Machine Learning: Assignment 2

Karsten Standnes - STNKAR012

October 2017

1 Introduction

In this assignment I'll be looking at the performance of different models when fitting a data created by a unknown mode. In "Task 1" two linear models with an offset is fitted to dataset and the relative performance is measured. "Task 2" looks at making legendre based model and fit them to a dataset made from a sinus function. Different values for λ, the regularization parameter is used. In "Task 3" a set of facial images are processed and used to produce eigenfaces. This task evolves around eigenfaces and how they can be used to reconstruct a picture.

Contents

2 Task 1

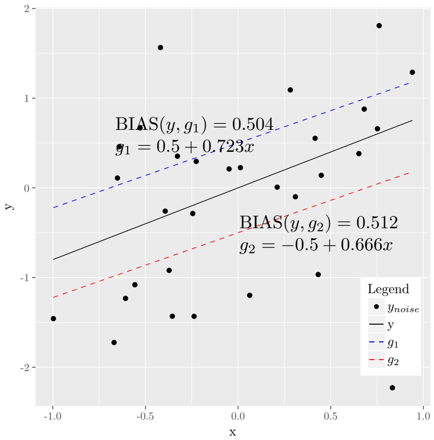
In this task we look at the linear function of the form:

$$\begin{aligned} &y_i = 0.8x_i + \epsilon_i; \quad -1 \leq x \leq 1, i = 1, ..., N \\ &\text{where: } \epsilon_i \sim \text{Normal}(0,1) \end{aligned} \tag{1}$$

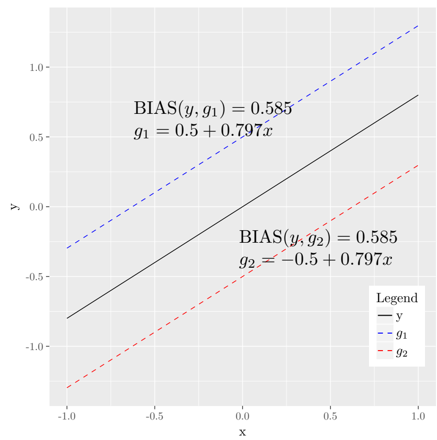
2.1 Task 1 i

First a dataset of size $N = 30$ is created from using equation 1 using a set of x values generated from the Uniform$(-1, 1)$. Then two models are attempted fit two the dataset. The models fitted are:

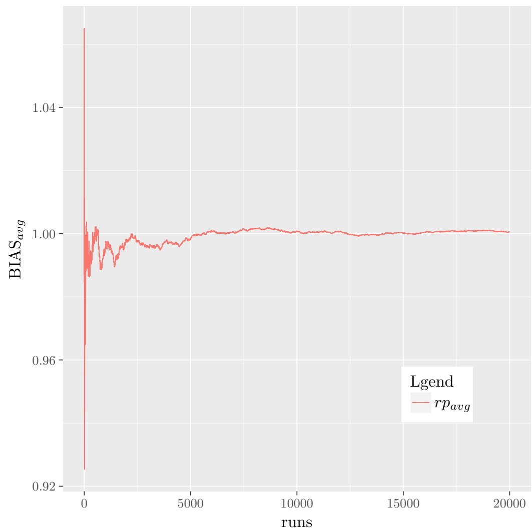
$$\begin{aligned} g_1(x) &= 0.5 + b_1 x \\ g_2(x) &= -0.5 + b_2 x \end{aligned}$$



(a) g_1 and g_2 fitted to the noisy data.



(b) Average fitted g_1 and g_2 over 20000 data sets.



(c) Relative bias $\frac{\text{BIAS}(y,g_1)}{\text{BIAS}(y,g_1)}$ for 1 to 20000 runs.

Figure 1: .

In figure 1a we see the underlining function together with a dataset created from equation 1 and the linear functions $g_1(x)$ and $g_2(x)$ fitted to the data. Both functions in this case performs fairly well with pretty similar slope, with $g_1(x)$ performing slightly better. This being the result from only one dataset means that it can't be viewed as representative for the performance of the functions. To get a better picture, 20000 dataset are created and averaged over. Figure 1b shows $g_1(x)$ and $g_2(x)$ after 20000 runs. We can see that the bias and slope is the same for both giving a relative performance (rp) equal to 1. In figure 1c below the relative performance is plotted as a function of average relative bias after 1 to 20000 runs. After around 5000 runs the relative bias starts to converge towards 1, meaning the models perform equally well. The functions have a constant ± 0.5 making it impossible to perfectly fit the underlying function. Intuitively the best models will have the slope with a offset of ± 0.5. This is confirmed when the slope of the functions converge to $b_1, b_2 \approx 0.8$, the slope of the original underlying function.

2.2 Task 1 ii

By using the model in equation 1, 10000 datasets of size $N = 30$ are simulated. Each is set is split into 21 combinations of validation and training sets, where the validation set is of the size i and the training set of the size $30 - i$, with $i \in [5, 25]$. For each of the set the models $g_1(x)$ and $g_2(x)$ are attempted fit to the training data and validated on the validation set. For each dataset and i the best of the two models is selected. For each model selected the error measures E_{out} and E_{val} is calculated and the average over all the datasets is taken to give the errors for each combination of training and validation set. Below the averaged best for each i is plotted.

