# Computer lab 1

Andreas Arrestam
Emma Bertmar
Manfred Clase

LINKÖPINGS UNIVERSITET

# Contents

## Statement of Contribution

During Lab 1, each team member was responsible for one assignment, which included both the code and the corresponding part of the report. Andreas Arrestam was responsible for Assignment 1, Manfred Clase for Assignment 2, and Emma Bertmar for Assignment 3. However, all team members discussed the tasks together and supported each other in completing the code and writing the report sections. Once all parts were finished, the members sat together to review the problems and solutions. This ensured that every group member understood the code, the results, and the conclusions.

# 1. Assignment 1. Handwritten digit recognition with K-nearest neighbours.

The purpose of assignment 1 was to explore the K-nearest neighbours model and to investigate how varying K affects the error.

## 1.1 Data partitioning

For this assignment the training set was 50%, the validation set 25% and testing set 25%. To reduce bias the dataset is randomized before it is divided into the different sets.

- **Training set**: Used to fit the model.
- **Validation set**: Used to tune the hyperparameters.
- **Test set**: Used to evaluate the performance of the model.

## 1.2 K-nearest neighbour model

The K-nearest neighbour model takes an input and finds the k closest data points and makes a prediction based on the majority class of the k closest neighbours. For this exercise, the model was applied to both the test and training data using K=30 and a rectangular kernel. First, the training data was predicted based on the training data points. Then, the test data are classified by comparing them to the training data points (see Listing 1.1).

Listing 1.1: Implementation of K-nearest neighbour model

```
fitted_traindata = kknn(Digit~., train, train, k=30, kernel = "rectangular")
fitted_testdata = kknn(Digit~., train, test, k=30, kernel = "rectangular")

pred_train <-fitted_traindata$fitted.values #the average label between the 30
    nearest neighbours
pred_test <-fitted_testdata$fitted.values
```

Confusion matrix for training data.

| True/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 |

Confusion matrix for test data.

| True/Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 77 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 84 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 109 | 0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | 103 | 0 | 1 | 2 | 0 | 3 |
| 5 | 0 | 0 | 1 | 0 | 0 | 100 | 0 | 0 | 0 | 2 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 90 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 111 | 0 | 0 |
| 8 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 74 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 86 |

Based on the confusion matrices, the misclassification rate for both test and training sets was computed:

$$\text{Training misclassification rate} = 1 - \frac{\text{Number of correct predictions}}{\text{Total number of observations}} = 1 - \frac{1825}{1911} \approx 0,0450$$

$$\text{Test misclassification rate} = 1 - \frac{\text{Number of correct predictions}}{\text{Total number of observations}} = 1 - \frac{934}{957} \approx 0,0533$$

The confusion matrix outputs the predicted classifications for the true digits. It shows that some digits were more difficult to classify than others. For the train test the most difficult digits to classify were the digits 4, 8 and 9 but for the test data it was the digits 1, 4 and 8.

The misclassification rate for the train set was 4.5% and for the test set it was 5.33%. These values are very low which indicates that the model performs well.

## 1.3 Cases of digit 8

Two cases of digit ”8” which were easiest to classify were identified (see Figures 1.1, 1.2), and three cases of digit ”8” that were hardest to classify were also identified (see Figures 1.3, 1.4, 1.5). After identifying these cases, a heat map was created to visualize the different cases.

The two cases that were easiest to classify are easy to recognize visually, as they have the characteristics of the digit 8. But the cases that were difficult for the model to classify are also not easy to recognize with the human eye, as these has no or very little similarity to the characteristics of an 8.

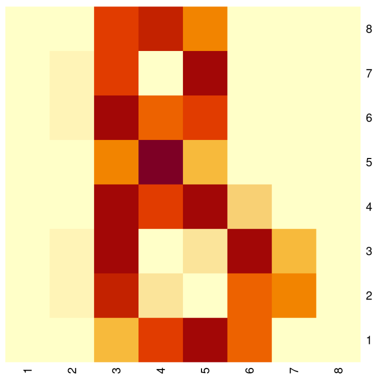

Figure 1.1: Case 1 easy classification.

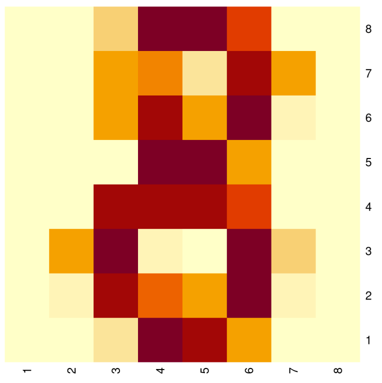

Figure 1.2: Case 2 easy classification.

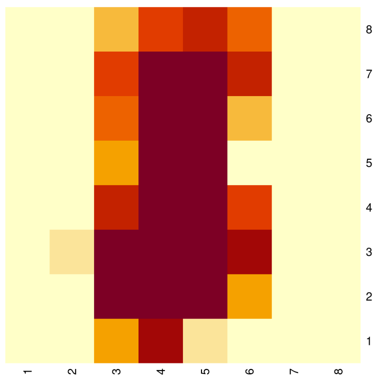

Figure 1.3: Case 1 difficult classification.

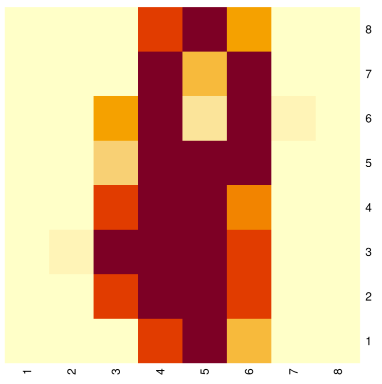

Figure 1.4: Case 2 difficult classification.

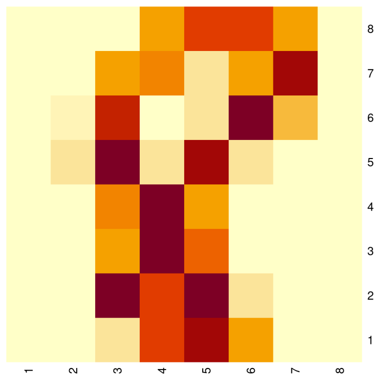

Figure 1.5: Case 3 difficult classification.

