Machine Learning

Assignment 2

Model Fitting, Cross-Validation, Regularisation and Image Scaling.

Julian Albert
ALBJUL005

Department of Statistical Sciences
University of Cape Town
South Africa
September 30, 2019

Contents

Problem 1

In this Problem I consider the model

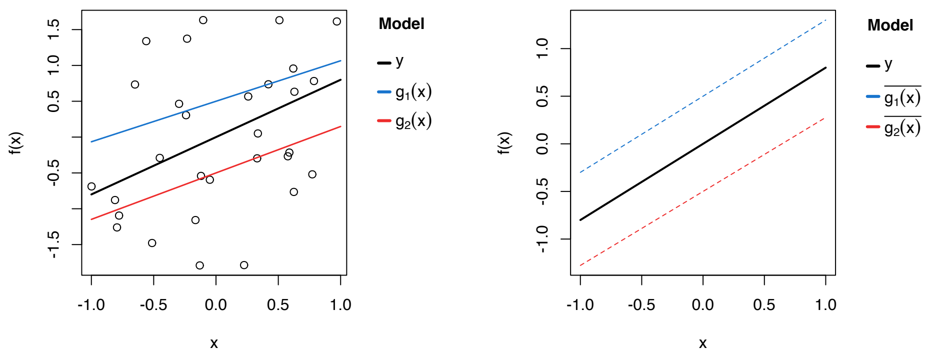
$$y_i = 0.8x_i + \epsilon_i; \quad -1 \le x \le 1, i = 1, \dots, N$$
$$\text{where: } \epsilon_i \sim \text{Normal}(0,1)$$

i) Model Fitting

Suppose $x \sim U(-1,1)$, we can simulate a dataset of size $N = 30$ and fit the models below

$$g_1(x) = \quad 0.5 + b_1 x \tag{1}$$
$$g_2(x) = -\ 0.5 + b_2 x \tag{2}$$

These models are fit using the lm() function in R, the resulting model fits are depicted in Figure (1a). We can see that Model 1 lies above the true model whilst Model 2 lies below, this is fairly obvious as the intercepts are equidistant above/below zero. From Figure (1b) we obtain the mean models over 1000 fits, we may expect that Model 1 overestimates whilst Model 2 underestimates the true model.



(a) Model Fits (b) Mean Fit of 1000 Models

Figure 1: Models $g_1(x)$ and $g_2(x)$ fit once (left) and fit 1000 times then averaged (right).

ii) Validation and Out-Of-Sample Errors

Here we will consider the effect validation size has on different errors. First we define our dataset to be $\mathcal{D}$ containing x and y which can be subdivided into a training set $\mathcal{D}_{train}$ and a validation $\mathcal{D}_{val}$. To choose the best model $g^*(x)$ we fit both models $g_1(x_{train})$ and $g_2(x_{train})$ as defined in Equation (1) and (2), where x_{train} is the data from $\mathcal{D}_{train}$. We use the coefficients from the fitted model (m) denoted $\hat{\beta}_m$ to obtain predictions for our validation set using

$$\hat{y}^m = \hat{\beta}^m x_{val}$$

We can find the validation error E^m_{val} using

$$E^m_{val} = \frac{1}{N}\sum_{i=1}^{N}(y^m_{val_i} - y^{\hat{m}}_{val_i})^2$$

This is done for both model $m = 1$ and $m = 2$, the model (m) chosen is the one which corresponds to the lower E^m_{val}. The chosen model is then $g^*(x)$ which is used to determine E_{out} such that it is the MSE between the true model $y = 0.8x$ and the predictions $\hat{y} = \beta^* x$ plus σ^2. [1]. We are interested in how E_{val} and E_{out} are impacted by the validation set size, to determine this we can first see how E_{val} fairs.

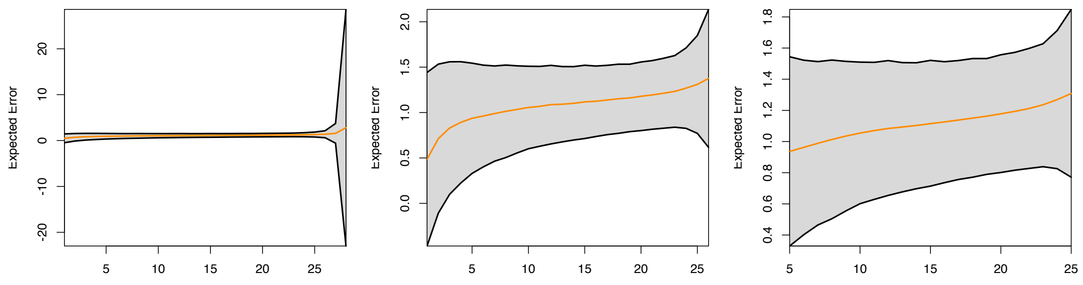


Figure 2: $E_{val} \pm \sigma_{val}$ for Different Validation Size Ranges.

From Figure (2) we see that if we use all the data as a validation set then then the the uncertainty in E_{val} as measured by σ_{val} goes to extreme values (left-most plot). As the range is decreased (middle plot) we begin to see that when the validation set size is neither too small nor too large the uncertainty is more stable, thus E_{val} may provide a good estimate of E_{out}.

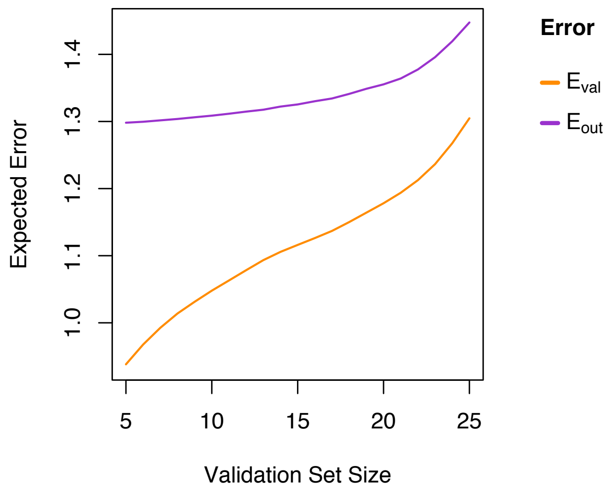


Figure 3: E_{val} and E_{out} for Validation Size $= 5,\ 6,\ \ldots,\ 25$ Averaged over 10000 Datasets.

[1] β^* represents the coefficients obtained from the training process of the model with the lowest E^m_{val}

Figure (3) illustrates the results of simulating 10 000 datasets of size $N = 30$, for each dataset $\mathcal{D}$ we subdivide it into $\mathcal{D}_{train}^{(-i)}$ and $\mathcal{D}_{val}^{(i)}$ where $\mathcal{D}_{train}^{(-i)}$ denotes the training set, a set excluding i observations and $\mathcal{D}_{val}^{(i)}$ denotes the validation set, a set only containing i observations. To be explicit, together $\mathcal{D}_{train}^{(-i)}$ and $\mathcal{D}_{val}^{(i)}$ contain all observations in $\mathcal{D}$. We consider $i = 5,\ 6,\ \ldots,\ 25$, the result is for each dataset we have training and validation sets of different sizes, this is done in the attempt to see how validation size impacts expected errors.

