STA5068Z: Machine Learning
Regularization & Dimensionality Reduction

Zach Wolpe

02 November 2020

Abstract

In this paper, we examine the importance of regularization & model validation - an imperative part of any machine learning application. We implement an instance of K fold cross validation, as well as examine theoretical results in the limit. Finally, we showcase an application of Principal Component Analysis (PCA) for dimensionality reduction to represent image data in a lower dimensional feature space for the purpose of facial recognition.

Question 1

1.i. Two Linear Models

Figure 2 shows the the data generated process, the data, & the two fitted models.

The two models force the fit through $y = -0.5$ and $y = 0.5$ respectively. Thus are individually biased in the expectation - in the limit. However if we average over both models we should find an unbiased model - as the true intercept term is 0.

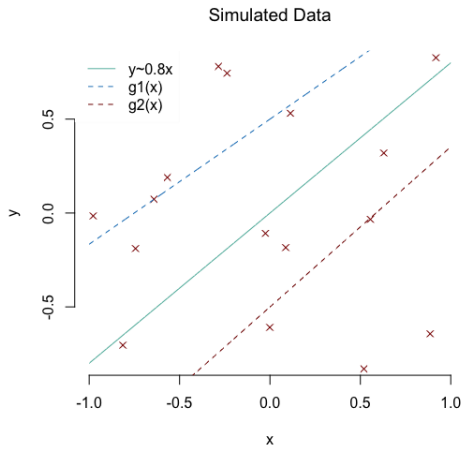

Figure 2: Fitting g1(x) and g2(x) to the generated data.

1.ii. Expectations: Repeated Experimentation

Here we perform an experiment that splits a generated dataset into various train/validation splits / thereafter compute the validation error:

$$E_{val}[g^*(x)] = \frac{1}{K}\sum_k e(g^*(x), f(x))$$

.

The experiment was conducted 10′000 times, thereafter we can compute the expected out of sample error:

$$\begin{aligned}\mathbf{E}_{\mathcal{D}}[E_{out}[g^*(x)]] &= \mathbf{E}_x[(\mu(x) - f(x))^2] + \mathbf{E}_{x,D}[g(x) - \mu(x)] + \mathbf{E}_{x,D}[\epsilon(x)^2] \\ &= bias^2 + variance + \sigma^2\end{aligned} \tag{1}$$

We begin by analysing $\mu(x)$ - the average over all best hypothesis. As expected, figure 3 shows that the bias is effectively 0 - estimated by averaging over all runs. The average best fit almost perfectly tracks the true target function.

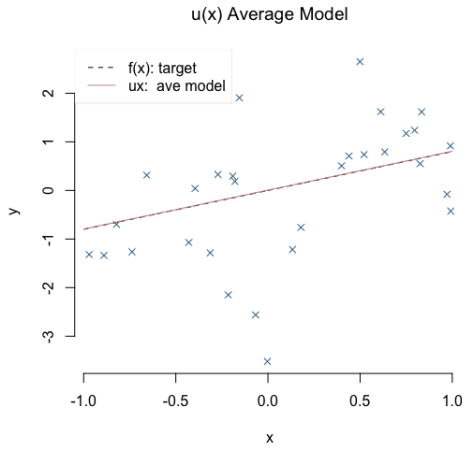

Figure 3: Average model over all datasets.

It is then evident that almost all out of sample error is captured by the variance & noise in the model - again this is expected as the model is sufficient complex to capture the true underlying function (which happens to only require a single parameter).

If we then average the out of sample error & validation error over the various levels of train/validation split we produce figure 4.

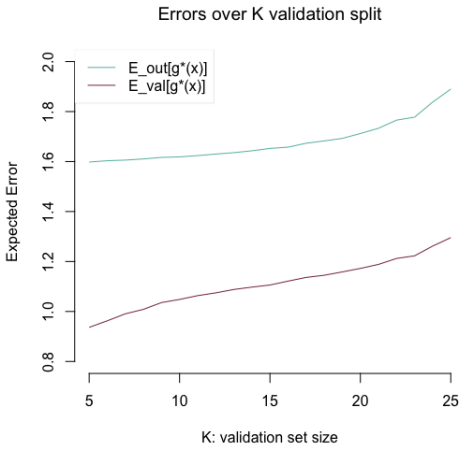

Figure 4: Average errors over $10'000$ experiments.