## 1.4 Training and validation errors

A larger K makes the model less complex and reduces variance by averaging over more neighbours. When increasing the K-value it is shown in Figure 1.6 that the error for the validation set and the training set is increasing but for K=3 the validation error is minimum. This is the optimal K-value and using the optimal K-value the error for the test set is 2.4%.

The pattern, training error < test error < validation error. This pattern is reasonable given that the validation set was used for tuning K. This indicates good generalization performance and minimal overfitting.

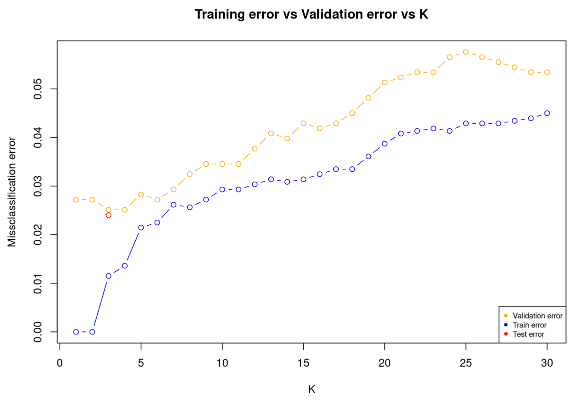

Figure 1.6: Misclassification error with varying K-value.

## 1.5 Cross-entropy validation error

The cross-entropy error is calculated accordingly:

$$H = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{C} y_{n,i}\log(p_{n,i})$$

The cross-entropy validation error with a varying K is shown in Figure 1.7. From the graph, the optimal K-value can be found to be K=6.

The cross entropy is a more suitable choice because it takes the predicted probability into consideration when calculating the loss. It penalizes low confidence probabilities and rewards high confidence ones. For example, predicting a class with a probability of 0.51 is very different from predicting it with 0.99, which the misclassification rate does not take into consideration.

Listing 1.2: Code for cross-entropy function

```
true_labels<-as.numeric(valid$Digit)
n<-nrow(valid)

y_true <- matrix(0, nrow = n, ncol = ncol(fit_valid$prob))
for (k in 1:nrow(valid)) {
y_true[k, true_labels[k]] <- 1
}
cross_entropy[i] = -mean(rowSums(y_true*log(fit_valid$prob + 1e-15)))
```

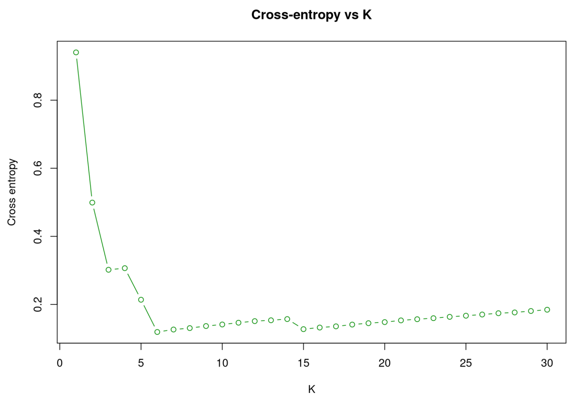

Figure 1.7: Cross-entropy with varying K-value

# 2. Assignment 2. Linear regression and ridge regression

The purpose of this assignment was to explore the Linear Regression Model and different related values and methods that can be used to understand and optimize the model. The data provided that the model was trained on was provided int the file **parkinsons.csv** which contained different biomedical voice measurements of 42 people.

## 2.1 Data Partitioning

The Linear Regression model is, similar to the KNN-model, also based on the holdout method. The dataset is split into two sets:

- **Training set**: Used to fit the model.
- **Test set**: Used to evaluate the performance of the model.

The dataset was randomized and then split into the two abovementioned sets. The proportions of the set sizes was 60% training set, 40% test set.

Before training, the input features were scaled to ensure consistent feature ranges. This preprocessing step was performed using the `caret` package in R, where scaling parameters were learned from the training set and then applied to both the training and test sets. This is how it was implemented in our solution.

Listing 2.1: Scaling

```
# scale data
  library(caret)

  scaler <- preProcess(df)
  train_scaled <- predict(scaler, train_df)
  test_scaled <- predict(scaler, test_df)
```

## 2.2 Linear Regression model

The Linear Regression model is a so-called regression model. A regression model is a statistical tool for estimating a continuous value given inputs. In this model we try to fit the dataset onto a straight line that best represents the relationship different input data and target value.

The Linear Regression prediction value $\hat{y}$ can be written as follows:

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

In this assignment we applied the MSE (Mean Squared Error) as our loss-function. The purpose of the loss-function is calculate the loss of a certain instance of coefficient, this value is to be minimized and thus the model's prediction properties should improve over a given dataset.

The Mean Squared Error loss function is defined as:

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

The model was then trained on the scaled training data set. When we have retreived our model from training we can calculate our new predictions and MSE on our two datasets. The code section below was used to perform this.

Listing 2.2: Training

```
model=lm(motor_UPDRS~., train_scaled)

#Calculate prediction and MSE on the training data set
Preds_train=predict(model, newdata=train_scaled)
MSE_train=mean((train_df$motor_UPDRS-Preds_train)^2)

#Calculate prediction and MSE on the test data set
Preds_test=predict(model, newdata=test_scaled)
MSE_test=mean((test_df$motor_UPDRS-Preds_test)^2)
```

The model yielded the following Mean Squared Error of the two datasets:

Table 2.1: Mean Squared Error of the Linear Regression model on training and test sets.

| **Dataset** | **Mean Squared Error (MSE)** |
|---|---|
| Training Set | 0.8785431 |
| Test Set | 0.9354477 |

By running `summary(model)` some useful information about the features and their coefficients can be presented. Among these are the p-value, which is useful to determine the significance of the model's features. This is the output of this call:

Table 2.2: Linear regression coefficients, standard errors, t-values, and p-values for the scaled Parkinson's dataset. Significance codes: *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

| **Variable** | **Estimate** | **Std. Error** | **t value** | **Pr(>\|t\|)** | **Significance** |
|---|---|---|---|---|---|
| (Intercept) | 6.575e-15 | 1.583e-02 | 0.000 | 1.000000 | |
| Jitter... | 1.869e-01 | 1.496e-01 | 1.250 | 0.211496 | |
| Jitter.Abs. | -1.696e-01 | 4.081e-02 | -4.156 | 3.32e-05 | *** |
| Jitter.RAP | -5.270e+00 | 1.884e+01 | -0.280 | 0.779688 | |
| Jitter.PPQ5 | -7.457e-02 | 8.778e-02 | -0.850 | 0.395659 | |
| Jitter.DDP | 5.250e+00 | 1.884e+01 | 0.279 | 0.780541 | |
| Shimmer | 5.924e-01 | 2.060e-01 | 2.876 | 0.004055 | ** |
| Shimmer.dB. | -1.727e-01 | 1.393e-01 | -1.239 | 0.215380 | |
| Shimmer.APQ3 | 3.207e+01 | 7.717e+01 | 0.416 | 0.677738 | |
| Shimmer.APQ5 | -3.875e-01 | 1.138e-01 | -3.405 | 0.000669 | *** |
| Shimmer.APQ11 | 3.055e-01 | 6.124e-02 | 4.989 | 6.37e-07 | *** |
| Shimmer.DDA | -3.239e+01 | 7.717e+01 | -0.420 | 0.674739 | |
| NHR | -1.854e-01 | 4.557e-02 | -4.068 | 4.85e-05 | *** |
| HNR | -2.385e-01 | 3.640e-02 | -6.553 | 6.45e-11 | *** |
| RPDE | 4.068e-03 | 2.267e-02 | 0.179 | 0.857576 | |
| DFA | -2.803e-01 | 2.014e-02 | -13.919 | <2e-16 | *** |
| PPE | 2.265e-01 | 3.289e-02 | 6.886 | 6.75e-12 | *** |

From the data, it was concluded that the coefficients marked with one or more `*` should be regarded as significant.

## 2.3 Auxillary Functions

The following functions were implemented as part of the assignment.

### 2.3.1 Log Likelihood

The log-likelihood function measures how probable the observed data $y$ is given the model parameters $\theta$ and error standard deviation $\sigma$. The log-likelihood was implemented in the solution as follows:

Listing 2.3: loglike-function

```
loglike <- function(theta, sigma){
  n = length(y)
  return (-n/2 * log(2*pi*sigma^2) - 1/(2*sigma^2) * sum((y - x %*% theta)^2))
}
```

### 2.3.2 Ridge

Ridge regression adds a penalty on the squared magnitude of the coefficients to prevent overfitting. The Ridge objective function is:

$$\text{Ridge}(\theta, \sigma; \lambda) = -\log \mathcal{L}(\theta, \sigma) + \lambda \|\theta\|_2^2$$

The Ridge-penalty function was implemented in the solution as follows:

Listing 2.4: ridge-function

```
ridge <- function(theta, sigma,lambda){
  penalty_value <- lambda * sum(theta[-1]^2)
  return(-loglike(theta, sigma) + penalty_value)
}
```

### 2.3.3 RidgeOpt

This function finds the optimal coefficients $\theta$ and $\sigma$ for a given penalty $\lambda$ by minimizing the Ridge objective using numerical optimization:

$$(\hat{\theta}, \hat{\sigma}) = \arg\min_{\theta,\sigma} \left[ -\log \mathcal{L}(\theta, \sigma) + \lambda \|\theta\|_2^2 \right]$$

The optimization is performed using the BFGS algorithm via R's `optim()` function.

The RidgeOpt function was implemented in the solution as follows:

Listing 2.5: ridge$_opt - function$

```
ridge_opt <- function(lambda){
  theta_init <- coef(model)
  sigma_init <- sigma(model)
  params_init <- c(theta_init, sigma_init)

  ridge_wrapper <- function(parameter_vector){
    n_theta <- length(theta_init)
    theta <- parameter_vector[1:n_theta]
    sigma <- parameter_vector[n_theta + 1]

    return(ridge(theta, sigma, lambda))
  }

  result <- optim(par=params_init, fn=ridge_wrapper, method="BFGS")
  return (result)
}
```

### 2.3.4 DF

The degrees of freedom in Ridge regression quantify the effective number of parameters after regularization:

$$\text{DF}(\lambda) = \text{trace}\big(X(X^\top X + \lambda I)^{-1} X^\top\big)$$

Here, $X$ is the design matrix and $I$ is the identity matrix. Larger $\lambda$ reduces the degrees of freedom, effectively shrinking the model complexity.

Listing 2.6: DF-function

```
DF <- function(X, lambda) {
  n_features <- ncol(X)
  I <- diag(n_features)          # identity matrix
  XtX <- t(X) %*% X
  H <- X %*% solve(XtX + lambda * I) %*% t(X)  # "hat" matrix
  df <- sum(diag(H))              # trace of H
  return(df)
}
```

## 2.4 Ridge Regression Results and Model Comparison

Using the `RidgeOpt` function, the optimal parameter vector $\theta$ was estimated for three different regularization parameters: $\lambda = 1$, $\lambda = 100$, and $\lambda = 1000$. These parameters were then used to predict the *motor_UPDRS* values for both the training and test datasets, and the Mean Squared Errors (MSE) were computed.

The results are summarized in Table 2.3.

Table 2.3: Ridge regression performance and model complexity for different values of $\lambda$.

| $\lambda$ | Training MSE | Test MSE | Degrees of Freedom |
|---|---|---|---|
| 1 | 4.261 | 4.283 | 13.861 |
| 100 | 1.137 | 1.179 | 9.925 |
| 1000 | 0.977 | 0.996 | 5.644 |

From the results, it can be observed that increasing the regularization parameter $\lambda$ leads to a reduction in both training and test MSE, as well as a decrease in the degrees of freedom. This indicates that stronger regularization improves generalization performance while reducing model complexity.

Among the tested values, $\lambda = 1000$ is the most appropriate choice, as it achieves the lowest test MSE while also having the smallest degrees of freedom. This suggests that the model with $\lambda = 1000$ provides the best balance between bias and variance and is less prone to overfitting.

### Conclusion

A stronger ridge penalty results in better predictive performance for this dataset. Therefore, based on both error metrics and model complexity, the model with $\lambda = 1000$ is preferred.

### Complete Codebase

Code can be found at https://gitlab.liu.se/mancl774/tdde01-labs.

# 3. Assignment 3. Logistic regression and basis function expansion

The purpose of this assignment was to explore logistic regression models and to investigate how basis function expansion affected the model's predictions.