We can see thet E_{val} tends to represent a conservative estimate of E_{out}, when we set aside more data for validation there are fewer training data points hence the expected validation error increases [1]. As the validation set size approaches the dataset size the two measure tend towards each other, this is due to there being no learning in the model.

Problem 2

Suppose $x \sim U(-1, 1)$, we can simulate a dataset of size $N = 30$ and fit the models below

$$\begin{aligned} y_i &= \sin(\pi x_i) + \epsilon_i; \quad -1 \le x \le 1, i = 1, \ldots, N \\ \text{where: } &\epsilon_i \sim \text{Normal}(0, 1) \end{aligned} \tag{3}$$

i) Plot Data and Simulated Model

Figure (4) shows the simulated data and true underlying model $y_i = \sin(\pi x_i)$.

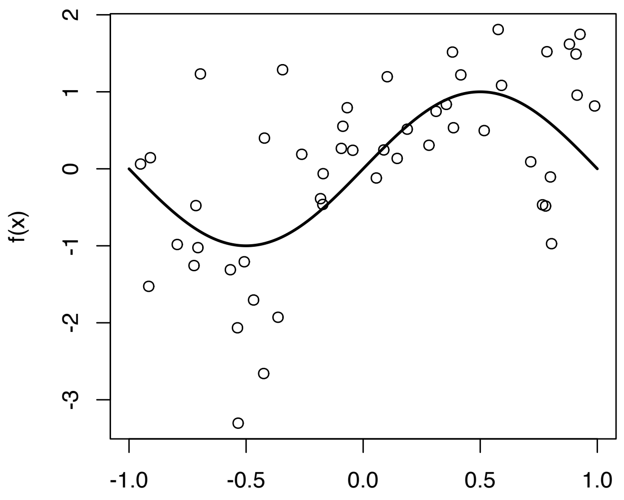


Figure 4: Simulated Model as per Equation (3).

ii) Regularisation

We can consider the model

$$y_i = \sum_{q=0}^{10} \beta_q L_q(x) \tag{4}$$

where $L_q(x)$ is is a Legendre polynomial of order q. To fit this polynomial model is a special case of fitting a linear model in a new space $\mathcal{Z}$. This is done under a non-linear transformation $\Phi : \mathcal{X} \to \mathcal{Z}$, where for the Q^{th} order polynomial, Φ transforms vector $\vec{x}$ into a $\vec{z}$ of Legendre polynomials, thus

$$\vec{z} = \begin{bmatrix} 1 & L_1(x) & L_2(x) & \dots & L_{Q_f}(x) \end{bmatrix}$$

In matrix notation we can define $\vec{y} = \mathbf{Z}\vec{w} + \vec{\epsilon}$ where;

$$\mathbf{Z} = \begin{bmatrix} \vec{z}_1 \\ \vec{z}_2 \\ \vdots \\ \vec{z}_N \end{bmatrix} = \begin{bmatrix} 1 & L_1(x_1) & \dots & L_Q(x_1) \\ 1 & L_1(x_2) & \dots & L_Q(x_2) \\ \vdots & \vdots & & \vdots \\ 1 & L_1(x_N) & \dots & L_Q(x_N) \end{bmatrix}$$

The optimal solution for an unconstrained weight vector is given by

$$\hat{\vec{w}} = \left(\mathbf{Z}'\mathbf{Z}\right)^{-1} \mathbf{Z}'\vec{y} \tag{5}$$

Thus for a constrained problem (including a regularisation parameter λ) we get

$$\hat{\vec{w}} = \left(\mathbf{Z}'\mathbf{Z} + \lambda\mathbf{I}\right)^{-1} \mathbf{Z}'\vec{y} \tag{6}$$

Where $\mathbf{I}$ is the $(Q_f + 1)$ identity matrix to penalise each coefficient by λ. This way complexity is controlled, we can see from Figure (5) that with $\lambda = 0$ there is a complex model fit, this is likely overfitting our data and will not generalise well to unseen data. However, for $\lambda = 5$ we see a curve that lies closer to the true model, although this is likely to perform worse in-sample the generalisation should perform better out-of-sample. The extreme case of λ being very large will simply fit a straight line through our data and is basically a naive fit/guess.

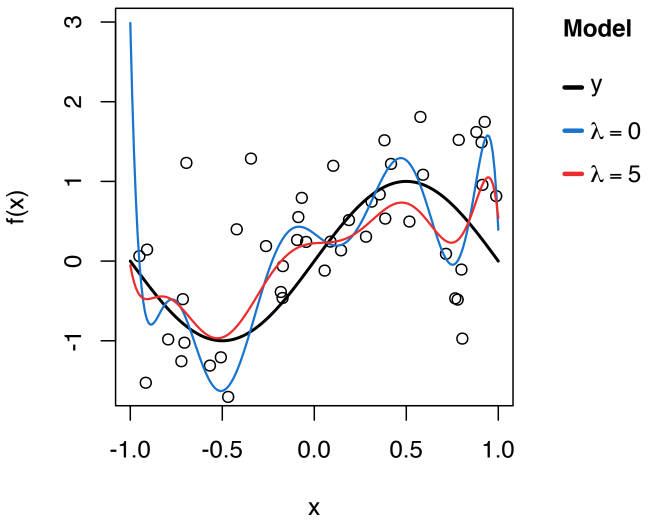


Figure 5: $L_{10}(x)$ Polynomial Fitted Models with Regularisation Parameter λ.

iii) Cross-Validation

So how do we choose the appropriate value for regularisation? We can use 10-fold cross-validation. This creates a quasi-OOS test of regularisation performance. We take our data and split it into training (IS) and testing (OOS) sets, we then get an OOS error based on the predictions between the true testing set response and our predicted response from the test set. The K-fold algorithm pseudo-code is shown in Algorithm (1).

Algorithm 1 10-Fold Cross Validation

```
1: procedure KFOLD.CV($x$, $\lambda$)    ▷ The CV error for data ($x$) and complexity parameter ($\lambda$)
2:     Define $n$ Observation, $K$ folds and $Q_f$ order.
3:     Shuffle the Data and Split data into $n/K$ groups
4:     $\mathbf{I} \leftarrow$ matrix    ▷ Initialise $(Q_f + 1) \times (Q_f + 1)$ Identity matrix
5:     for $fold \leftarrow 1, K$ do
6:         Set an index equal to $fold$.
7:
8:         Subset training data to be group not fold of the shuffled data.
9:         Denoting the predictors as $\boldsymbol{X}_{-fold}$ and response as $\vec{y}_{-fold}$
10:
11:        Subset testing data to be group fold of the shuffled data.
12:        Denoting the predictors as $\boldsymbol{X}_{fold}$ and response as $y_{fold}$
13:
14:        Calculate the $\hat{\vec{w}}$ coefficients as per Equation 6 using $\boldsymbol{X}_{-fold}$ and $\vec{y}_{-fold}$
15:        Calculate prediction $\hat{f}(\boldsymbol{X}_{fold}) = \boldsymbol{X}_{fold}\hat{\vec{w}}$
16:        Cross-Validation Error is $CV_{fold} = \frac{1}{K}\sum_{i=1}^{K}(y_{fold_i} - \hat{f}(\boldsymbol{X}_{fold_i}))$
17:    end for
18: end procedure
```

Thus for each λ we will get K number of CV errors, we can take the average of this for each value of λ to get an average error measure and find the λ that corresponds to the minimum of that. We generate a vector of regularisation parameters ($0.1 \leq \lambda \leq 10$) and obtain 10 CV errors per λ.