Two notable trends are evident in figure 4: firstly, and most obviously, both validation errors and out of sample errors increase in k. This is expected as we

are training on smaller training sets (datasets of size $N - k$) which results in less data to learn the true underlying function, thus worse results on average.

A second, less obvious observation, is that (in an unbiased model) the validation error converges to the out of sample error as k grows (the difference between the curves shrinks). This is because we are better able to approximate the out of sample error with larger validation sets.

In our case, we do not see sufficient convergence, this is due to the biased induced by forcing the fit through either $y = 0.5$ or $y = -0.5$. The model is unable to capture the true target due to this forced bias, thus the validation error cannot converge to the true out of sample error. The discrepancy between the curves does, however, shrink - though marginally.

This dichotomous relationship can be summarized by the expression:

$$E_{out}[g] \approx E_{out}[g^-] \approx E_{val}[g^-]$$

That is to say, large k allows the validation error to closely track/approximate the training set out of sample error (right approximation) whilst small k allows the out of sample error on the training set to approximate the true out of sample error on the full dataset.

To illustrate why our validation error could not converge to the true out of sample error, I ran the experiment once more however fitting a simple linear model $y\ a + bx$ learning both parameters a and b. The repeated the experiment as above, to produce figure 5:

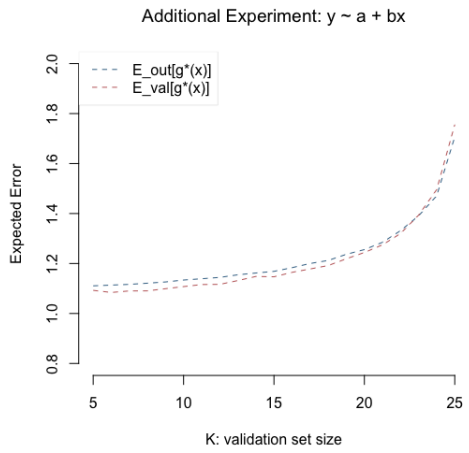

Figure 5: Fitting $y\ a + bx$ over $10'000$ experiments.

Figure 5 shows that the validation error of this unbiased model is able to perfectly track the out of sample error, even over small k - probably a consequence of the very simple underlying true function being perfectly encapsulated by the model.

Question 2

2.i. Data Generating Process

Figure 6 shows the the data generated process (target function) & the randomly generated data - with noise.

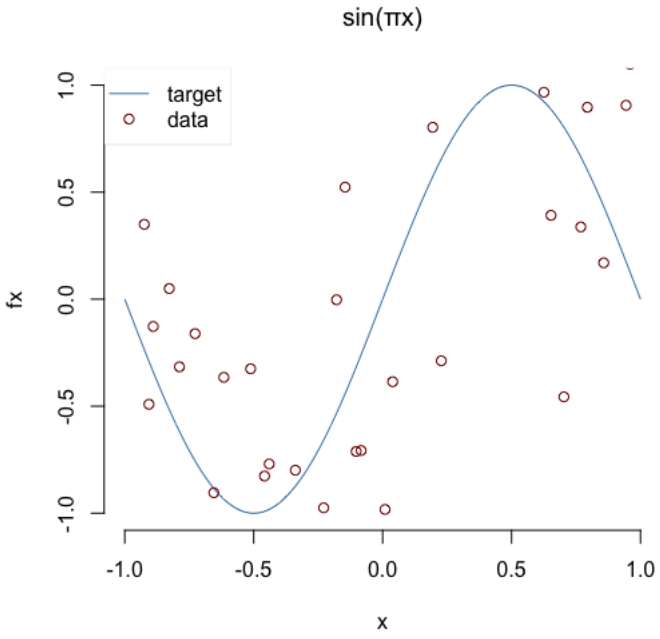

Figure 6: The generated data distribution & generated data.