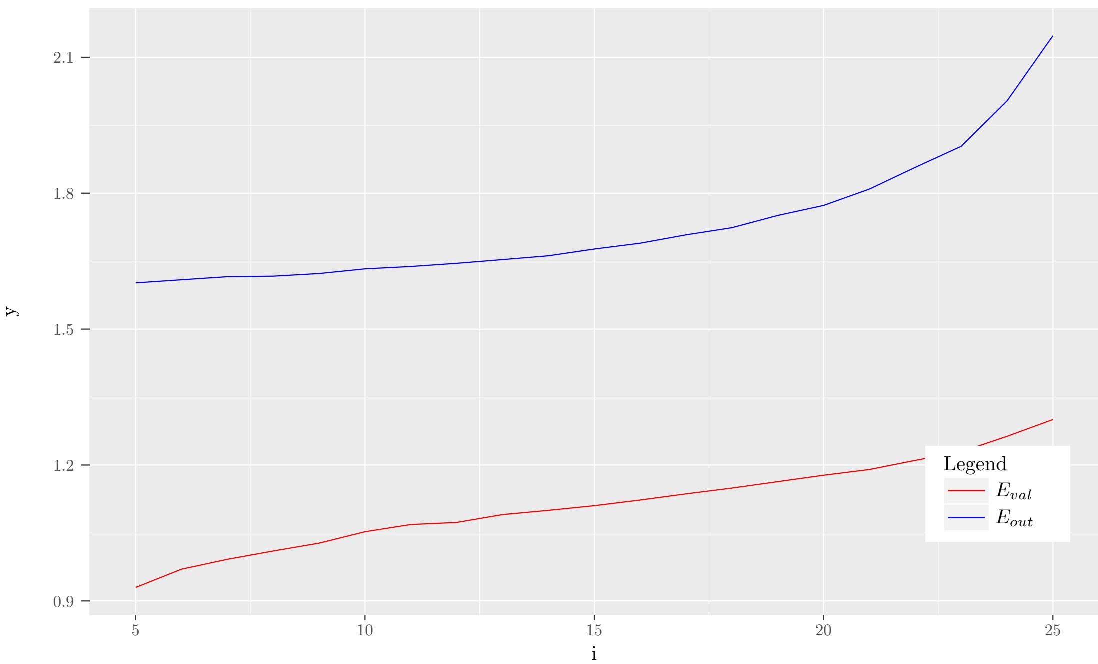


Figure 2: Error measures E_{out} and E_{val} are plotted for different sizes of training and validation sets.

In figure 2 it is a clear difference between the value of E_{val}(use MSE) and E_{out}(use BIAS) which is pretty stable for different sizes of the training and validation set. We can see that both the errors for increases for increasing i. E_{out} starts with incrementing slowly, then more rapidly around $i \sim 15 - 20$, while E_{val} has a more steady increase with a faster increase in the start $(i < 10)$. We can deduce from this that E_{val} will consequently give a lower value than E_{out} and that decreasing the training set in advantage for increasing the validation set give a worse performance, at least for datasets of size $N = 30$. This makes sense since the less data the models have available during training, the more likely it is to fit the noise of the data. Then increasing the validation set does not help other than giving a more precise measure of the performance. That being said, having a large enough validation set is also important to keep up the precision of the measure of error. Here the conclusions of it being a disadvantage to decrease the training set can be drawn using the average of 10000 sets of data.

3 Task 2

In "Task 2" we look at the function of the form:

$$\begin{aligned} &y_i = \sin(\pi x_i) + \epsilon_i; \quad -1 \leq x \leq 1, i = 1, ..., N \\ &\text{where: } \epsilon_i \sim \text{Normal}(0, 1) \end{aligned} \tag{2}$$

For each of the subtasks a dataset/datasets of size $N = 50$ with $x \sim \text{Uniform}(-1, 1)$.

3.1 Task 2 i

Simulate a dataset using equation **??** and plot it together with the underlying model:

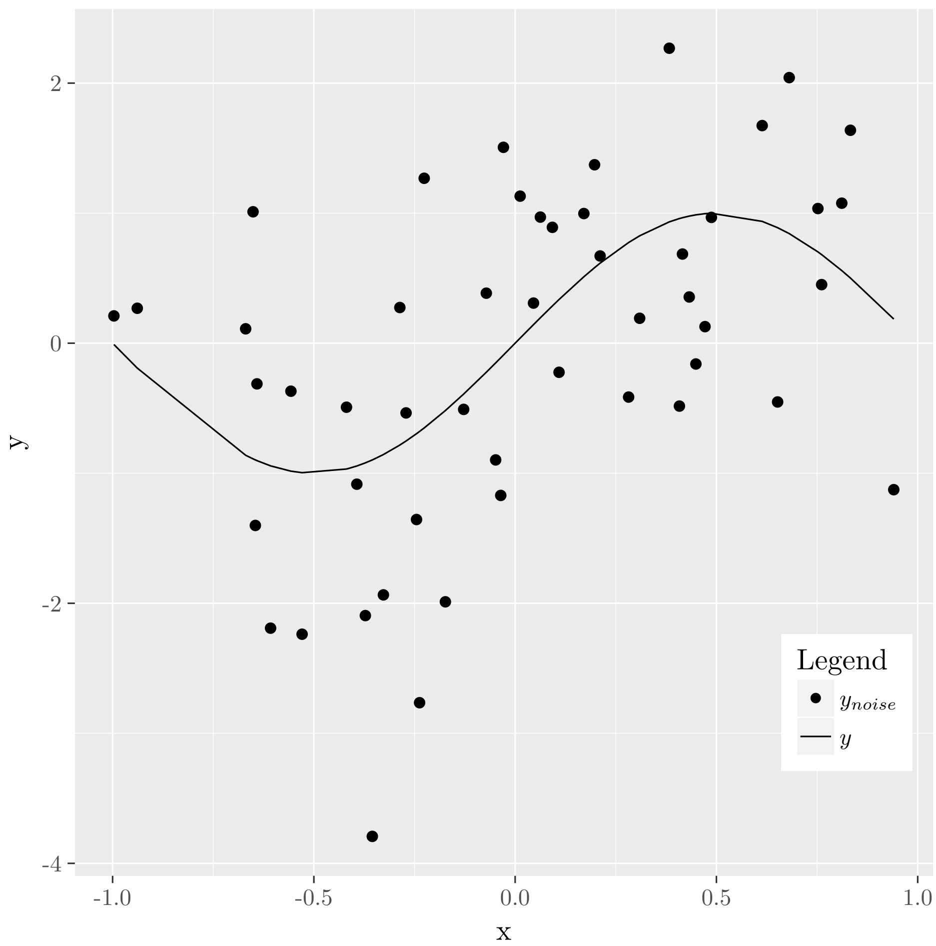


Figure 3: Underlying function with simulated dataset using equation 2

3.2 Task 2 ii

In this subtask two models are fitted to the noisy data, the models are based on

$$y_i = \sum_{q=0}^{Q_f=10} \beta_q L_q(x) \tag{3}$$

,$L_q(x)$ being the Legendre polynomial of q-th order. The Legendre polynomial is given by:

$$L(x) = 2^q \sum_{k=0}^{q} x^k \binom{q}{k} \binom{\frac{q+k-1}{2}}{q} \tag{4}$$

. The two models have a regularization parameter λ equal to 0 and 5 for the models. Using linearization to fit the models to the noisy data:

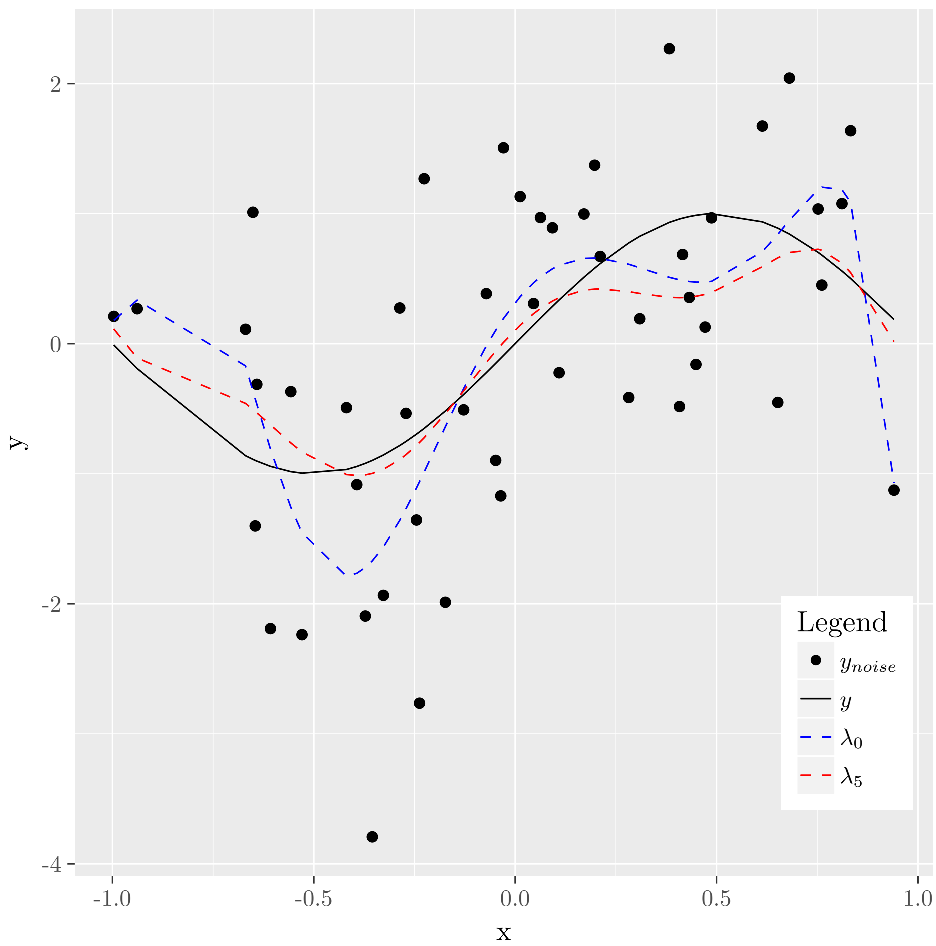
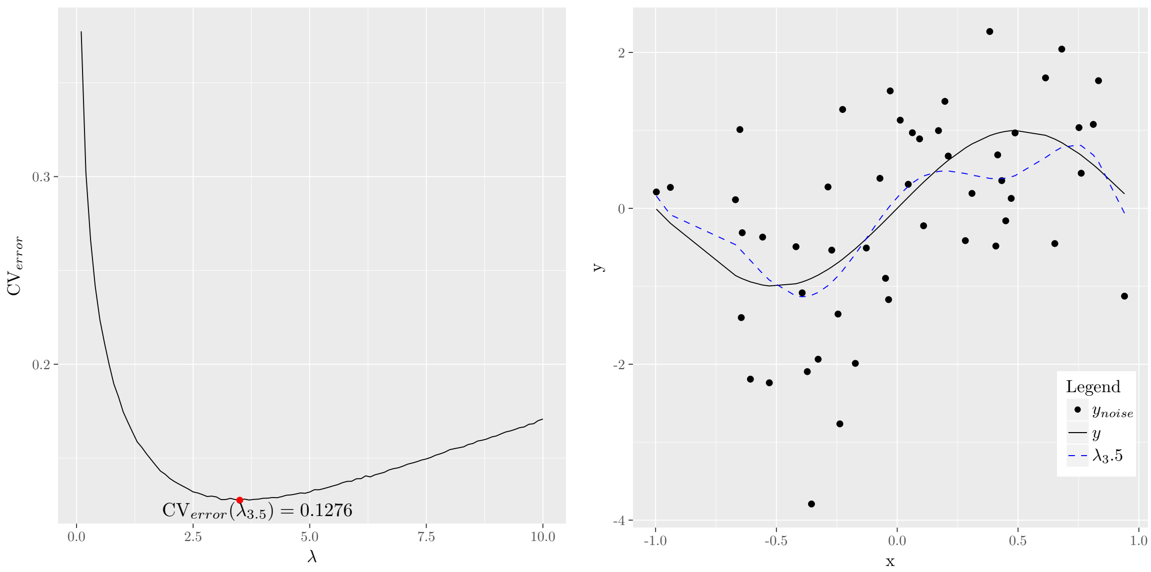


Figure 4: Two legendre models are fitted to the dataset with different values for λ.

From the figure 4 we can see that the model with a higher regularization parameter give a result that is much more similar to the underlying function. The model with $\lambda = 0$ has much more rapid change in direction, leading to it quickly missing the underlying function. This indicate that regularization increase the precision of the model compared to no regularization, at least for regularization parameter $\lambda = 5$.

3.3 Task 2 iii

To try to find the best regularization parameter for the legendre based model we can use cross-validation. In this task 10-fold cross-validation is used, meaning the dataset is split into 10 folds where each fold is used as a validation set with the other nine as the training set. This is done for all folds, then the mean-squared-error is calculated for the results of all the folds. Below are two plots, figure 5a giving the best λ based on the CV-error for 1000 dataset and figure 5b showing a model fitted with the best found λ on the data from figure 4.



(a) λ_i giving the lowest CV-error over 1000 datasets

(b) Model fitted with best $\lambda_i = 3.5$.

Figure 5: Task 2 iii

From the results above we can clearly see that the CV-error drastically decrease in the start when increasing the value of λ, then it has stabilize while curving slightly around $\lambda \in [2.5, 5.0]$ before it gradually increases in a seemingly linear fashion. When plotting the function with the obtained best regularization parameter $\lambda = 3.5$, it performs quite similar as the model with $\lambda = 5$ in figure 4. Looking at the CV-error plot this seems very reasonable, giving that both values give a very similar error. It's hard to say, but it might also seem like the model with $\lambda = 5$ performs better on this dataset. This does not destroy the result that $\lambda = 3.5$ is the best in figure 5a since an *average best* does not guarantee a *always best*.