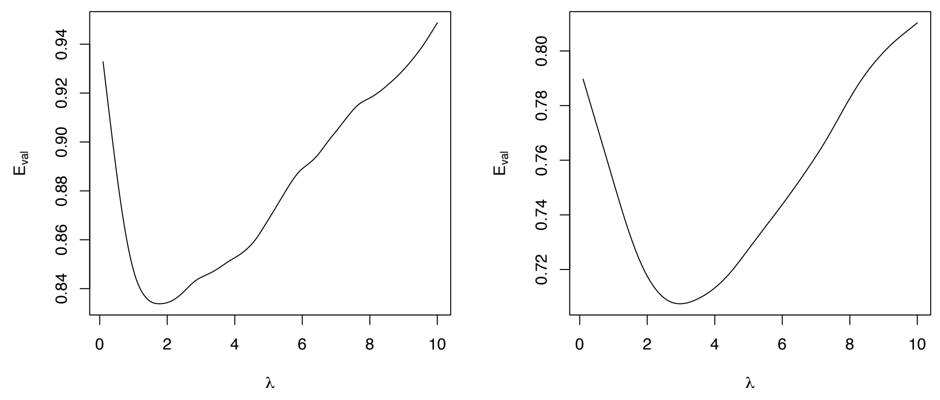


Figure 6: 10-fold Cross Validation Errors, Mean over folds (left) and median over folds (right).

From Figure (6) we can see that the over 10 folds, both the mean and median values show a fast initial decline in E_{val} before reaching some turning point — the point which acts as a proxy for where we begin to overfit. The median of CV per fold is plotted for sanity checking, but the minimum $\bar{CV}$ is taken to correspond with $\lambda^* \approx 1.7$. This λ^* is chosen and a model fit yielding Figure (7). We can see that the chosen model is more complex than $\lambda = 5$ in Figure (5), however, the fit seems reasonably close to the true model given the data.

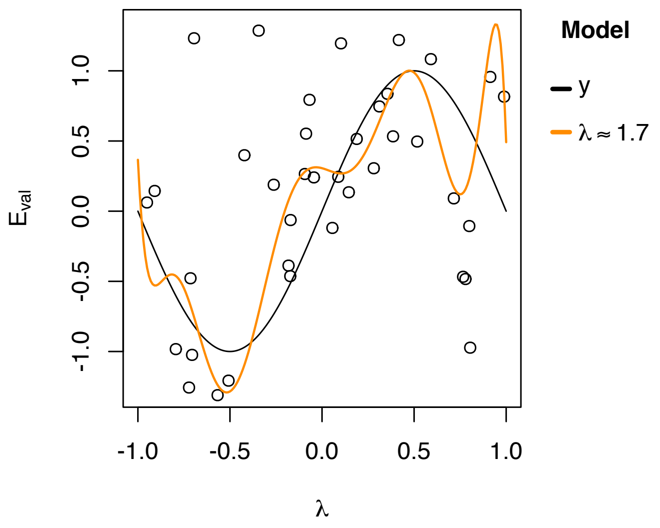


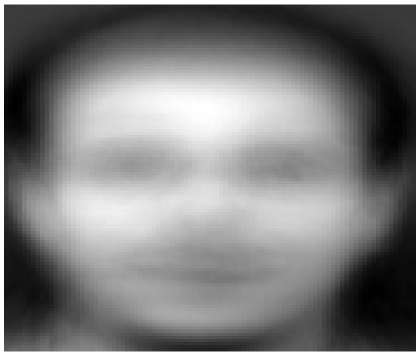
Figure 7: $L_{10}(x)$ Polynomial Fitted Models with Optimal Regularisation Parameter $\lambda \approx 2$ From 10-fold Cross Validation.

Problem 3

Here we will consider a dataset consisting of images from the faces.zip folder given to us. The folder contains $N = 400$ grey scale images which have dimension (width$\times$height) 92×112 or $p = 10,304$.

i) Image Scaling

First we convert each image into a matrix of grey colour values using image@grey command in R. We correctly orient each of the images and collapse this matrix into a vector which is subsequently bound with all other image vectors to give a big matrix $(N \times Wdth * Ht)$ representing all image colour values that has dimension (400×10304). As the rows represent each image we take the column means and column standard deviation to find the mean image (μ) and standard deviation (σ) image vector which is then "squashed" into a matrix.


(a) Mean Image

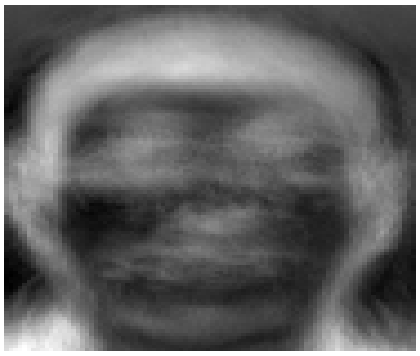

(b) Standard Deviation Image

Figure 8: Mean and Standard Deviation Images for 400 Images from faces.zip

We can use these two statistic matrices to scale our images, where the scaled image $\mathbf{Z}$ is represented by

$$\mathbf{Z} = \frac{\mathbf{X} - \mu}{\sigma} \tag{7}$$

The resulting original vs scaled image 168.pgm is plotted in Figure (9), we can see that the main difference is dark detail has been scaled to lighter shades of grey.

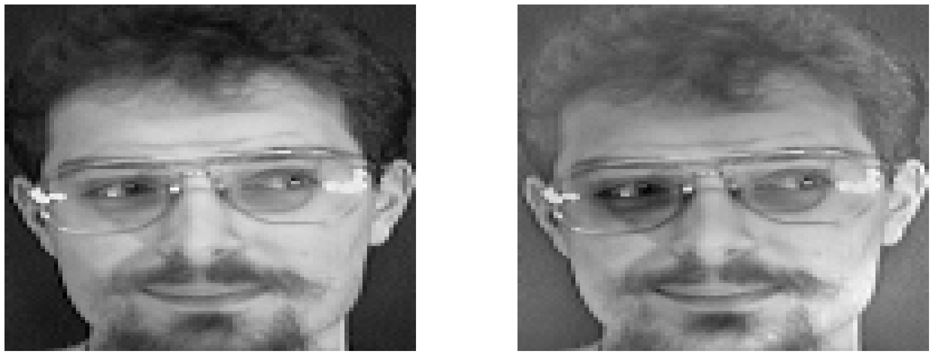

(a) Original (b) Scaled

Figure 9: Original ($\mathbf{X}$) and Scaled ($\mathbf{Z}$) Image for File `168.pgm`

ii) Eigenfaces

Eigenfaces are the set of eigenvectors used in human face recognition problems. The idea is to reduce the dimensions from (Height$\times$width) to a space where only the detail required to reproduce the image is kept. For basic facial recognition the dimensions need not be large as we can recognise a face as being a certain shape with eyes, a nose, a mouth etc. It is only when we need a higher level of detail that we would need more dimensions in our problem, reducing the dimension saves computational cost. In other words, as images of faces are very highly correlated we can compress the data to a low-dimensional subspace.