2.ii. Effects of Regularization

Utilizing the Legendre polynomials as basis functions:

$$y_i = \sum_{q=0}^{10} \beta_q \mathcal{L}_q(x)$$

We fit the Legendre polynomial basis functions for various orders of Q_f. Though the question requires $Q_f = 10$, we fit the model for varying values of Q_f in order to assess the effect of model complexity. For each Q_f, we contrast an unconstrained model - $\lambda = 0$ - with a severely constrained model - $\lambda = 5$.

Firstly lets examining the case where $Q_f = 10$ shown in figure 7. We see that the regularization - though harsh - effectively controls the parameterization: producing a far more realistic, smoother, curve that much more closely

approximates the true target function. It is possible that the penalty $\lambda = 5$ is far too severe & better out of sample error could be achieved utilizing a less aggressive penalty.

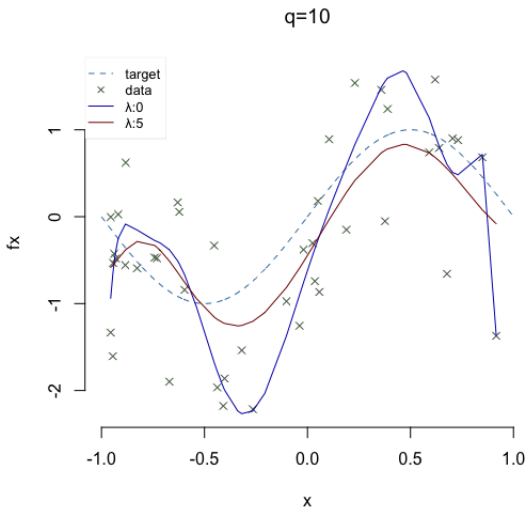

Figure 7: Model 10^{th} order Legendre polynomials: $Q_f = 10$

One might find it interesting to fit the polynomial basis over a range of Q_f's. The results of fitting the values for $Q_f = \{0, 1, ..., 13\}$ are available in figure 8 & 9.

Regularized models are fairly homogeneous - though models with many more degrees of freedom exhibit slightly less smooth regularized models, they still capture the essence of the true target function. The importance of regularization is self-evident.

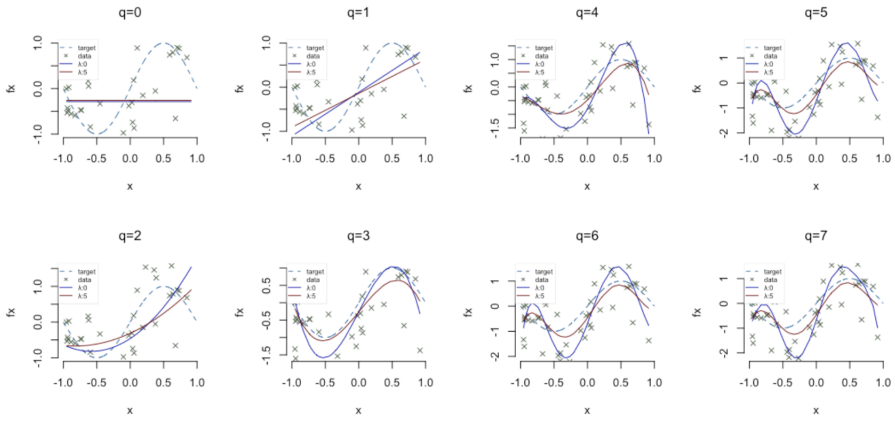

Figure 8: Regularized & unregularized model for varying order polynomials.