## 3.1 Initial visualization

A scatter plot was created showing *Plasma Glucose concentration* versus *Age*, with observations colored according to diabetes status (see Figure 3.1). Orange data points represent individuals diagnosed with diabetes, while blue points represent non-diabetic individuals. The plot shows that classifying diabetes using a standard logistic regression model is difficult, although Plasma Glucose separates the groups to some extent.

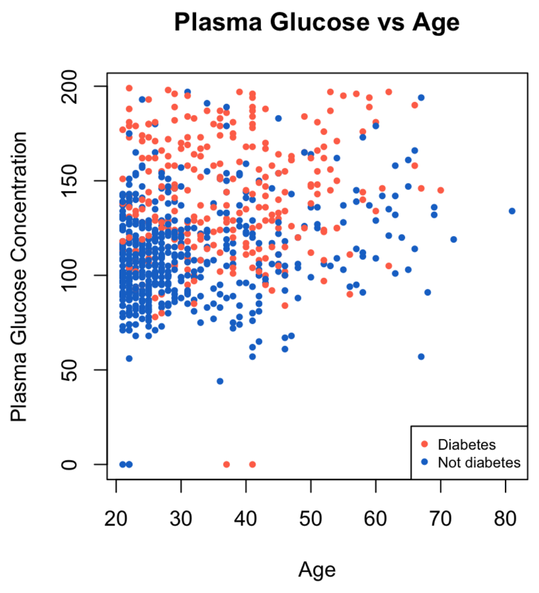

Figure 3.1: Scatter plot with data points colored based on real diabetes status.

## 3.2 Logistic regression model

A logistic regression model was trained using $y$ = Diabetes as the target and $x_1$ = Plasma glucose concentration and $x_2$ = Age as features, with a classification threshold of $r = 0.5$. The general probabilistic form of a logistic regression model is

$$\Pr(y = 1) = \frac{1}{1 + \exp\left(-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)\right)}$$

where the coefficients $\beta$ were obtained from training the model (Listing 3.1). For the given dataset, the fitted model is

$$\Pr(\text{Diabetes} = 1) = \frac{1}{1 + \exp\big(5.912449 - 0.035644 \cdot \text{Plasma glucose concentration} - 0.024778 \cdot \text{Age}\big)}$$

Listing 3.1: Training a logistic regression model

```
# Fit a logistic regression model
model = glm(diabetes ~ glucose_2h + age,
            data = dataframe, family = "binomial")
summary(model)

# Predict probabilities from the model and classify using threshold r = 0.5
probability = predict(model, type = "response")
prediction = ifelse(probability > 0.5, 1, 0)
```

A scatter plot was created showing *Plasma Glucose concentration* versus *Age*, with observations colored according to predicted diabetes status, see Figure 3.2. Orange data points represent individuals predicted to have diabetes, while blue points represent individuals predicted to be non-diabetic.

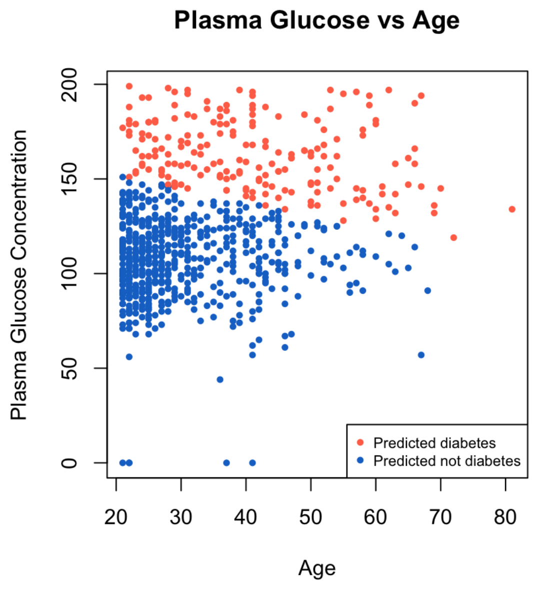

Figure 3.2: Scatter plot of trained logistic regression model with data points colored based on predicted diabetes status.

The performance of the logistic regression model was evaluated using a confusion matrix and the training misclassification rate. From the code in Listing 3.2, the confusion matrix was obtained as

| True\Predicted | 0 | 1 |
|---|---|---|
| 0 | 436 | 64 |
| 1 | 138 | 130 |

The training misclassification rate was then calculated as the fraction of incorrectly classified observations:

$$\text{Training misclassification rate} = \frac{\text{Number of incorrect predictions}}{\text{Total number of observations}} = \frac{64+138}{436+64+138+130} \approx 0,26$$

The training misclassification rate of 0.26 is quite high, indicating that the model does not classify diabetes very accurately when using only Plasma Glucose concentration and Age as features.

Listing 3.2: Creating confusion matrix

```
# Confusion matrix
confusion_matrix <- table(dataframe$diabetes, prediction)
print(confusion_matrix)

# Misclassification rate
```

```
incorrect <- sum(confusion_matrix) - sum(diag(confusion_matrix))
misclassification <- incorrect/sum(confusion_matrix)
print(paste("Misclassification rate = ", misclassification))
```

## 3.3 Decision Boundary Analysis

A decision boundary line for the classification specified in Section 3.2 has been calculated. In a binary classification where the threshold is set to $r = 0.5$, the decision boundary is the point where the probability of $y = 1$ equals 0.5, which corresponds to the following equation:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0.$$

Solving for $x_1$ gives

$$x_1 = -\frac{\beta_0 + \beta_2 x_2}{\beta_1}.$$

And with the features used in this assignment, this becomes

$$\text{Plasma glucose concentration} = -\frac{\beta_0}{\beta_1} - \frac{\beta_2}{\beta_1} \cdot \text{Age}.$$

This boundary line is added to the scatter plot of the predicted diabetes status, as shown in Listing 3.3. The resulting scatter plot, including the decision boundary line, is shown in Figure 3.3. The plot shows that the decision boundary captures the general trend in the data relatively well. However, there is some overlap between the two classes indicating some misclassified data points.