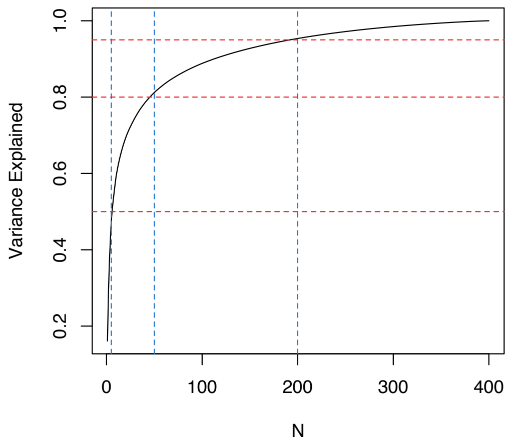


Figure 10: % Variance Explained as a Function of K Eigenfaces.

To determine how many eigenvectors are required we turn to Figure (10). Before reduction our covariance matrix ($\mathbf{\Sigma}$) is $10,304 \times 10,304$ thus yielding $10,304$ eigenvectors. In cases where

$N \ll p$ we can use an alternative definition for the covariance matrix. Let

$$\mathbf{\Sigma}' = \frac{1}{N}\mathbf{X}'\mathbf{X}$$

represent a matrix $\mathbf{A}$ which is $N \times N$. Now

$$\left(\frac{1}{N}\mathbf{X}\mathbf{X}'\right)\psi = \underline{\lambda}\psi$$

Where ψ is $\frac{1}{N}\mathbf{X}\mathbf{X}'$'s eigenvectors. We now have

$$\begin{aligned}
&\Rightarrow \mathbf{X}^T\left(\frac{1}{N}\mathbf{X}\mathbf{X}^T\right)\psi = \underline{\lambda}\left(\mathbf{X}^T\psi\right)\\
&\Rightarrow \left(\frac{1}{N}\mathbf{X}^T\mathbf{X}\right)\left(\mathbf{X}^T\psi\right) = \underline{\lambda}\left(\mathbf{X}^T\psi\right)\\
&\Rightarrow \phi = \left(\mathbf{X}^T\psi\right) \text{ the eigenvector of } \Sigma_{p\times p} = \frac{1}{N}\mathbf{X}^T\mathbf{X}
\end{aligned}$$

We know that the eigenvalues of the covariance matrix represent the variances of principal dimensions of our centered data, thus from Figure (10) shows the number of eigenfaces (5, 50, 200) explain about (50, 80, 95)% of the variance respectively. Choosing the number of eiegenfaces is thus a function of how detailed we would like our image to be.

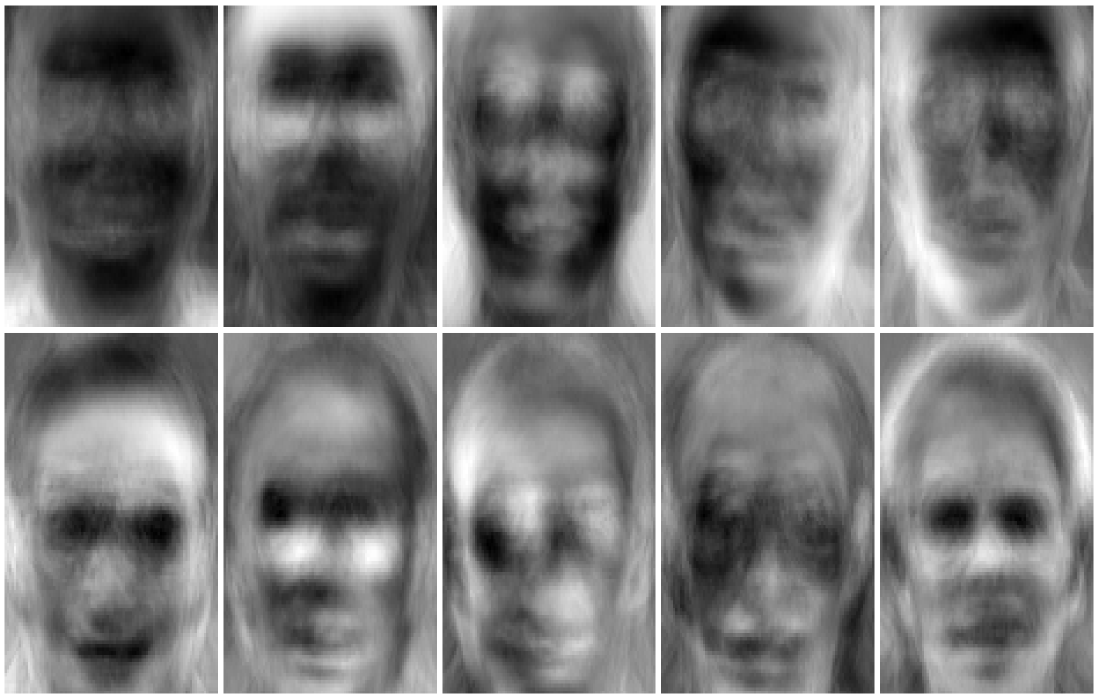
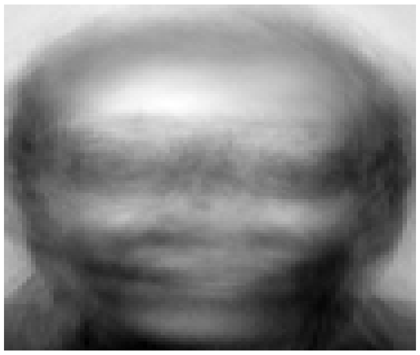


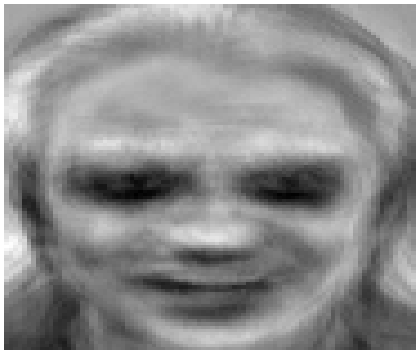
Figure 11: First 10 Eigenfaces.

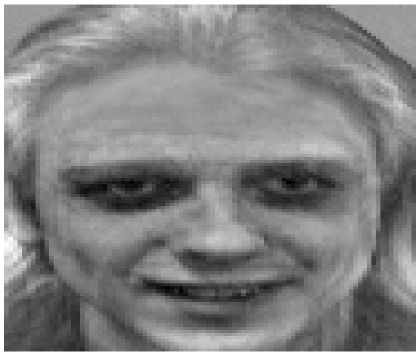
Figure (11) shows the first 10 eigenfaces, we can see that the general shape of a face with eyes a nose and mouth are captured relatively early, however, 10 eigenfaces is not enough to identify people from the dataset. As the eigenfaces increase so does the image detail.

iii) Image Reconstruction

Here we can use the above derivations of eigenfaces along with the scaled image 115.pgm to plot the reconstructed versions of the image when the first 5, 50 and 200 eigenfaces are used. From Figure (10) we know that these respectively explain (50, 80, 95)% of the variance, further supported by Figure (12). We can clearly see that when the first 5 eigenfaces are used we only have a rough outline of a face, we are unclear of gender or hairstyle but can nearly identify the image is likely a human face. Using the first 50 eigenfaces shows a clearer image, eyes are open with longish hair, distinctive nose with a concave mouth shape — overall the level of detail is sufficient for identifying a human face. Lastly when we use the first 200 eigenfaces (explaining 95% of the variance) we get an image close to the target (scaled image). Distinct features such as teeth, dimples and ears are visible. Depending on the objective for the image use we can determine that this is enough to uniquely identify this person.