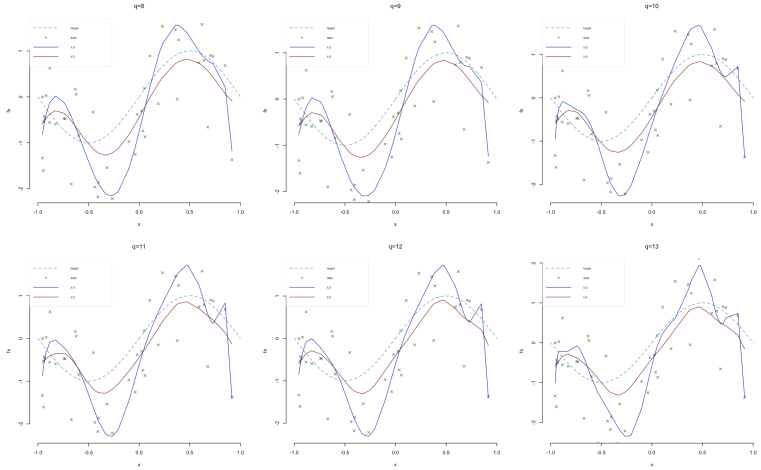

Figure 9: Regularized & unregularized model for varying order polynomials.

2.iii. 10-Fold Cross Validation

For each λ, we average validation errors over each validation set, to produce figure 10. λ is taken such that it minimises this out of sample estimate (validation error). Very little regularization is required to achieve the fit that ought to maximise generalization.

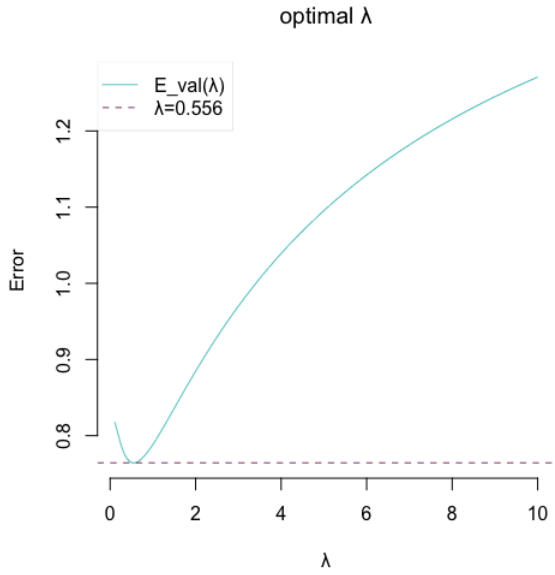

Figure 10: Finding the optimal λ: that which minimizes the validation error over the K folds.

Utilizing this value of $\lambda = 0.556$ we fit the data to the model to produce figure 11. The fit is generally smooth, it appears a bit more elongated than the true known target function however that is merely a consequence of the sampling variation of the randomly generated dataset.

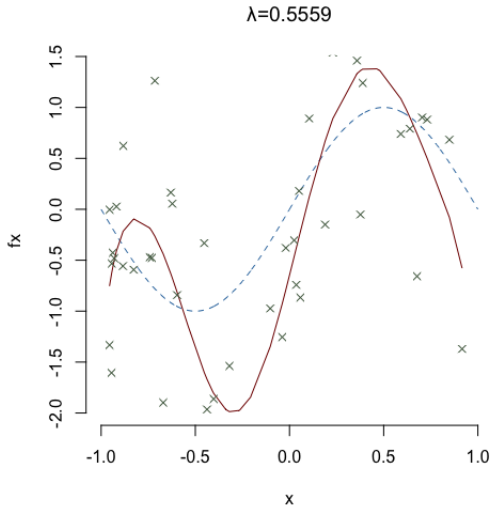

Figure 11: The optimal level of regularization - that which minimizes the validation error (a proxy for out of sample error). That is, the model that generalizes best to unseen data.

For good measure, we plot the fits for various levels of λ, seen in figure 12. One ought to note that negative λ values appear to increase the degrees of freedom / variation in the model. Large values of λ over constrain the model, producing a fit far smoother than warranted by the data.