4 Task 3

Using a image set of 400 pictures of 40 people with 10 of each person, we look at eigenfaces and reconstructing faces from them using principal component analysis PCA.

4.1 Task 3 i

(a) Mean image/face

(b) Standard deviation image/face

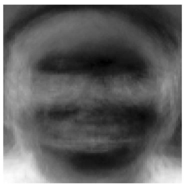
(c) Original image/face

(d) Scaled image/face

Figure 6: Task 2 i - SD and mean image/face of the set together with tha scaled and original image/face of i" *168.pgm*

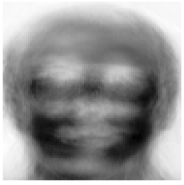
4.2 Task 3 ii

Deriving the first ten eigenfaces by using the whole set of images:



(a)

(b)



(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

Figure 7: Ten first eigenfaces based on image set.

4.3 Task 3 iii

In this subtask we will look at how it is possible to reconstruct a image using eigenfaces. Below the image "*115.pgm*" together with reconstruted images based on 5, 50 and 200 eigenfaces.

(a) Scaled image

(b) Reconstructed with 5 eigenfaces

(c) Reconstructed with 50 eigenfaces

(d) Reconstructed with 200

Figure 8: Scaled version of "*115.pgm*" together with reconstructed versions.

The purpose of using eigenfaces is to efficiently store and encode images. Eigenfaces are derived eigenvalues of the co-variance matrix of the set of images. By using eigenfaces it is possible to reduce the dimensionality by using a smaler set of images to represent the orginal set. The eigenvalues stores the variation in the set of face images and can be combined to reconstruct a specific image. As seen in figure 8 that as the number of eigenfaces increase the face shown starts to look more and more similar to the scaled version of the real face. Another interesting observation is that as the number of eigenfaces used increase each image has a lesser impact in improving the quality. The difference in increasing from 5 to 50 eigenfaces is much more dramatic than from 50 to 200 eigenfaces. This indicate that to get a recognizable image a small amount of eigenfaces is needed and increasing the number after that will only improve the detail