(a) Ev = 5


(b) Ev = 50


(c) Ev = 200

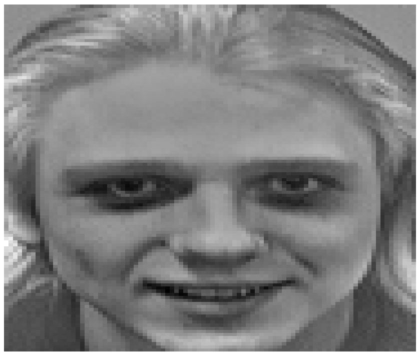

(d) Scaled Image

Figure 12: Scaled Image for File 115.pgm and Reconstructed Versions of the Image using Different Numbers of First Eigenfaces (Ev)

- END -

References

[1] Abu-Mostafa, Y. S. *Learning from data*, vol. 4.

Appendix

R Code

```r
##
#
# Author: Julian Albert
# Date: 22 September 2019
#
# Description:
# ML Assignment 2 - Regularisation, validation set size and image eigenfaces.
#
#------------------------------------------------------------------------------#

# 0. Clean Workspace, Directory and Library ----

## Clean Workspace
rm(list=ls())
dev.off() # Close Plots
setwd("~") # Clear Path to User

## Locations
project_folder <- "/Documents/UCT/Coursework/MachineLearning"
loc_script <- "/Assignment_2/UCT_Assignment/Code/MachineLearning_Ass2"
loc_figs <- "/Assignment_2/UCT_Assignment/Figs"
loc_data <- "/Assignment_2/UCT_Assignment/Code/MachineLearning_Ass2/Faces"

## Data File name
dat_faces_filen <- numeric()
for(i in 1:400) dat_faces_filen[i] <- paste(i, ".pgm", sep = "")

## Directories
dir_script <- paste("~", project_folder, loc_script, sep = '')
dir_figs <- paste("~", project_folder, loc_figs, sep = '')
dir_data <- paste("~", project_folder, loc_data, sep = '')

## Set Working Directory to Script Location
setwd(dir_script)

## Libraries - Lazyload
if (!require("pacman")) install.packages("pacman")
p_load(tidyverse, data.table, Cairo, parallel, doParallel, pixmap, rARPACK)

colour_vec <- c("dodgerblue3", "firebrick2", "forestgreen", "gold",
                "darkorange", "darkorchid3") # colour for pretty plot

# Problem 1 ----

# Simulate a Dataset

set.seed(123)
N <- 30
x_seq <- seq(-1, 1, length.out = N)
epsilon <- rnorm(N, 0, 1)
x_dat <- runif(N, -1, 1)
y_dat <- 0.8*x_dat + epsilon
data <- as.data.frame(cbind(x_dat, y_dat))

# fit models g1 and g2
g1_betas <- lm(y_dat ~ 0 + x_dat, offset = rep(+0.5, length(x_dat)))$coefficients
g2_betas <- lm(y_dat ~ 0 + x_dat, offset = rep(-0.5, length(x_dat)))$coefficients
g1_pred <- +0.5 + g1_betas*x_seq
g2_pred <- -0.5 + g2_betas*x_seq
```

```r

## i) plot models and fits
setwd(dir_figs)
cairo_pdf("ML_Ass2_fig_P1_modelfits.pdf", height = 5, width = 5)
par(mar = par()$mar + c(0,0,0,5), pty = 's') # larger margins so legend on side
plot(data$y_dat~data$x_dat, xlab = "x", ylab = "f(x)")
lines(x_seq, 0.8*x_seq, col = 'black', lwd = 2)
lines(g1_pred ~ x_seq, col = colour_vec[1], lwd = 1.5)
lines(g2_pred ~ x_seq, col = colour_vec[2], lwd = 1.5)
legend("topright", title = expression(paste(bold("Model"))),
       inset = c(-0.35, 0), lwd = 3, cex = 1, xpd = TRUE, bty = "n",
       legend = c(expression(y), expression(g[1](x)), expression(g[2](x))),
       col = c('black', colour_vec[1:2]), y.intersp = 1.5, x.intersp = 0.4,
       seg.len = 0.8)
dev.off()

# from the plot data seems to have upper and lower band, both models same?

## i) What if we fitted model 1000 times?
N_dat <- 30
N_sims <- 1000
x_seq <- seq(-1, 1, length.out = N_dat)
epsilon <- rnorm(N_dat, 0, 1)
x_dat <- runif(N_dat, -1, 1)
y_dat <- 0.8*x_dat + epsilon
data <- as.data.frame(cbind(x_dat, y_dat))
model <- 0.8*x_seq

g1_mat_pred <- matrix(NA, N_sims, N_dat)
g2_mat_pred <- matrix(NA, N_sims, N_dat)

for(i in 1:N_sims){

  x_dat <- runif(N_dat, -1, 1)
  epsilon <- rnorm(N_dat, 0, 1)
  y_dat <- 0.8*x_dat + epsilon

  # fit models g1 and g2
  g1_betas <- lm(y_dat ~ 0 + x_dat, offset = rep(+0.5, length(x_dat)))$coefficients
  g2_betas <- lm(y_dat ~ 0 + x_dat, offset = rep(-0.5, length(x_dat)))$coefficients
  g1_mat_pred[i, ] <- +0.5 + g1_betas*x_seq
  g2_mat_pred[i, ] <- -0.5 + g2_betas*x_seq

}

g1_pred_means <- colMeans(g1_mat_pred)
g2_pred_means <- colMeans(g2_mat_pred)

cairo_pdf("ML_Ass2_fig_P1_mean_modelfits.pdf", height = 5, width = 5)
par(mar = par()$mar + c(0,0,0,5), pty = 's') # larger margins so legend on side
plot(x_seq, model, lwd = 2, type = 'l', xlab = "x", ylab = "f(x)",
     ylim = c(min(g2_pred_means), max(g1_pred_means)))
lines(x_seq, g1_pred_means, lty = "dashed", col = colour_vec[1])
lines(x_seq, g2_pred_means, lty = "dashed", col = colour_vec[2])
legend("topright", title = expression(paste(bold("Model"))),
       inset = c(-0.35, 0), lwd = 3, cex = 1, xpd = TRUE, bty = "n",
       legend = c(expression(y), bquote(bar(g[1](x))), bquote(bar(g[2](x)))),
       col = c('black', colour_vec[1:2]), y.intersp = 1.5, x.intersp = 0.4,
       seg.len = 0.8)
dev.off()

## ii)

# Simulate 10'000 datasets of y size = 30
n_datasets <- 10000
x_seq <- seq(-1, 1, length.out = 1000)
true_model <- 0.8*x_seq

func.sim_dataset <- function(n_datasets, N)
{
  # initialise data storage
  data_df <- list()
  # gen x, y
  for(i in 1:n_datasets){
    x_dat <- runif(N, -1, 1)
    epsilon <- rnorm(N, 0, 1)
    y_dat <- 0.8*x_dat + epsilon
    data_df[[i]] <- data.frame(X = x_dat, Y = y_dat)
  }
  return(data_df)
}

dat_datasets <- func.sim_dataset(n_datasets, N)
## Calc Eval and Eout
func.Eval_Eout <- function(data, i_start, i_end, x_seq, true_model)
{

  ## Initialise Storage...
  ## Need a matrix of Eval and Eout for each i
  mat_of_errors <- matrix(0, nrow = length(i_start:i_end), ncol = 2)