Listing 3.3: Add decision boundary.

```
# Add decision boundary line from logistic regression coefficients
abline(a = -coef(model)[1]/coef(model)[2],  # intercept
       b = -coef(model)[3]/coef(model)[2],  # slope
       col = "black", lwd = 1.5)
```

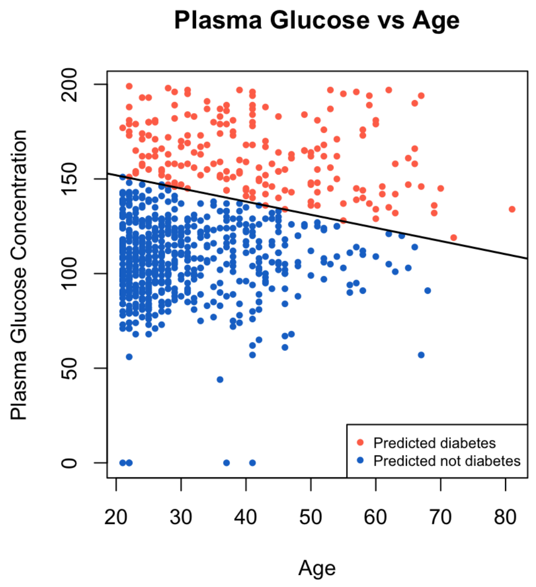

Figure 3.3: Scatterplot of predicted diabetes status with decision boundary line.

## 3.4 Threshold Analysis

Different classification thresholds were tested using the same calculations as in Section 3.2, but with the thresholds $r = 0.2$ and $r = 0.8$ instead of $r = 0.5$.

When using the threshold $r = 0.2$, more individuals were classified as diabetic than with $r = 0.5$ (Figure 3.4). The resulting misclassification rate was approximately 0.37. When using the threshold $r = 0.8$, fewer individuals were classified as diabetic than with $r = 0.5$ (Figure 3.5). The resulting misclassification rate was approximately 0.32. Both misclassification rates were higher than before, indicating worse performance.

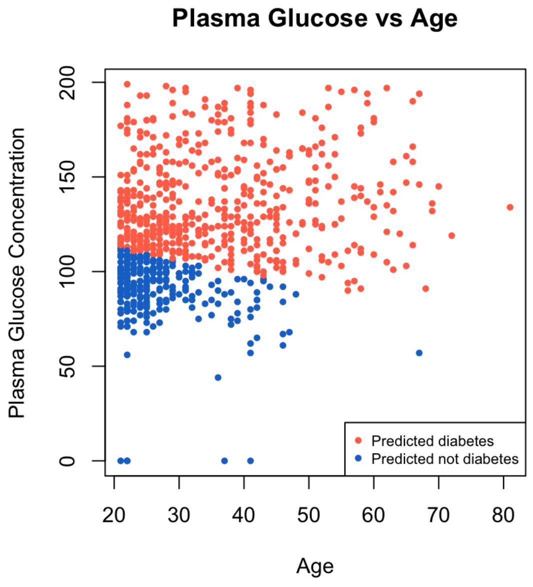

Figure 3.4: Scatter plot of predicted diabetes status with threshold $r = 0.2$.

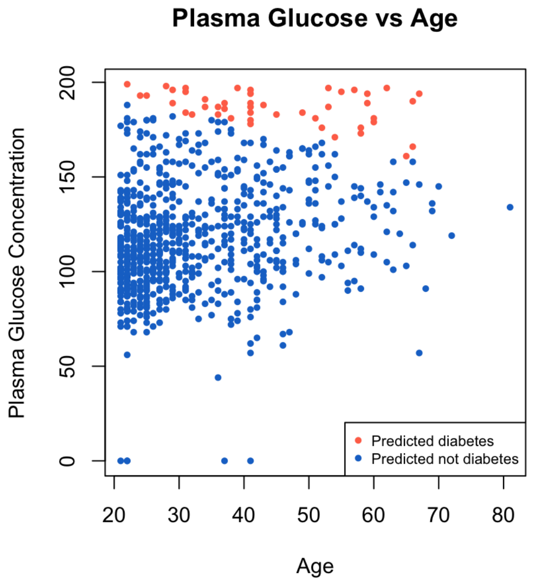

Figure 3.5: Scatter plot of predicted diabetes status with threshold $r = 0.8$.

## 3.5 Basis Function Expansion

A basis function expansion was performed, and the expanded model was trained using the code provided in Listing 3.4.

The misclassification rate for the expanded model was approximately 0.24, which is slightly reduced compared to the rate of 0.26 in the previous model. Even though the expanded model fits the data better, its increased complexity might lead to overfitting. The misclassification rate remains relatively high, confirming that predicting diabetes based solely on Age and Plasma Glucose Concentration is difficult.

The basis expansion trick also affected the decision boundary line. As shown in Figure 3.6, the boundary was transformed from a linear shape into a curved line.

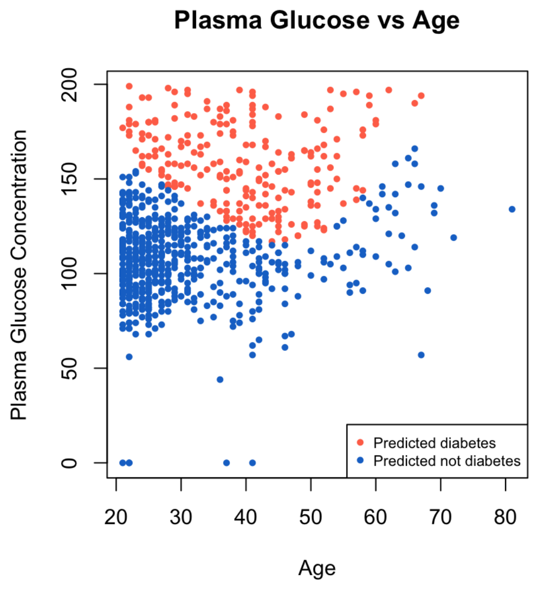

Figure 3.6: Scatter plot of predicted diabetes status with Basis Expansion Model.

Listing 3.4: Basis function expansion

```
# Fit logistic regression with basis-expanded features
model2 = glm(diabetes ~ glucose_2h + age + z1 + z2 + z3 + z4 + z5,
             data = dataframe, family = "binomial")
summary(model2)

# Calculate probabilities
probability_basis = predict(model2, type = "response")
prediction_basis = ifelse(probability_basis > 0.5, 1, 0)
```

# 4. Assignment 4. Theory

This chapter answers some questions using the course book *Machine Learning A First Course for Engineers and Scientists* (2022), written by Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten and Thomas B. Schön.