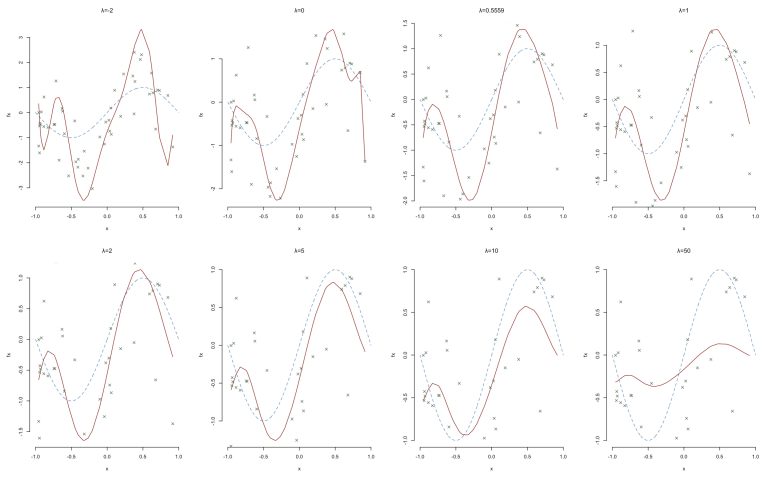

Figure 12: Fitting the model over various levels of λ to assess the effects of regularization.

Question 3

3.i. Scaling Images

Once the face images are vectorized they need to be scaled for efficient computation. The mean & standard deviation image - used for scaling - are given by figure 13. The mean image is indicative of features across all samples, whilst the standard deviation image is indicative of variation across these mean pixels.

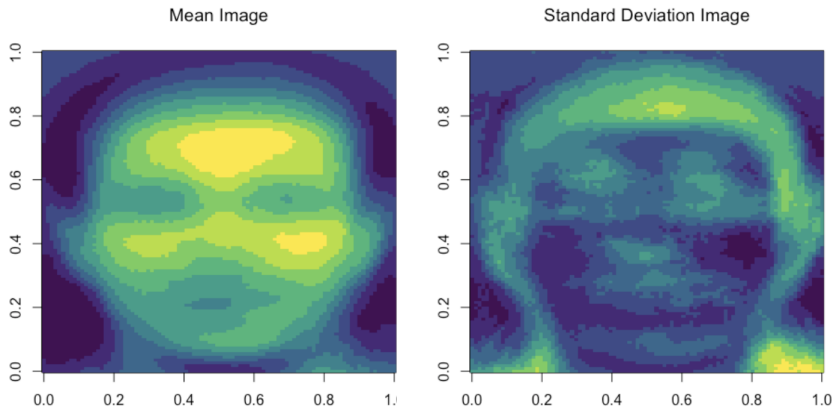

Figure 13: Mean & standard deviation images.

These mean & standard deviation images are used to scale the data. Figure 14 gives image 168 before & after scaling.

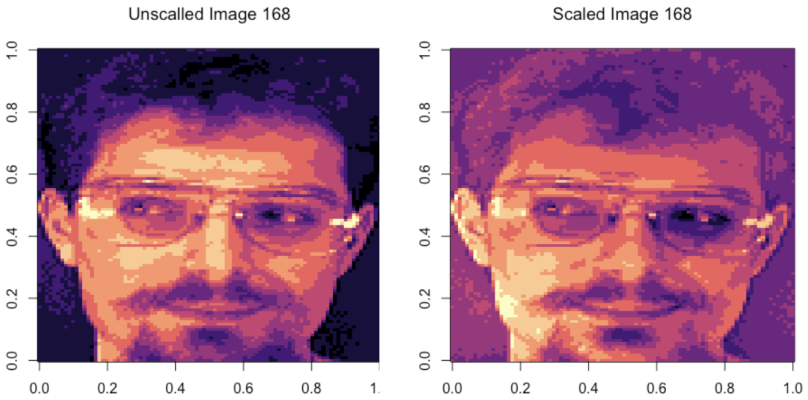

Figure 14: Image 168 before & after scaling.