  # For Dval of size 5, 6, ..., 25 and Dtrain of size 25, 24, ..., 5
  for(i in i_start:i_end)
  {
    # Subset data into training and validation
    idx <- sample(1:NROW(data), i, replace = FALSE)
    data_validation <- data[idx, ]
    data_train <- data[-idx, ]
    # fit models g1 and g2
    g1_betas <- lm(Y ~ 0 + X, offset = rep(+0.5, length(X)), data_train)$coefficients
    g2_betas <- lm(Y ~ 0 + X, offset = rep(-0.5, length(X)), data_train)$coefficients
    g1_pred <- +0.5 + g1_betas*data_validation$X
    g2_pred <- -0.5 + g2_betas*data_validation$X
    # Calculate 1/n * (Y - g(Y)_val)^2
    MSE_g1 <- mean((data_validation$Y - g1_pred)^2)
    MSE_g2 <- mean((data_validation$Y - g2_pred)^2)

    E_val <- min(MSE_g1, MSE_g2)

    # select model with lower MSE
    if(MSE_g1 < MSE_g2){
      gstar_betas <- g1_betas
      gstar_pred <- +0.5 + gstar_betas*(x_seq)
    } else {
      gstar_betas <- g2_betas
      gstar_pred <- -0.5 + gstar_betas*(x_seq)
    }

    # Calculate OOS error and store
    E_out <- mean( (gstar_pred - true_model)^2 ) + 1# ?
    mat_of_errors[(i-i_start+1), ] <- c(E_val, E_out)
  }

  # Name columns to not get confused by which measure is which
  dat_erros_val_and_oos <- as.data.frame(mat_of_errors)
  names(dat_erros_val_and_oos) <- c("E_val", "E_out")

  return(dat_erros_val_and_oos)
}

## Parallel, 21 linear models over 10'000 datasets = 210'000 linear models
system.time(
dat_Errors <- mclapply(dat_datasets, # list
                       func.Eval_Eout, i_start = 5, i_end = 25,
                       x_seq = x_seq, true_model = true_model, # function & pars
                       mc.cores = detectCores()) # how many cores
)

# Select Eval and Eout to plot
Evals <- lapply(dat_Errors, function(x) x$E_val)
Eouts <- lapply(dat_Errors, function(x) x$E_out)
Evals_Mean <- bind_cols(Evals) %>% rowMeans()
Eouts_Mean <- bind_cols(Eouts) %>% rowMeans()

cairo_pdf("ML_Ass2_fig_P1_EvalEout.pdf", height = 5, width = 5)
par(mar = par()$mar + c(0,0,0,5), pty = 's') # larger margins so legend on side
```

```r
plot(smooth.spline(y = Evals_Mean, x = 5:25), type = 'l', col = colour_vec[5],
     ylim = c(min(Evals_Mean), max(Eouts_Mean)), lwd = 1.5,
     xlab = "Validation Set Size", ylab = 'Expected Error')
lines(smooth.spline(y = Eouts_Mean, x = 5:25), col = colour_vec[6], lwd = 1.5)
legend("topright", title = expression(paste(bold("Error"))),
       inset = c(-0.35, 0), lwd = 3, cex = 1, xpd = TRUE, bty = "n",
       legend = c(expression(E[val]), expression(E[out])),
       col = colour_vec[5:6], y.intersp = 1.5, x.intersp = 0.4,
       seg.len = 0.8)
dev.off()

# Why between 5:25? what happens over full range cant have val of size 30 in lm()
system.time(
  dat_Errors <- mclapply(dat_datasets, # list
                         func.Eval_Eout, i_start = 1, i_end = 29,
                         x_seq = x_seq, true_model = true_model, # function & pars
                         mc.cores = detectCores()) # how many cores
)

## What happens to uncertainty around Eval?
Evals <- lapply(dat_Errors, function(x) x$E_val)
Evals_Mean <- bind_cols(Evals) %>% rowMeans()
Evals_Mean <- Evals_Mean[-29]
Evals_sd <- bind_cols(Evals) %>% apply(1, function(x) sd(x))
Evals_sd <- Evals_sd[-29]
cairo_pdf("ML_Ass2_fig_P1_EvalwSigma_128.pdf", height = 5, width = 5)
plot(x = c(1, 28), y = c(min(Evals_Mean-Evals_sd), max(Evals_Mean+Evals_sd)), type = 'n',
     xlab = "Validation Set Size", ylab = 'Expected Error', yaxs = 'i', xaxs = 'i')
polygon(c(1:28, rev(1:28)), c( (Evals_Mean-Evals_sd), rev(Evals_Mean+Evals_sd) ),
        col = "grey85")
lines(smooth.spline(y = Evals_Mean, x = 1:28), col = colour_vec[5], lwd = 2)
lines(smooth.spline(y = Evals_Mean+Evals_sd, x = 1:28), lwd = 2)
lines(smooth.spline(y = Evals_Mean-Evals_sd, x = 1:28), lwd = 2)
dev.off()

Evals <- lapply(dat_Errors, function(x) x$E_val)
Evals_Mean <- bind_cols(Evals) %>% rowMeans()
Evals_Mean <- Evals_Mean[-c(27, 28, 29)]
Evals_sd <- bind_cols(Evals) %>% apply(1, function(x) sd(x))
Evals_sd <- Evals_sd[-c(27, 28, 29)]
cairo_pdf("ML_Ass2_fig_P1_EvalwSigma_126.pdf", height = 5, width = 5)
plot(x = c(1, 26), y = c(min(Evals_Mean-Evals_sd), max(Evals_Mean+Evals_sd)), type = 'n',
     xlab = "Validation Set Size", ylab = 'Expected Error', yaxs = 'i', xaxs = 'i')
polygon(c(1:26, rev(1:26)), c( (Evals_Mean-Evals_sd), rev(Evals_Mean+Evals_sd) ),
        col = "grey85")
lines(smooth.spline(y = Evals_Mean, x = 1:26), col = colour_vec[5], lwd = 2)
lines(smooth.spline(y = Evals_Mean+Evals_sd, x = 1:26), lwd = 2)
lines(smooth.spline(y = Evals_Mean-Evals_sd, x = 1:26), lwd = 2)
dev.off()

Evals <- lapply(dat_Errors, function(x) x$E_val)
Evals_Mean <- bind_cols(Evals) %>% rowMeans()
Evals_Mean <- Evals_Mean[-c(1:4, 26:29)]
Evals_sd <- bind_cols(Evals) %>% apply(1, function(x) sd(x))
Evals_sd <- Evals_sd[-c(1:4, 26:29)]

cairo_pdf("ML_Ass2_fig_P1_EvalwSigma_525.pdf", height = 5, width = 5)
plot(x = c(5, 25), y = c(min(Evals_Mean-Evals_sd), max(Evals_Mean+Evals_sd)), type = 'n',
     xlab = "Validation Set Size", ylab = 'Expected Error', yaxs = 'i', xaxs = 'i')
polygon(c(5:25, rev(5:25)), c( (Evals_Mean-Evals_sd), rev(Evals_Mean+Evals_sd) ),
        col = "grey85")
lines(smooth.spline(y = Evals_Mean, x = 5:25), col = colour_vec[5], lwd = 2)