5 Appendix

5.1 R-Code: Question 1

```r
## Libraries and seed
library(ggplot2) #Ggplot library used for plotting
library(tcltk) #Used for loading bar
library(tikzDevice) #Using tikz to store ggplot as tex
set.seed(420)

## Help functions

# calculated function from "quote" expression
make_data_set <- function(
    expression,
    x
    )
    {
        y_dataset <- sapply(x, function(x) eval(expression))
    }

# fit linear model to data with offset
fit_function <- function(
    b_intercept,
    x,
    y,
    N
    )
    {
        g_i <- lm(y ~ 0+x, offset = rep(b_intercept,N))
        slope <- unname(coef(g_i))
    }

## Main functions

# Task 1, fit two linear functions to y with noise
# if i >= 1, performs the average of the runs
# set i = 1 to get only one run
# plot the two estimated functions with the orginal y - function
# also plot relative performance for between the two models
# when averaging many runs, recommended i > 100 for average run
task1i <- function(i)
    {
        N = 30

        sigma = 1
        b_y <- 0.8

        bias_g_1 <- 0
        bias_g_2 <- 0

        b_g_1_avg <- 0
        b_g_2_avg <- 0

        avg_relative_performance <- rep(0, i)

        y_dataset_avg <- rep(0, N)

        y_wthout_noise <- quote(0.8 * x)
```

```
        y_wth_noise <- quote(0.8 * x + rnorm(1,0,1))

        for(k in 1:i){

            x <- runif(n = N, min = -1, max = 1)

            y_dataset <- make_data_set(y_wth_noise, x)
            y_dataset_avg <- y_dataset_avg + y_dataset

            b_g_1 <- fit_function(0.5, x, y_dataset, N)
            b_g_2 <- fit_function(-0.5, x, y_dataset, N)

            b_g_1_avg <- b_g_1_avg + b_g_1
            b_g_2_avg <- b_g_2_avg + b_g_2

            bias_g_1 <- bias_g_1 + integrate(function(x) (0.5 + (b_g_1 - b_y) * x)^2, -1, 1)$value
            bias_g_2 <- bias_g_2 + integrate(function(x) (-0.5 + (b_g_2 - b_y) * x)^2, -1, 1)$value

            avg_relative_performance[k] <- bias_g_1 / bias_g_2
        }

        # Task average of all values
        bias_g_1 <- bias_g_1 / i
        bias_g_2 <- bias_g_2 / i
        b_g_1_avg <- b_g_1_avg / i
        b_g_2_avg <- b_g_2_avg / i
        y_dataset <- y_dataset_avg / i

        if(i > 1){
            x = seq(-1,1, length.out = N)
        }

        y_values <- eval(y_wthout_noise)

        g_1 <- quote(0.5 + b_g_1_avg * x)
        g_2 <- quote(-0.5 + b_g_1_avg * x)

        g_1_values <- eval(g_1)
        g_2_values <- eval(g_2)

        print(bias_g_1)
        print(bias_g_2)

        ggplot_df <- data.frame(x, y_dataset, y_values, g_1_values, g_2_values)
        avg_relative_performance_df <- data.frame(x = 1:i, avg_relative_performance)

        # plot fitted function together with orginal function y
        tikz(file = paste0("Pictures/Task1/task1ibias", i, ".tex"), width = 5, height = 5)
        ggplot1 <- ggplot(data = ggplot_df, aes(x = x)) +
                   geom_line(aes(y = y_values, colour = "y")) +
                   geom_line(aes(y = g_1_values, colour = "$g_1$"), linetype = "dashed") +
                   geom_line(aes(y = g_2_values, colour = "$g_2$"), linetype = "dashed") +
```

```
                   xlab("x") +
                   ylab("y") +
                   annotate("text", x = -0.1, y = 0.7,
                            label = paste0("$\\mathrm{BIAS}(y,g_1) = ",
                                            round(bias_g_1,3),
                                            "$ \\\\",
                                            "$g_1 = 0.5 + ",
                                            round(b_g_1_avg,3),
                                            " x$")) +
                   annotate("text", x = 0.5, y = -0.4,
                            label = paste0("$\\mathrm{BIAS}(y,g_2) = ",
                            round(bias_g_2,3),
                            "$ \\\\",
                            "$g_2 = -0.5 + ",
                            round(b_g_2_avg,3),
                            " x$"))
        #Only plot datapoints when running one dataset
        if(i == 1){
            ggplot1 <- ggplot1 +
            geom_point(aes(y = y_dataset, colour = "$y_{noise}$")) +
            scale_colour_manual("Legend",
                              breaks = c("$y_{noise}$", "y", "$g_1$", "$g_2$"),
                              values = c("blue", "red", "black", "black"),
                              guide = guide_legend(override.aes = list(
                                  linetype = c("blank", "solid", "dashed", "dashed"),
                                  shape = c(16, NA, NA, NA)
                                  ))) +
            theme(legend.position = c(0.9, 0.2))
        }else{
            ggplot1 <- ggplot1 +
            scale_colour_manual("Legend",
                              breaks = c( "y", "$g_1$", "$g_2$"),
                              values = c("blue", "red", "black"),
                              guide = guide_legend(override.aes = list(
                                  linetype = c( "solid", "dashed", "dashed"),
                                  shape = c( NA, NA, NA)
                                  ))) +
            theme(legend.position = c(0.9, 0.2))
        }

        ggsave(paste0("Pictures/Task1/task1i", i, ".png"))
        print(ggplot1) #need to print plot to write to .tex file
        dev.off()

        # plot average relative performance rp and shows value after averaging 1 to i times
        tikz(file = paste0("Pictures/Task1/task1ibias_avg", i, ".tex"), width = 5, height = 5)
        ggplot2 <- ggplot(data = avg_relative_performance_df, aes(x = x)) +
                   geom_line(aes(y = avg_relative_performance, colour = "$rp_{avg}$")) +
                   xlab("runs") +
                   ylab("$\\mathrm{BIAS}_{avg}$") +
                   theme(legend.position = c(0.8, 0.2))
        ggsave(paste0("Pictures/Task1/task1i_avg", i, ".png"))
        print(ggplot2)
```

```
        dev.off()
    }

# finding the val. and out. error for different sizes of training and validation
# set out of a set with total size N = 30
task1ii <- function(
    n_training_sets = 10000
    )
    {
        N = 30
        indexes <- 1:N

        # progress bar
        pb <- tkProgressBar(title = paste0("Running ", n_training_sets, " data sets"),
                            min = 0, max = n_training_sets, width = 300)

        length_df <- N - 2*5 +1
        error_df <- data.frame(g_1_error_val = rep(0,length_df),
                               g_2_error_val = rep(0,length_df),
                               g_1_error_out = rep(0,length_df),
                               g_2_error_out = rep(0,length_df),
                               collection_val = rep(0,length_df),
                               collection_out = rep(0,length_df))

        for(k in 1:n_training_sets){
            setTkProgressBar(pb, k, label = paste(round(k/n_training_sets*100, 0), "% done"))
            x <- runif(n = N, min = -1, max = 1)
            b_y <- 0.8
            y_wth_noise <- quote(0.8 * x + rnorm(1,0,1))
            y_dataset <- make_data_set(y_wth_noise, x)
            for(i in 5:(N-5)){
                train_indexes <- sample(indexes, N - i)
                test_indexes <- subset(indexes, !(indexes %in% train_indexes))

                x_train <- x[train_indexes]
                x_test <- x[test_indexes]

                y_train <- y_dataset[train_indexes]
                y_test <- y_dataset[test_indexes]

                b_g_1 <- fit_function(0.5, x_train, y_train, N - i)
                b_g_2 <- fit_function(-0.5, x_train, y_train, N - i)

                g_1 <- quote(0.5 + b_g_1 * x_test) # TO DO: pr ve utf re fit_function og f   ut verde f r settes i quote
                g_2 <- quote(-0.5 + b_g_2 * x_test)

                g_1_values <- eval(g_1)
                g_2_values <- eval(g_2)

                g_1_mse <- round(mean((y_test - g_1_values)^2),5)
                g_2_mse <- round(mean((y_test - g_2_values)^2),5)

                g_1_bias <- integrate(function(x) (0.5 + (b_g_1 - b_y) * x)^2, -1, 1)$value + 1
```

```
                g_2_bias <- integrate(function(x) (-0.5 + (b_g_2 - b_y) * x)^2, -1, 1)$value +1

                error_df$g_1_error_val[i-4] <- (error_df$g_1_error_val[i-4] + g_1_mse)
                error_df$g_2_error_val[i-4] <- (error_df$g_2_error_val[i-4] + g_2_mse)

                error_df$g_1_error_out[i-4] <- (error_df$g_1_error_out[i-4] + g_1_bias)
                error_df$g_2_error_out[i-4] <- (error_df$g_2_error_out[i-4] + g_2_bias)

                error_df$collection_val[i-4] <- (error_df$collection_val[i-4] + min(g_1_mse, g_2_mse))
                error_df$collection_out[i-4] <- (error_df$collection_out[i-4] + min(g_2_bias, g_2_bias))
            }
        }
        close(pb)
        error_df$test_set_amount <- 5:(N-5)

        error_df[-7] <- error_df[-7] / n_training_sets

        # Individual errors for g_1 and g_2
        ggplot1 <- ggplot(data = error_df, aes(x = test_set_amount)) +
                   geom_line(aes(y = g_1_error_val, colour = "g_1")) +
                   geom_line(aes(y = g_2_error_val, colour = "g_2")) +
                   geom_line(aes(y = g_1_error_out, colour = "g_1_2")) +
                   geom_line(aes(y = g_2_error_out, colour = "g_2_2"))
        ggsave("Pictures/Task2/task2ii1.png")

        # Best error chosen at each i
        tikz(file = paste0("Pictures/Task1/task1ii.tex"), width = 5, height = 5)
        ggplot2 <- ggplot(data = error_df, aes(x = test_set_amount)) +
                   geom_line(aes(y = collection_val, colour = "$E_{val}$")) +
                   geom_line(aes(y = collection_out, colour = "$E_{out}$")) +
                   xlab("i") +
                   ylab("y") +
                   scale_colour_manual("Legend",
                                    breaks = c("$E_{val}$", "$E_{out}$"),
                                    values = c("blue", "red")) +
                   theme(legend.position = c(0.9, 0.2))
        ggsave("Pictures/Task2/task2ii2.png")
        print(ggplot2)
        dev.off()
    }

## Run

main <- function()
    {
        #for(i in 1:10){
        #    task1i(i)
        #}
        #task1i(20000)
```

```
        task1i(1)
        task1ii()
    }

main()

## Plotting against the machine
```

