoseConcentration^2 * data$Age^2
data$z4 <- data$PlasmaGlucoseConcentration * data$Age^3
data$z5 <- data$Age^4

# Train the new logistic regression model using expanded new features z1...z5
logistic_model_expanded <- glm(Diabetes ~ PlasmaGlucoseConcentration + Age + z1 + z2 + z3 + z4 + z5,
                               data = data,
                               family = binomial)

# Extract predicted values and classify them to 0 or 1 with r = 0.5
data$Predicted_Probabilities_Expanded <- predict(logistic_model_expanded, type = "response") # Values between 0 and 1
data$Predicted_Classifications_Expanded <- ifelse(data$Predicted_Probabilities_Expanded > 0.5, 1, 0) # Values either 0 or 1

# Training misclassification error
error_expanded <- mean(data$Predicted_Classifications_Expanded != data$Diabetes)
cat(sprintf("\nTraining Misclassification Error (Expanded Model): %.2f%%\n", error_expanded * 100)) # Print the result

# Scatterplot of predicted Diabetes with new model
ggplot(data, aes(x = Age, y = PlasmaGlucoseConcentration, color = factor(Predicted_Classifications_Expanded))) + # Define x/y
```

```
    ↪ -axes
367 geom_point(alpha = 0.6) +  # Add points with 40% transparency
368  labs( # Labels for the plot
369   title = "Plasma Glucose Concentration vs Age (Basis Function
        ↪ Expansion)",
370   x = "Age",
371   y = "Plasma Glucose Concentration",
372   color = "Predicted Diabetes"
373 ) +
374 theme_minimal() +  # Clean theme
375 scale_color_manual(values = c("green", "red"), labels = c("No",
      ↪  "Yes")) # Customize color/label for Diabetes
```