## 4.1 Why can it be important to consider various probability thresholds in the classification problems, according to the book?

According to the book, it can be important to consider different probability thresholds in classification problems. If the classification model describes the true class probabilities well, then using the threshold r = 0.5 gives the lowest average misclassification rate. However, many real-world classification problems are asymmetric or imbalanced, meaning some type of errors may be more costly than others or one class may be more common than the other. In these cases, adjusting the threshold could lead to better decisions for the specific goal, even if the misclassification rate becomes higher (pages 49-50).

## 4.2 What ways of collecting correct values of the target variable for the supervised learning problems are mentioned in the book?

The book describes two main ways to collect correct values of the target variable for supervised learning. In some cases, the output $y$ can be collected together with the input $x$, so the correct target values are already available. In other cases, the output cannot be observed directly and has to be labeled manually by a domain expert (pages 13–14).

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

The cost function is defined as the average loss over the training data. For the linear regression model, the cost function can be expressed in matrix notation as

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

where $\|\cdot\|_2$ denotes the usual Euclidean vector norm and $\|\cdot\|_2^2$ its square. This is taken from formula (3.11) in the book (pages 40-41).

# A. Code

## A.1 Full R code for assignment 1

```
install.packages('readxl')
install.packages('kknn')
library(readxl)
library(kknn)

#Each row in dataset is the image of a handwritten digit, each column
   represents the pixel intensity values and
#And the last column shows the actual digit 0-9.

#### Exxercise 1####
data = read.csv("/home/arre/Universitet/TDDE01/Lab1/optdigits.csv", header =
   FALSE)

colnames(data)[ncol(data)] <- "Digit" #give a label to last column

data[,ncol(data)] <- as.factor(data[,ncol(data)]) #makes the target variable
   categorical

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

id1 = setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid = data[id2,]

id3 = setdiff(id1, id2)
test = data[id3,]

missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

#### Exercise 2####


fitted_traindata = kknn(Digit~., train, train, k=30, kernel = "rectangular")
fitted_testdata = kknn(Digit~., train, test, k=30, kernel = "rectangular")
```

```
pred_train <-fitted_traindata$fitted.values #the average label between the 30
    nearest neighbours
pred_test <-fitted_testdata$fitted.values


#Each row predicted digit, each column actual digit
cm_test <- table(test$Digit, pred_test)
cm_train <- table(train$Digit, pred_train)


test_miss_rate <-missclass(pred_test, test$Digit)
train_miss_rate <-missclass(pred_train, train$Digit)

print(cm_test)
print(cm_train)
print(test_miss_rate)
print(train_miss_rate)

#End

#### Exercise 3####
train_probs_8 = fitted_traindata$prob[, "8"]
train_8_indices <-which(train$Digit == "8")

#vector where all elements correspond to one example where 8 is the true digit
train_probs_actual_8 <-train_probs_8[train_8_indices]


easiest_8idx <- order(train_probs_actual_8, decreasing = TRUE)[1:2]
hardest_8idx <- order(train_probs_actual_8)[1:3]

easiest_rows <- train_8_indices[easiest_8idx]
hardest_rows <- train_8_indices[hardest_8idx]

#Case 1
pixels <- train[easiest_rows[1], -ncol(train)]
pixels_matrix<- matrix(as.numeric(pixels), nrow = 8, ncol = 8, byrow = TRUE)
heatmap(pixels_matrix, Colv = NA, Rowv = NA, scale = "none")

#Case 25
pixels <- train[easiest_rows[2], -ncol(train)]
pixels_matrix<- matrix(as.numeric(pixels), nrow = 8, ncol = 8, byrow = TRUE)
heatmap(pixels_matrix, Colv = NA, Rowv = NA, scale = "none")

#Case 7
pixels <- train[hardest_rows[1], -ncol(train)]
pixels_matrix<- matrix(as.numeric(pixels), nrow = 8, ncol = 8, byrow = TRUE)
heatmap(pixels_matrix, Colv = NA, Rowv = NA, scale = "none")
```

```
#Case 30
pixels <- train[hardest_rows[2], -ncol(train)]
pixels_matrix<- matrix(as.numeric(pixels), nrow = 8, ncol = 8, byrow = TRUE)
heatmap(pixels_matrix, Colv = NA, Rowv = NA, scale = "none")

#Case 72
pixels <- train[hardest_rows[3], -ncol(train)]
pixels_matrix<- matrix(as.numeric(pixels), nrow = 8, ncol = 8, byrow = TRUE)
heatmap(pixels_matrix, Colv = NA, Rowv = NA, scale = "none")

####Exercise 4####
error_rate_train <- numeric(30)
error_rate_valid <- numeric(30)
cross_entropy <- numeric(30)
for (i in 1:30){
  fit_train = kknn(Digit~., train, train, k=i, kernel = "rectangular", scale =
       TRUE)
  fit_valid = kknn(Digit~., train, valid, k=i, kernel = "rectangular", scale =
       TRUE)

  pred_train2 <- fit_train$fitted.values
  pred_valid <- fit_valid$fitted.values

  error_rate_train[i] <-missclass(pred_train2, train$Digit)
  error_rate_valid[i] <-missclass(pred_valid, valid$Digit)

  true_labels<-as.numeric(valid$Digit)
  n<-nrow(valid)

  y_true <- matrix(0, nrow = n, ncol = ncol(fit_valid$prob))
  for (k in 1:nrow(valid)) {
    y_true[k, true_labels[k]] <- 1
  }

  cross_entropy[i] = -mean(rowSums(y_true*log(fit_valid$prob + 1e-15)))

}
best_k <- which.min(error_rate_valid)
fit_test = kknn(Digit~., train, test, k=best_k, kernel = "rectangular", scale
    = TRUE)
pred_test <-fit_test$fitted.values
error_test2 <-missclass(pred_test, test$Digit)

plot(1:30, error_rate_valid, type = "b", col="orange", main="Training error vs
     Validation error vs K", xlab="K",
      ylab="Missclassification error", ylim = range(c(error_rate_train, error_
         rate_valid)))
lines(1:30, error_rate_train, type="b", col="blue")
points(best_k, error_test2, type = "b", col="red")
```

```
legend("bottomright", legend = c("Validation error", "Train error", "Test
   error"),
       col = c("orange", "blue", "red"), pch = 16, cex = 0.7)

####Exercise 5####
plot(1:30, cross_entropy, type="b", col="green4", main="Cross-entropy vs K",
   xlab = "K", ylab = "Cross entropy")
optimal_k <- which.min(cross_entropy)
```