code/Task1.R

5.2 R-Code: Question 2

```
## Libraries and seed
library(ggplot2)
library(tcltk)
library(tikzDevice)
set.seed(420)
## Help functions

options(tikzMetricPackages = c("\\usepackage[utf8]{inputenc}","\\usepackage[
    T1]{fontenc}", "\\usetikzlibrary{calc}", "\\usepackage{amssymb}"))

# Take a expression and compute the values based on input z
make_data_set <- function(
    expression,
    x
    )
    {
        y_dataset <- sapply(x, function(x) eval(expression))
        return(y_dataset)
    }

# legendre expression within sum formula
l <- function(
    x,q,k
    ){
      legendre <- x**k * choose(q,k) * choose((q+k-1)/2,q)
      return(legendre)
    }

# produces legendre polynomial
legendres <- function(
    x,q
    )
    {
    func <- rep(0,length(x))
        for(k in 0:q){
            func <- func + l(x,q,k)
        }
        #print(func)
    legendre_polynomial <- 2**q * func
    return(legendre_polynomial)
    }

# use regularization paramter lambda to train weights use in the model
regularEstimation <- function(
    data,
    lambda,
    Q_order
    )
    {
        dimentions <- dim(data)
        y = matrix(data[,2], nrow = dimentions[1])
        Z = matrix(rep(0,(dimentions[1] * (Q_order + 1))), nrow = dimentions
            [1])
        for(i in 1:dimentions[1]){
            Z[i,1] <- 1
            for(j in 2:(Q_order+1)){
                Z[i,j] = legendres(data[i, "x"], j-1)
```

```
            }
        }
        weights = solve((t(Z)%*%Z) + (lambda * diag(Q_order + 1)))%*%(t(Z)%*
            %y)
        return(weights)
    }
# use weights from regularEstimation to regularize the legendre based model
regularLinearization <- function(
    x,
    data,
    lambda,
    Q_order
    ){
        legendreMatrix <- regularEstimation(data, lambda, Q_order)
        pol <- legendreMatrix[1]
        for(i in 2:(Q_order +1)){
            pol <- pol + (legendreMatrix[i] * (legendres(x, i-1)))
        }
        return(pol)
    }

# creates n_folds of N indexes from 1 to N
create_cv_idexes <- function(N, n_folds){
    indexes_per_fold <- ceiling(N/n_folds)
    index_matrix <- matrix(0L, nrow = n_folds, ncol = indexes_per_fold)
    index_available <- 1:50
    for(i in 1:n_folds){
        selected_indexes <- sample(index_available, indexes_per_fold)
        index_available <- index_available[! index_available %in% selected_
            indexes]

        index_matrix[i, ] <- selected_indexes
    }
    return(index_matrix)
}

# calculate the cv error
cv_error <- function(
    N,
    lambda,
    data,
    n_folds,
    cv_indexes,
    Q_order
    )
    {
        indexes <- 1:N
        cv_indexes <- create_cv_idexes(N, n_folds)
        cv_error <- c()
        for(i in 1:n_folds){
            test_indexes <- cv_indexes[i, ]
            train_indexes <- subset(indexes, !(indexes %in% test_indexes))

            x_train <- data$x[train_indexes]
            x_test <- data$x[test_indexes]

            y_train <- data$y[train_indexes]
            y_test <- data$y_real[test_indexes]