3.ii. Eigen Faces

Using the data, we now visualize the first 10 eigen faces - the eigen vectors associated with the faces dataset - as seen in figure 15. These vectors capture the most variation in the data - thus are best able to distinguish the discrepancies between different classes (faces) whilst significantly reducing the dimensionality of the dataset.

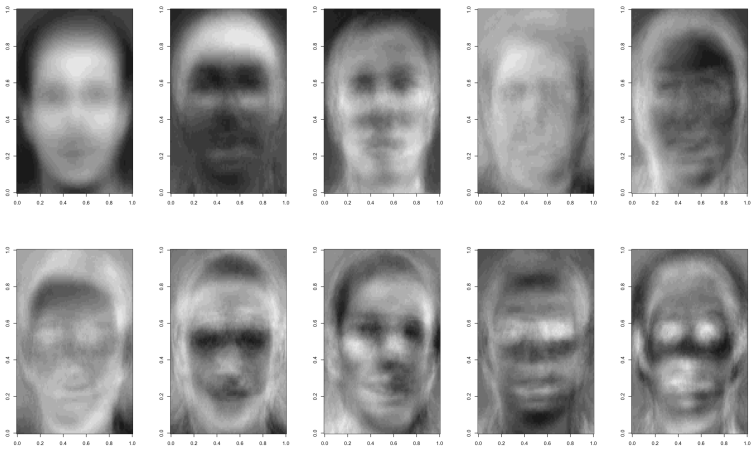

Figure 15: First 10 Eigen faces.

3.iii. Reconstruct an Image

Finally, we contrast the face of image 115 with the reconstructed images produced by $\{5, 50, 200\}$ eigen faces. As expected, the more eigen faces used the closer we approximate the true - high resolution - image.

A suitable trade-off between sufficient dimensionality reduction & capturing sufficient variation in the data ought to be achieved. One might consider the desired application when determining how much variation in the data is sufficient to achieve the desired task.

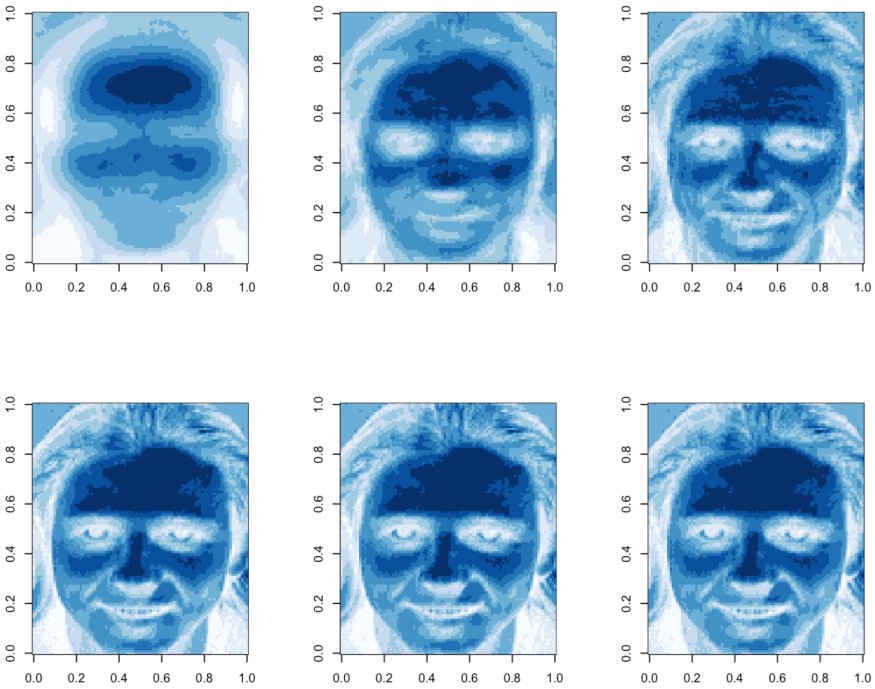

Figure 16: First 10 Eigen faces.