## A.2 Full R code for assignment 2

```

#ASSIGNMENT 2

parkinsons = read.csv("parkinsons.csv", header=TRUE)

  n=dim(parkinsons)[1]
  set.seed(12345)
  id=sample(1:n, floor(n*0.6))
  train=parkinsons[id,]
  test=parkinsons[-id,]

  library(dplyr)

  df=train%>%select(motor_UPDRS, Jitter...:PPE)
  train_df <- train %>% select(motor_UPDRS, Jitter...:PPE)
  test_df  <- test  %>% select(motor_UPDRS, Jitter...:PPE)

  # scale data
  library(caret)

  scaler <- preProcess(df)
  train_scaled <- predict(scaler, train_df)
  test_scaled <- predict(scaler, test_df)

  # Select features
  train_df <- train_scaled %>% select(motor_UPDRS, Jitter...:PPE)
  test_df  <- test_scaled  %>% select(motor_UPDRS, Jitter...:PPE)


  model=lm(motor_UPDRS~., train_scaled)

  #Calculate prediction and MSE on the training data set
  Preds_train=predict(model, newdata=train_scaled)
  MSE_train=mean((train_df$motor_UPDRS-Preds_train)^2)

  #Calculate prediction and MSE on the test data set
  Preds_test=predict(model, newdata=test_scaled)
```

```
MSE_test=mean((test_df$motor_UPDRS-Preds_test)^2)

#

# Correct X for test set
x <- model.matrix(motor_UPDRS ~ ., data = test_scaled)
y <- test$motor_UPDRS

loglike <- function(theta, sigma){
  n = length(y)
  return (-n/2 * log(2*pi*sigma^2) - 1/(2*sigma^2) * sum((y - x %*% theta)
      ^2))
}

ridge <- function(theta, sigma,lambda){
  penalty_value <- lambda * sum(theta[-1]^2)
  return(-loglike(theta, sigma) + penalty_value)
}

# theta <- coef(model)
# sigma <- sigma(model)
# loglike(theta, sigma(model))
# ridge(theta, sigma, 1)

ridge_opt <- function(lambda){
  theta_init <- coef(model)
  sigma_init <- sigma(model)
  params_init <- c(theta_init, sigma_init)

  ridge_wrapper <- function(parameter_vector){
    n_theta <- length(theta_init)
    theta <- parameter_vector[1:n_theta]
    sigma <- parameter_vector[n_theta + 1]

    return(ridge(theta, sigma, lambda))
  }

  result <- optim(par=params_init, fn=ridge_wrapper, method="BFGS")
  return (result)
}

DF <- function(X, lambda){
  XtX <- t(X) %*% X
  eig_vals <- eigen(XtX, symmetric = TRUE)$values
  df <- sum(eig_vals / (eig_vals + lambda))
  return(df)
}

# compute optimal Ridge coefficients for different lambda values
ridge_1    <- ridge_opt(lambda = 1)
```

```
  ridge_100  <- ridge_opt(lambda = 100)
  ridge_1000 <- ridge_opt(lambda = 1000)

  # Prepare feature matrices
  X_train <- as.matrix(train_scaled %>% select(Jitter...:PPE))
  X_test  <- as.matrix(test_scaled  %>% select(Jitter...:PPE))

  # Extract only the theta coefficients (exclude sigma)
  theta_1    <- as.matrix(ridge_1$par[-length(ridge_1$par)])
  theta_100  <- as.matrix(ridge_100$par[-length(ridge_100$par)])
  theta_1000 <- as.matrix(ridge_1000$par[-length(ridge_1000$par)])

  # Predictions

  # remove intercept from coefficients
  theta_1_no_intercept    <- theta_1[-1]
  theta_100_no_intercept  <- theta_100[-1]
  theta_1000_no_intercept <- theta_1000[-1]

  # prdictions for training and test sets
  y_train_pred_1    <- X_train %*% theta_1_no_intercept
  y_test_pred_1     <- X_test  %*% theta_1_no_intercept

  y_train_pred_100  <- X_train %*% theta_100_no_intercept
  y_test_pred_100   <- X_test  %*% theta_100_no_intercept

  y_train_pred_1000 <- X_train %*% theta_1000_no_intercept
  y_test_pred_1000  <- X_test  %*% theta_1000_no_intercept

  # Compute training and test MSE
  mse_train_1    <- mean((train_scaled$motor_UPDRS - y_train_pred_1)^2)
  mse_test_1     <- mean((test_scaled$motor_UPDRS  - y_test_pred_1)^2)

  mse_train_100  <- mean((train_scaled$motor_UPDRS - y_train_pred_100)^2)
  mse_test_100   <- mean((test_scaled$motor_UPDRS  - y_test_pred_100)^2)

  mse_train_1000 <- mean((train_scaled$motor_UPDRS - y_train_pred_1000)^2)
  mse_test_1000  <- mean((test_scaled$motor_UPDRS  - y_test_pred_1000)^2)

  # Compute degrees of freedom for each lambda
  df_1    <- DF(X_train, lambda = 1)
  df_100  <- DF(X_train, lambda = 100)
  df_1000 <- DF(X_train, lambda = 1000)
```