lines(smooth.spline(y = Evals_Mean+Evals_sd, x = 5:25), lwd = 2)
lines(smooth.spline(y = Evals_Mean-Evals_sd, x = 5:25), lwd = 2)
dev.off()

# Problem 2 ----

## Simulate dataset
set.seed(123)
N = 50
x_seq <- seq(-1, 1, 0.01)
x_dat <- runif(N, -1, 1)
```

```r
epsilon <- rnorm(N, 0, 1)
y_dat <- sin(pi*x_dat) + epsilon
data <- as.data.frame(cbind(x_dat, y_dat))
model <- sin(pi*x_seq)
Qf <- 10

setwd(dir_figs)
cairo_pdf("ML_Ass2_fig_P2_sim_model_init.pdf", height = 5, width = 5)
plot(x_seq, model, type = 'l', lwd = 2, xlab = "x", ylab = "f(x)",
     ylim = c(min(y_dat), max(y_dat)))
points(data)
dev.off()

## Legendre polynomial function
func.Legendre <- function(x, q)
{
  Lq = 0
  for(i in 0:q){
    Lq = Lq + ((x^i)*choose(q, i)*choose((q+i-1)/2, q))
  }
  return((2^q)*Lq)
}

## Target function with q-th order Legendre Polynomial
func.rand_Legfunc <- function(x, q)
{
  fx = 0
  beta_vec = runif(q+1, -1, 1)
  for(i in 0:q){
    fx = fx + (func.Legendre(x, i)*beta_vec[(i+1)])
  }
  return(fx)
}

### Takes X -> Z maps to Z... z = [1 L1 L2 ... LQ]'
### Z = [z1 z2 ... zN]' where z1 = [1 1 ... 1]', z2 = [L1(x1) L1(x2) ... L1(x3)]'

## Fit model
func.fit_model <- function(X, Y, Qf, lambda = 0)
{
  # initialise Z
  Z <- matrix(0, nrow = NROW(X), ncol = (Qf+1))
  # Lq for each column
  for(i in 0:Qf){
    Z[, (i+1)] <- func.Legendre(X, i)
  }
  # calc w* and hence predict
  I <- diag(NCOL(Z))
  Hat_mat <- Z %*% solve(crossprod(Z) + lambda*I) %*% t(Z)
  Beta_Hat <- solve(crossprod(Z) + lambda*I) %*% t(Z) %*% Y
  Y_hat <- Z %*% Beta_Hat

  return(list(Hat_mat = Hat_mat, Beta_Hat = Beta_Hat, Y_hat = Y_hat))
}

test0 <- func.fit_model(x_dat, y_dat, Qf, lambda = 0)
test5 <- func.fit_model(x_dat, y_dat, Qf, lambda = 5)

func.predict <- function(x_seq, betas, Qf)
{
  Z <- matrix(0, nrow = NROW(x_seq), ncol = (Qf+1))
  for(i in 0:Qf){
    Z[, (i+1)] <- func.Legendre(x_seq, i)
  }
  pred <- Z %*% betas
  return(pred)
}

pred0 <- func.predict(x_seq, test0$Beta_Hat, Qf)
pred10 <- func.predict(x_seq, test5$Beta_Hat, Qf)

setwd(dir_figs)
cairo_pdf("ML_Ass2_fig_P2_modelfits.pdf", height = 5, width = 5)
```

```r
par(mar = par()$mar + c(0,0,0,5), pty = 's') # larger margins so legend on side
plot(x_seq, model, type = 'l', lwd = 2, xlab = "x", ylab = "f(x)",
     ylim = c(min(pred0, pred10), max(pred0, pred10)))
points(data)
lines(x_seq, pred0, col = colour_vec[1], lwd = 1.5)
lines(x_seq, pred10, col = colour_vec[2], lwd = 1.5)
legend("topright", title = expression(paste(bold("Model"))),
       inset = c(-0.35, 0), lwd = 3, cex = 1, xpd = TRUE, bty = "n",
       legend = c(expression(y), expression(lambda==0), expression(lambda==5)),
       col = c('black', colour_vec[1:2]), y.intersp = 1.5, x.intersp = 0.4,
       seg.len = 0.8)
dev.off()

## Cross validation - K-fold
func.cv <- function(lambda, data, k.folds)
{
  ## is data a subset of the fold number?
  stopifnot(NROW(data) %% k.folds == 0)
  ## data shuffle all the rows
  data <- as_tibble(data)
  data_shuffled <- sample_frac(data, 1L)
  MSE <- numeric()
  ## Split data into k folds
  folds <- rep(1:k.folds, nrow(data_shuffled)/k.folds)
  data_in_folds <- split(data_shuffled, folds)

  for(fold in 1:k.folds)
  {
    ## Use fold as validation, all else as testing
    dat_train <- bind_rows(data_in_folds[-fold])
    dat_validate <- bind_rows(data_in_folds[fold])
    # fit model with specified lambda
    beta_hat <- func.fit_model(dat_train$x_dat, dat_train$y_dat, 10, lambda)$Beta_Hat
    yhat <- func.predict(dat_validate$x_dat, beta_hat, 10)
    MSE[fold] <- mean((yhat - dat_validate$y_dat)^2)
  }
  # MSE for the 10 folds and specified lambda value
  return(MSE)
}

# For different values of lambda
lambda_vec <- seq(0.1, 10, length.out = 500)
testcv <- mclapply(lambda_vec, func.cv, data = data, k.folds = 10,
                   mc.cores = detectCores())

cairo_pdf("ML_Ass2_fig_P2_CVerrors.pdf", height = 5, width = 10)
par(mfrow = c(1, 2))
lapply(testcv, mean) %>% # mean over folds...
  unlist %>%
  smooth.spline(x = lambda_vec) %>%
  plot(type = 'l', xlab = expression(lambda),
       ylab = expression(E[val]))
lapply(testcv, median) %>% # median over folds...
  unlist %>%
  smooth.spline(x = lambda_vec) %>%
  plot(type = 'l', xlab = expression(lambda),
       ylab = expression(E[val]))
dev.off()

# approximate lambda based on plots
lambdaOpt <- 1.70
testOpt <- func.fit_model(x_dat, y_dat, Qf = 10, lambda = lambdaOpt)
predOpt <- func.predict(x_seq, testOpt$Beta_Hat, Qf)

setwd(dir_figs)
cairo_pdf("ML_Ass2_fig_P2_opt_fit.pdf", height = 5, width = 5)
par(mar = par()$mar + c(0,0,0,5), pty = 's') # larger margins so legend on side
plot(x_seq, model, xlab = bquote(lambda), ylab = bquote(E[val]), type = 'l',
     ylim = c(min(predOpt), max(predOpt)))
points(data)
lines(x_seq, predOpt, col = colour_vec[5], lwd = 1.5)
legend("topright", title = expression(paste(bold("Model"))),
       inset = c(-0.35, 0), lwd = 3, cex = 1, xpd = TRUE, bty = "n",
       legend = c(expression(y), bquote(lambda%~~%.(lambdaOpt))),
       col = c('black', colour_vec[5]), y.intersp = 1.5, x.intersp = 0.4,
       seg.len = 0.8)
dev.off()

# Problem 3 ----

## Initial pars, 400 92x112 images
N <- 400; Ht <- 112; Wdth <- 92