```

```
            temp_data <- data.frame(x = x_train, y =y_train)

            lambdaX <- regularLinearization(x_test, temp_data, lambda, 10)

            cv_error <- c(cv_error, ((y_test - lambdaX)^2))
        }
        cv_error <- mean(cv_error)
    }

## Main functions

# plotting underlying function with data set
task2i <- function()
    {
        N = 50
        x = runif(n = N, min = -1, max = 1)

        y <- quote(sin(pi*x))
        y_wth_noise <- quote(sin(pi*x) + rnorm(1,0,1))

        y_dataset <- make_data_set(y_wth_noise, x)

        ggplot_df <- data.frame(x, y_dataset)

        tikz(file = "Pictures/Task2/task2i.tex", width = 5, height = 5)
        ggplot1 <- ggplot(data = ggplot_df, aes(x = x)) +
                   geom_point(aes(y = y_dataset, colour = "$y_{noise}$")) +
                   geom_line(aes(y = sin(pi*x), colour = "$y$")) +
                   xlab("x") +
                   ylab("y") +
                   scale_colour_manual("Legend",
                                     breaks = c( "$y_{noise}$", "$y$"),
                                     values = c("black", "black"),
                                     guide = guide_legend(override.aes =
                                         list(
                                         linetype = c( "blank", "solid"),
                                         shape = c( 16, NA)
                                         ))) +
                   theme(legend.position = c(0.9, 0.2))
        ggsave("Pictures/Task2/task2i.png")
        print(ggplot1)
        dev.off()
    }

# Fit two legendre polynomial with different regularization paramters
task2ii <- function(
    )
    {
        N = 50
        sigma = 1
        x <- runif(n = N, min = -1, max = 1)
        y <- sin(pi*x) + rnorm(N,0,sigma^2)

        data <- data.frame(x,y)

        lambda0 <- regularLinearization(x, data, 0, 10)
        lambda5 <- regularLinearization(x, data, 5, 10)

        ggplot_df <- data.frame(x,y,lambda0,lambda5)
```

```r

        tikz(file = "Pictures/Task2/task2ii.tex", width = 5, height = 5)
        ggplot1 <- ggplot(data = ggplot_df, aes(x = x)) +
                    geom_point(aes(y = y, colour = "$y_{noise}$")) +
                    geom_line(aes(y = sin(pi*x), colour = "$y$")) +
                    geom_line(aes(y = lambda0, colour = "$\\lambda_0$"), linetype = "dashed") +
                    geom_line(aes(y = lambda5, colour = "$\\lambda_5$"), linetype = "dashed") +
                    xlab("x") +
                    ylab("y") +
                    scale_colour_manual("Legend",
                                        breaks = c( "$y_{noise}$", "$y$", "$\\lambda_0$", "$\\lambda_5$"),
                                        values = c("blue", "red", "black", "black"),
                                        guide = guide_legend(override.aes = list(
                                            linetype = c( "blank", "solid", "dashed", "dashed"),
                                            shape = c( 16, NA, NA, NA)
                                            ))) +
                    theme(legend.position = c(0.9, 0.2))
        ggsave("Pictures/Task2/task2ii.png")
        print(ggplot1)
        dev.off()
    }

# plot result from task 2 iii when finding the best regularization parameter
plot_task2iii <- function
    (
    ggplot_df
    )
    {
        lowest_error = ggplot_df[which.min(ggplot_df[,2]),]
        str(lowest_error)
        ?tikzTest
        tikz(file = "Pictures/Task2/task2iii.tex", width = 5, height = 5)
        ggplot1 <- ggplot(data = ggplot_df, aes(x = lambdas, y = error_vector)) +
                    geom_line() +
                    geom_point(data = lowest_error,
                               aes(x = lambdas, y = error_vector),
                               color = "red") +
                     geom_text(data = lowest_error,
                               aes(label = paste0("$\\mathrm{CV}_{error}(\\lambda_{",
                                                  lambdas, "}) = ",
                                                  round(error_vector,4), "$")),
                                    hjust = 0.4, vjust = 1.3) +
                     labs(x = "$\\lambda$", y = "$\\mathrm{CV}_{error}$")
                     #     title = paste("CV error- regularisation $\\lambda$ between",
                     #                   ggplot_df[1,"lambdas"], "and",
                     #                   ggplot_df[nrow(ggplot_df), "lambdas"]))
        print(ggplot1)
        dev.off()
```

```r
        ggsave("Pictures/Task2/avg_2iii1000runs.png")
    }

# Using CV-error to fund the best regularization parameter for dataset made by
# this model
task2iii <- function(

    )
    {
        N = 50
        sigma = 1
        x <- runif(n = N, min = -1, max = 1)
        y <- sin(pi*x) + rnorm(N, 0, sigma)
        y_real <- sin(pi*x)

        data <- data.frame(x, y, y_real)

        lambda = 0.1
        lambdas <- seq(from = 0.1, 10, by = 0.1)
        n_folds = 10
        error_vector <- integer(length(lambdas))
        cv_indexes <- create_cv_idexes(N, n_folds)

        pb <- tkProgressBar(title = paste0("Running ", length(lambdas), " lambdas"),
                            min = 0, max = length(lambdas), width = 300)
        for(i in 1:length(lambdas)){
            setTkProgressBar(pb, i, label = paste(round(i/length(lambdas)*100, 0), "% done"))
            lambda = lambdas[i]
            print(lambda)
            error_vector[i] <- cv_error(N = N, lambda = lambdas[i], data = data,
                                        n_folds = 10, cv_indexes = cv_indexes,
                                        Q_order = 10)
        }
        close(pb)
        write.csv(error_vector, "error_01")
        print(error_vector)
        ggplot_df <- data.frame(lambdas, error_vector)
        write.csv(ggplot_df, "ggplot_01.csv", row.names = FALSE)
        plot_task2iii(ggplot_df)
    }

# averaging the CV-error for the different lambdas over n_runs dataset
# takes some hours to run on large amounts of dataset,
# if using this more parallelization would be a good step to improve performance
task2iii_avg_runs <- function(
    n_runs
    )
    {
        N = 50
        sigma = 1
        n_folds = 10
        lambdas <- seq(from = 0.1, 10, by = 0.1)
        error_vector <- integer(length(lambdas))
```

```
        cv_indexes <- create_cv_idexes(N, n_folds)
        error_result <- error_vector

        pb <- txtProgressBar(min = 0, max = n_runs, style = 3)
        for(r in 1:n_runs){
            setTxtProgressBar(pb, r)
            x <- runif(n = N, min = -1, max = 1)
            y <- sin(pi*x) + rnorm(N, 0, sigma)
            y_real <- sin(pi*x)

            data <- data.frame(x, y, y_real)

            for(i in 1:length(lambdas)){
                error_vector[i] <- cv_error(N = N, lambda = lambdas[i], data = data,
                                            n_folds = 10, cv_indexes = cv_indexes,
                                            Q_order = 10)
            }
            error_result <- error_result + error_vector
        }
        close(pb)
        error_result <- error_result / n_runs
        png("test_avg_2ii.png")
        plot(error_result, type = "b")
        dev.off()
        avg_2iii <- data.frame(lambdas, error_result)
        write.csv(avg_2iii, "avg_2iii.csv", row.names = FALSE)
    }

# plot the best lambda found, a simplification of the plot for ii
task2iii_plot_best_lambda <- function(
    best_lambda
    )
    {
        N = 50
        sigma = 1
        x <- runif(n = N, min = -1, max = 1)
        y <- sin(pi*x) + rnorm(N,0,sigma^2)

        data <- data.frame(x,y)

        lambda_best <- regularLinearization(x, data, best_lambda, 10)

        ggplot_df <- data.frame(x,y,lambda_best)

        tikz(file = "Pictures/Task2/task2iii_best_lambda.tex", width = 5, height = 5)
        ggplot1 <- ggplot(data = ggplot_df, aes(x = x)) +
                   geom_point(aes(y = y, colour = "$y_{noise}$")) +
                   geom_line(aes(y = sin(pi*x), colour = "$y$")) +
                   geom_line(aes(y = lambda_best,
                                 colour = paste0("$\\lambda_", best_lambda, "$")),
                                 linetype = "dashed") +
                   xlab("x") +
                   ylab("y") +
                   scale_colour_manual("Legend",
                                   breaks = c( "$y_{noise}$", "$y$",
```

```
                                                  paste0("$\\lambda_", best_lambda, "
                                                  $")),
                                              values = c("blue", "black", "black"),
                                              guide = guide_legend(override.aes =
                                                  list(
                                                   linetype = c( "blank", "solid", "
                                                       dashed"),
                                                   shape = c( 16, NA, NA)
                                                   ))) +
                        theme(legend.position = c(0.9, 0.2))
          ggsave("Pictures/Task2/task2iii_best_lambda.png")
          print(ggplot1)
          dev.off()
     }
## Run

main <- function()
     {
          task2i()
          task2ii()
          #task2iii()

          task2iii_avg_runs(1000)

          #big rung for iii, stored to save computation if replot is nessesary
          #ggplot_df <- read.csv("avg_2iii1000runs.csv")
          #colnames(ggplot_df)[2] <- "error_vector"
          #plot_task2iii(ggplot_df)
          task2iii_plot_best_lambda(3.5)
     }

main()
```