Appendix

Question 1 code

```
rm(list = ls())

# Question 1

# ---------- Question 1.i ----------x
x <- seq(-1, 1, length.out = 1000)
y <- 0.8*x
set.seed(24356)


generate_data <- function(ii=NA,N=30, sig=1) {
  x_samples <- sort(sample(x, size = N))
  y <- 0.8*x_samples + rnorm(n = N)
  return(cbind(x=x_samples, y=y))
}
data <- generate_data()

# ----- plot -----x
plot(x,y, 'l', main = 'Simulated_Data', frame.plot = F, col='#68bbaf', font.main=1)
points(data, col='darkred', pch=4)

# ----- model 1 -----x
g1 <- lm(y ~ 0 + x, offset = rep(0.5, nrow(data)), data = as.data.frame(data))
yhat_1 <- predict(object = g1, newdata = data.frame(x))

# ----- model 2 -----x
g2 <- lm(y ~ 0 + x, offset = rep(-0.5, nrow(data)), data = as.data.frame(data))
yhat_2 <- predict(object = g2, newdata = data.frame(x))

# ----- add to plot -----x
lines(x, yhat_1, col='#2685c4', lty=2)
lines(x, yhat_2, col='#800000', lty=2)

# ----- legend -----x
legend('topleft', legend = c('y~0.8x', 'g1(x)', 'g2(x)'),
       lty=c(1,2,2), col = c('#68bbaf', '#2685c4', '#800000'), box.lwd = 0.01)


# ---------- Question 1.ii ----------x

# ----- create datasets -----x
N <- 30
split_range <- 5:25
datasets <- lapply(matrix(1:10000), generate_data, N=N, sig=1)


# for each dataset
results <- c()
for (data in datasets) {


  for (i in split_range) {
    val_index <- sample(1:N, i, replace = F)
    X_val <- data[val_index,1]
    Y_val <- data[val_index,2]
    X_train <- data[-val_index,1]
    Y_train <- data[-val_index,2]

    # ----- model 1 -----x
    g1 <- lm(Y_train ~ 0 + X_train, offset = rep(0.5, length(X_train)))
    ypred_g1 <- predict(object = g1, newdata = data.frame(X_train=X_val))


    # ----- model 2 -----x
    g2 <- lm(Y_train ~ 0 + X_train, offset = rep(-0.5, length(X_train)))
    ypred_g2 <- predict(object = g2, newdata = data.frame(X_train=X_val))

    # ----- SSE: E_in -----x
    E_in_g1 <- sum(g1$residuals^2) / length(X_train)
    E_in_g2 <- sum(g2$residuals^2) / length(X_train)

    # ----- E_val -----x
    E_val_g1 <- sum((ypred_g1 - Y_val)^2) / length(X_val)
    E_val_g2 <- sum((ypred_g2 - Y_val)^2) / length(X_val)

    # ----- functions g(x) -----x
    g1x <- function(xx) rep(0.5, length(xx)) + g1$coefficients*xx
    g2x <- function(xx) rep(-0.5, length(xx)) + g2$coefficients*xx

    #g1x <- function(xx) predict(object = g1, newdata = data.frame(X_train=xx))
    #g2x <- function(xx) predict(object = g2, newdata = data.frame(X_train=xx))
```

```
    # ----- select g*(x) ------x
    ifelse(E_val_g1 < E_val_g2,
           {g_star <- g1; E_val_gstar <- E_val_g1; E_in_gstar <- E_in_g1; g_starx <-
                g1x; b0 <-  0.5; b1 <- g1$coefficients},
           {g_star <- g2; E_val_gstar <- E_val_g2; E_in_gstar <- E_in_g2; g_starx <-
                g2x; b0 <- -0.5; b1 <- g2$coefficients})

    # ----- E_out ------x
    # error_gstar <- function(xx) ( 0.8*xx - g_starx(xx))^2
    # E_out_gstar <- integrate(f = error_gstar, lower = -1, upper = 1)$value + (sig
        ^2)
    # E_out_gstar <- integrate(f = error_gstar, lower = -1, upper = 1)$value + (1^2)