## Read in images
images <- mclapply(1:N, function(i){
  read.pnm(paste(dir_data, "/", dat_faces_filen[i], sep = ""), cellres = 1)},
  mc.cores = (detectCores() - 1))

# image(images[[1]]@grey, col = grey(seq(0, 1, length = 256)), axes = FALSE)

## Rotate images to correct orientation
func.rotate <- function(x) t(apply(x, 2, rev))
image_oriented <- lapply(images, function(image){ func.rotate(image@grey) })
## Oriented Images need to be in vector format
image_oriented_vec <- lapply(image_oriented, as.vector) %>% do.call("rbind", .)
## Rows are image no. indices... column mean/sd gives pixel mean across all images
image_mean <- colMeans(image_oriented_vec) %>% matrix(., nrow = Wdth, ncol = Ht)
image_sd <- apply(image_oriented_vec, 2, sd) %>% matrix(., nrow = Wdth, ncol = Ht)

## Plot mean and standard deviation image
setwd(dir_figs)
cairo_pdf("ML_Ass2_fig_P3_mean_image.pdf", height = 5, width = 5)
image(image_mean, col = grey(seq(0, 1, length = 256)), axes = FALSE)
dev.off()

cairo_pdf("ML_Ass2_fig_P3_sd_image.pdf", height = 5, width = 5)
image(image_sd, col = grey(seq(0, 1, length = 256)), axes = FALSE)
dev.off()

## Scale images
image_scaled <- lapply(image_oriented,
                       function(image){ (image - image_mean) / image_sd })

## Image 168... plot original (oriented) and scaled
n <- 168

cairo_pdf("ML_Ass2_fig_P3_orient_image168.pdf", height = 5, width = 5)
image(image_oriented[[n]], col = grey(seq(0, 1, length = 256)), axes = FALSE)
dev.off()

cairo_pdf("ML_Ass2_fig_P3_scaled_image168.pdf", height = 5, width = 5)
image(image_scaled[[n]], col = grey(seq(0, 1, length = 256)), axes = FALSE)
dev.off()

## Want each row to be a wdth*ht vector
images_scaled_vec <- lapply(image_scaled,
                            function(image){ as.vector(t(image)) }) %>%
  do.call("rbind", .)

## Get Eigenvalues/Vectors using prcomp >> unit eigenvectors
### N << p (much smaller) do eigen decomp on smaller matix
### rather use XXt
## new eigen vector is Xt psi
A <- 1/N * images_scaled_vec %*% t(images_scaled_vec)
PCA <- prcomp(A)
image_eigen_vectors <- t(images_scaled_vec) %*% PCA$rotation
image_eigen_vectors_norm <- apply(image_eigen_vectors, 2, function(x){ x/sqrt(sum(x^2)) })

check_scree <- eigen(A)
check_eigs <- t(images_scaled_vec) %*% check_scree$vectors
setwd(dir_figs)
cairo_pdf("ML_Ass2_fig_P3_eigenvalues.pdf", height = 5, width = 5)
plot(cumsum(check_scree$values)/sum(check_scree$values), type = 'l',
xlab = "N", ylab = "Variance Explained")
abline(h = c(0.5, 0.8, 0.95), lty = 'dashed', col = colour_vec[2])
abline(v = c(5, 50, 200), lty = 'dashed', col = colour_vec[1])
dev.off()
```

```r

## Plot first ten eigenfaces

cairo_pdf("ML_Ass2_fig_P3_eigenfaces.pdf", height = 7.5, width = 10)
par(mfrow = c(2, 5))
par(mar = c(0.2, 0.2, 0.2, 0.2))
for (i in 1:10){
  eigen_vec_mat <- matrix(image_eigen_vectors_norm[, i], nrow = 92, byrow = TRUE)
  image(eigen_vec_mat, col = grey(seq(0, 1, length = 256)), axes = FALSE)
}
dev.off()

## reconstruct an image based on eigenfaces used
func.reconstruct <- function(images_scaled_vec, image_eigen_vectors_norm)
{
  ## Project onto the Eigen Space
  project.eig <- t(data.matrix(images_scaled_vec)) %*% image_eigen_vectors_norm
  ## Project back
  reconstruct.img <- project.eig %*% t(image_eigen_vectors_norm)
  ## Plot the image... Vector needs to be matrix
  image(matrix(reconstruct.img, nrow = 92, byrow = TRUE),
        col = grey(seq(0, 1, length = 256)), axes = FALSE)
}

## Image 115, c(x, y) corresponds to number of eigenfaces used
n = 115
cairo_pdf("ML_Ass2_fig_P3_recon5_image115.pdf", height = 5, width = 5)
func.reconstruct(images_scaled_vec[n, ], image_eigen_vectors_norm[, c(1:5)])
dev.off()

cairo_pdf("ML_Ass2_fig_P3_recon50_image115.pdf", height = 5, width = 5)
func.reconstruct(images_scaled_vec[n, ], image_eigen_vectors_norm[, c(1:50)])
dev.off()

cairo_pdf("ML_Ass2_fig_P3_recon200_image115.pdf", height = 5, width = 5)
func.reconstruct(images_scaled_vec[n, ], image_eigen_vectors_norm[, c(1:200)])
dev.off()

cairo_pdf("ML_Ass2_fig_P3_scaled_image115.pdf", height = 5, width = 5)
image(image_scaled[[n]], col = grey(seq(0, 1, length = 256)), axes = FALSE)
dev.off()
```

Plagiarism Declaration Form

A copy of this form, completed and signed, to be attached to all coursework submissions to the Statistical Sciences Department.

COURSE CODE:	**STA5068Z**
COURSE NAME:	**Machine Learning**
STUDENT NAME:	**Julian Albert**
STUDENT NUMBER:	**ALBJUL005**
GROUP NUMBER:	**1**

PLAGIARISM DECLARATION

- I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
- I have used a generally accepted citation and referencing style. Each contribution to, and quotation in, this tutorial/report/project from the work(s) of other people has been attributed, and has been cited and referenced.
- This tutorial/report/project is my own work.
- I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.
- I acknowledge that copying someone else's assignment or essay, or part of it, is wrong, and declare that this is my own work.
- Agreement to this statement does not exonerate me from the University's plagiarism rules.

Signature: [illegible] Date: September 30, 2019