code/Task2.R

5.3 R-Code: Question 3

```
## Libraries and seed
library(pixmap)
library(reshape2)
library(ggplot2)
library(RColorBrewer)
library(gtools)
## Help functions

#Load all faces from "Faces" folder
load_faces <- function(
    pattern,
    directory
    )
    {
        temp = paste0("Faces/", list.files(path = "Faces", pattern = "*.pgm"
            ))
        #Files are read in alpha order, need to use mixedsort to get
        #right order
        temp <- mixedsort(temp)
        myfiles = lapply(temp, read.pnm)
        return(myfiles)
    }

#Get and transform the diffrent needed data
get_image_data <- function(
    faces_list,
    n_pictures,
    height,
    width
    )
    {
        image_data <<- array(0, dim = c(n_pictures, height, width))
        image_transpose <<- array(0, dim = c(n_pictures, width, height))
        image_up <<- image_transpose
        image_mean <<- matrix(0, nrow = width, ncol = height)
        image_standard_deviation <<- image_mean
        image_scale <<- image_transpose
        for(i in 1:length(faces_list)){
            image <- faces_list[[i]]@grey
            image_data[i, , ] <<- image
            image_transpose[i, , ] <<- t(image)
        }

        for(i in 1:n_pictures){
            for(j in 1:height){
                image_up[i, , (height - j + 1)] <<- image_transpose[i, , j]
            }
        }

        for(k in 1:width){
            for(l in 1: height){
                image_up_value <- image_up[, k, l]
                image_mean[k,l] <<- mean(image_up_value)
                image_standard_deviation[k,l] <<- sd(image_up_value)
            }
        }

```

```
        for(m in 1:n_pictures){
            image_scale[m, , ] <<- (image_up[m  , , ] - image_mean) / image_
                standard_deviation
        }

    }

#Plot a face to pdf in two different ways
plot_face_pdf <- function(
    gg_melt,
    shades_of_grey = 256
    )
    {
        gg <- ggplot(gg_melt, aes(x = Var1, y = Var2)) +
            geom_tile(aes(fill = value)) +
            theme_classic()
        gg2 <- ggplot(gg_melt, aes(x = Var1, y = Var2)) +
            geom_tile(aes(fill = value)) +
            #scale_fill_gradientn(colours = brewer.pal(shades_of_grey,"Greys
                ")) +
            scale_colour_brewer(palette = brewer.pal(shades_of_grey, "Greys"
                ), direction = -1)
            theme_classic()
        #print(gg)
        print(gg2)
    }

#Plot a face to the wanted folder as png
plot_face_png <-function(
    gg_melt,
    shades_of_grey = 256,
    name
    )
    {
        name <- paste0("Pictures/Task3/", name)
        gg <- ggplot(gg_melt, aes(x = Var1, y = Var2)) +
            geom_tile(aes(fill = value)) +
            scale_fill_gradientn(colours = rev(brewer.pal(shades_of_grey,"
                Greys"))) +
            theme_classic()
        #Remove all axis and legends from the plot to only view the image
        gg <- gg + theme(axis.line=element_blank(),axis.text.x=element_blank
            (),
          axis.text.y=element_blank(),axis.ticks=element_blank(),
          axis.title.x=element_blank(),
          axis.title.y=element_blank(),legend.position="none",
          panel.background=element_blank(),panel.border=element_blank(),
          panel.grid.major=element_blank(),
          panel.grid.minor=element_blank(),plot.background=element_blank())
        ggsave(name)
    }

#Convert data to input_matrix
convert_to_input_matrix <- function(
    n_faces,
    width,
    height
    )
    {
```

```
        X <<- matrix(0, nrow = n_faces, ncol = width * height)
        for(i in 1: n_faces){
            picture <- image_scale[i, 1, ]
            for(j in 2: width){
                picture <- cbind(picture, image_scale[i, j, ])
            }
            X[i, ] <<- picture
        }
        pca <<- prcomp(X)
        A <- X %*% t(X)
        store <- prcomp(A)
        image_eigen <<- t(X) %*% store$rotation
    }

#make a eigen face image
eigen_image <- function(
    face_vector,
    width,
    height
    )
    {
        eigen_face <- matrix(NA, ncol = height, nrow = width)
        for(i in 1: width){
            eigen_face[i, ] <- face_vector[(height * (i - 1) + 1):(height *
                i)]
        }
        return(eigen_face)
    }

#recreate face from principal component analysis PCA
recreate_face <- function(
    image_nr,
    n_faces
    )
    {
        recreate <- tcrossprod(pca$x[, 1:n_faces], pca$rotation[, 1:n_faces
            ])
        recreate_matrix <- t(matrix(data = rev(recreate[image_nr, ]),
                                nrow = 112, ncol = 92))
    }

## Main functions

# plot mean, sd, original and scaled version og image_nr
task3i <- function(
    image_nr
    )
    {
        gg_mean <- melt(image_mean)
        gg_standard_deviation <- melt(image_standard_deviation)

        gg_up <- melt(image_up[image_nr, , ])
        gg_scale <- melt(image_scale[image_nr, , ])

        plot_face_png(gg_mean, name = "i/mean.png")
        plot_face_png(gg_standard_deviation, name = "i/sd.png")
        plot_face_png(gg_up, name = "i/org.png")
        plot_face_png(gg_scale, name = "i/scale.png")
    }
```

```
# plot the n_eigen_faces first eigenfaces
task3ii <- function(
    n_eigen_faces,
    width,
    height
    )
    {
        for(i in 1:n_eigen_faces){
            eigen <- image_eigen[, i]
            plot_face_png(melt(eigen_image(eigen, width, height)),
                          name = paste0("ii/eigen_face", i, ".png"))
        }
    }

# Recreate face from different amount of eigenfaces.
task3iii <- function(
    image_nr
    )
    {
        rotate_matrix <- function(x) t(apply(x, 2, rev))
        eigen_n_vector <- c(1, 5, 50, 200)
        plot_face_png(melt(image_scale[image_nr, ,]),
                      name = paste0("iii/image",
                                    image_nr, ".png"))
        for(K in eigen_n_vector){
            recreate <- recreate_face(image_nr, K)
            #rotate matrix the right way, rotating 90 degrees twice
            recreate <- rotate_matrix(rotate_matrix(recreate))
            plot_face_png(melt(recreate), name = paste0("iii/image",
                                                        image_nr,
                                                        "eigen_faces",
                                                        K, ".png"))
        }
    }

## Run

main <- function()
    {
        # matrices and df
        # image_data
        # image_transpose
        # image_up
        # image_mean
        # image_standard_deviation
        # image_scale
        # X
        # pca

        n_faces = 400
        height = 112
        width = 92
        shades_of_grey = 256
        faces <- load_faces()
        get_image_data(faces, n_faces, height, width)
        convert_to_input_matrix(n_faces, width, height)

        task3i(image_nr = 168)
```

```
        task3ii(n_eigen_faces = 10, width = width, height = height)
        task3iii(image_nr = 115)

    }

main()
```

code/Task3.R