## A.3 Full R code for assignment 3

```
# Read csv file and add column names
dataframe = read.csv("pima-indians-diabetes.csv", header = FALSE)
```

```r
colnames(dataframe) <- c("times_pregnant", "glucose_2h", "blood_pressure",
                   "skinfold_thickness", "serum_insulin", "mass_index",
                   "pedigree", "age", "diabetes")



# ---------------- Task 1: Scatterplot ----------------
# Plot of Age and Plasma Glucose, colored by diabetes status
plot(x = dataframe$age, y = dataframe$glucose_2h,
     xlab = "Age",
     ylab = "Plasma Glucose Concentration",
     xlim = c(min(dataframe$age), max(dataframe$age)),
     ylim = c(min(dataframe$glucose_2h), max(dataframe$glucose_2h)),
     col = ifelse(dataframe$diabetes == 0, "dodgerblue3", "coral1"),
     pch = 16,  # filled circles
     cex = 0.7,  # size of dots
     main = "Plasma Glucose vs Age")

legend("bottomright", legend = c("Diabetes", "Not diabetes"),
       col = c("coral1", "dodgerblue3"), pch = 16, cex = 0.7)



# ---------------- Task 2: Logistic regression model ----------------
# Fit a logistic regression model
model = glm(diabetes ~ glucose_2h + age,
            data = dataframe, family = "binomial")
summary(model)

# Predict probabilities from the model and classify using threshold r = 0.5
probability = predict(model, type = "response")
prediction = ifelse(probability > 0.5, 1, 0)

# Confusion matrix
confusion_matrix <- table(dataframe$diabetes, prediction)
print(confusion_matrix)

# Misclassification rate: Proportion of incorrectly classified observations
incorrect <- sum(confusion_matrix) - sum(diag(confusion_matrix))
misclassification <- incorrect/sum(confusion_matrix)
print(paste("Misclassification rate = ", misclassification))

# Scatterplot: Classifications from model
plot(x = dataframe$age, y = dataframe$glucose_2h,
     xlab = "Age",
     ylab = "Plasma Glucose Concentration",
     xlim = c(min(dataframe$age), max(dataframe$age)),
     ylim = c(min(dataframe$glucose_2h), max(dataframe$glucose_2h)),
     col = ifelse(prediction == 0, "dodgerblue3", "coral1"),
     pch = 16,  # filled circles
     cex = 0.7,  # size of dots
     main = "Plasma Glucose vs Age")

legend("bottomright", legend = c("Predicted diabetes", "Predicted not diabetes
    "),
       col = c("coral1", "dodgerblue3"), pch = 16, cex = 0.7)



# ---------------- Task 3: Decision boundary ----------------
# Add decision boundary line from logistic regression coefficients
abline(a = -coef(model)[1]/coef(model)[2],  # intercept
       b = -coef(model)[3]/coef(model)[2],  # slope
       col = "black", lwd = 1.5)


# ---------------- Task 4: Scatterplots with basis thresholds ----------------
# Predict using threshold r = 0.2
prediction2 = ifelse(probability > 0.2, 1, 0)

# Confusion matrix
confusion_matrix2 <- table(dataframe$diabetes, prediction2)
print(confusion_matrix2)

# Misclassification rate
incorrect2 <- sum(confusion_matrix2) - sum(diag(confusion_matrix2))
misclassification2 <- incorrect2/sum(confusion_matrix2)
print(paste("Misclassification rate 2 = ", misclassification2))

# Scatterplot
plot(x = dataframe$age, y = dataframe$glucose_2h,
     xlab = "Age",
     ylab = "Plasma Glucose Concentration",
     xlim = c(min(dataframe$age), max(dataframe$age)),
     ylim = c(min(dataframe$glucose_2h), max(dataframe$glucose_2h)),
     col = ifelse(prediction2 == 0, "dodgerblue3", "coral1"),
     pch = 16,  # filled circles
     cex = 0.7,  # size of dots
     main = "Plasma Glucose vs Age"
)

legend("bottomright", legend = c("Predicted diabetes", "Predicted not diabetes
    "),
       col = c("coral1", "dodgerblue3"), pch = 16, cex = 0.7)



# Predict using threshold r = 0.8
prediction3 = ifelse(probability > 0.8, 1, 0)
```

```r
# Confusion matrix
confusion_matrix3 <- table(dataframe$diabetes, prediction3)
print(confusion_matrix3)

# Misclassification rate
incorrect3 <- sum(confusion_matrix3) - sum(diag(confusion_matrix3))
misclassification3 <- incorrect3/sum(confusion_matrix3)
print(paste("Misclassification rate 3 = ", misclassification3))

# Scatterplot
plot(x = dataframe$age, y = dataframe$glucose_2h,
     xlab = "Age",
     ylab = "Plasma Glucose Concentration",
     xlim = c(min(dataframe$age), max(dataframe$age)),
     ylim = c(min(dataframe$glucose_2h), max(dataframe$glucose_2h)),
     col = ifelse(prediction3 == 0, "dodgerblue3", "coral1"),
     pch = 16,  # filled circles
     cex = 0.7,  # size of dots
     main = "Plasma Glucose vs Age"
)

legend("bottomright", legend = c("Predicted diabetes", "Predicted not diabetes
    "),
       col = c("coral1", "dodgerblue3"), pch = 16, cex = 0.7)



# ---------------- Task 5: Basis function expansion ----------------
# Add basis features to dataset
dataframe$z1 = dataframe$glucose_2h^4
dataframe$z2 = dataframe$glucose_2h^3 * dataframe$age
dataframe$z3 = dataframe$glucose_2h^2 * dataframe$age^2
dataframe$z4 = dataframe$glucose_2h^4 * dataframe$age^3
dataframe$z5 = dataframe$age^4

# Fit logistic regression with basis-expanded features
model2 = glm(diabetes ~ glucose_2h + age + z1 + z2 + z3 + z4 + z5,
             data = dataframe, family = "binomial")
summary(model2)

# Calculate all probabilities for each person (with their x1, x2)
probability_basis = predict(model2, type = "response")
prediction_basis = ifelse(probability_basis > 0.5, 1, 0)

# Confusion matrix
confusion_matrix_basis <- table(dataframe$diabetes, prediction_basis)
print(confusion_matrix_basis)

# Misclassification rate
```

```
incorrect_basis <- sum(confusion_matrix_basis) - sum(diag(confusion_matrix_
    basis))
misclassification_basis <- incorrect_basis/sum(confusion_matrix_basis)
print(paste("Misclassification rate = ", misclassification_basis))

# Scatterplot
plot(x = dataframe$age, y = dataframe$glucose_2h,
     xlab = "Age",
     ylab = "Plasma Glucose Concentration",
     xlim = c(min(dataframe$age), max(dataframe$age)),
     ylim = c(min(dataframe$glucose_2h), max(dataframe$glucose_2h)),
     col = ifelse(prediction_basis == 0, "dodgerblue3", "coral1"),
     pch = 16,  # filled circles
     cex = 0.7,  # size of dots
     main = "Plasma Glucose vs Age"
)

legend("bottomright", legend = c("Predicted diabetes", "Predicted not diabetes
    "),
       col = c("coral1", "dodgerblue3"), pch = 16, cex = 0.7)
```