    # ----- store results ------
    results <- rbind(results, cbind(i, E_in_gstar, E_val_gstar, b0, b1))
  }
}


# ----- Expected error curves ------x
# plot(data, pch=4, col='red')
# lines(x,y, 'l', lty=2)
# points(X_val, Y_val, col='blue')
# points(X_train, Y_train, col='red')
# lines(x, predict(g1, newdata = data.frame(X_train=x)), col='red')
# lines(seq(-1,1,length.out = 100), 0.5 + g1$coefficients*seq(-1,1,length.out = 100)
    , col='red')
# lines(seq(-1,1,length.out = 100), -0.5 + g2$coefficients*seq(-1,1,length.out =
    100), col='red')
# lines(x, predict(g2, newdata = data.frame(X_train=x)), col='blue')




# for each split
# ave:  b0, b1
agg_b0 <- c()
agg_b1 <- c()
E_vals <- c()
E_ins  <- c()
for (i in unique(results[,1])) {
  sub     <- results[results[,1]==i,]
  agg_b0  <- c(agg_b0, mean(sub[,4]))
  agg_b1  <- c(agg_b1, mean(sub[,5]))
  E_vals  <- rbind(E_vals, cbind(i, mean(sub[,3])))
  E_ins   <- rbind(E_ins, cbind(i, mean(sub[,2])))
}


# ------ compute u(x): average model ------x
ux <- function(xx) mean(agg_b0) + xx*mean(agg_b1)

# ------ visualize u(x) ------x
plot(data, pch=4, col='#407294', frame=F, main='u(x) Average Model', font.main=1)
lines(x,y, 'l', lty=2, col='black')
lines(x, ux(x), col='#c6939b')
legend('topleft', legend = c('f(x): target', 'ux:  ave_model'),
       lty=c(2,1), col = c('black', '#c6939b'), box.lwd = 0.1)


# ------ Compute Bias: bias^2 ------x
bias_func <- function(xx) (0.8*xx - ux(xx))^2
bias      <- integrate(f = bias_func, lower = -1, upper = 1)$value



# for each split

var  <- c()
for (i in 1:nrow(results)) {
  gx    <- function(xx) results[i,4] + results[i,5]*xx
  diff  <- function(xx) (gx(xx) - ux(xx))^2
  var   <- rbind(var, cbind(results[i,1], integrate(f=diff, lower = -1, upper = 1)$
       value))
}



# for each split
vars <- c()
for (i in unique(var[,1])) {
  sub  <- var[var[,1]==i,]
```

```
  vars <- rbind(vars, colMeans(sub))
}
# plot(vars[,1], vars[,2])


# ----- Out Sample Error -----x
E_out_g <- bias + vars[,2] + 1
E_vals


par(mfrow=c(1,1))
plot(5:25, E_out_g, 'l', col='#68bbaf',
     ylim = c(0.8, 2.),
     main='Errors over K validation split', frame.plot = F,
     font.main = 1, xlab='K: validation set size', ylab = 'Expected Error')
lines(E_vals[,1], E_vals[,2], 'l', col='#853155')
legend('topleft', legend = c('E_out[g*(x)]', 'E_val[g*(x)]'),
       col = c('#68bbaf', '#853155'), lty=1, box.lwd = 0.1)



# ----- In Sample Errors -----x
# lines(5:25, E_ins[,2], col='red')




# ------------------------------------------------- Additional Experiment: Unbiased Model
    ------------------------------------------------------x


# ----- create datasets -----x
N <- 30
split_range <- 5:25
datasets <- lapply(matrix(1:10000), generate_data, N=N, sig=1)


# for each dataset
results <- c()
for (data in datasets) {


  for (i in split_range) {
    val_index <- sample(1:N, i, replace = F)
    X_val <- data[val_index,1]
    Y_val <- data[val_index,2]
    X_train <- data[-val_index,1]
    Y_train <- data[-val_index,2]

    # ----- model 1 -----x
    g1 <- lm(Y_train ~ X_train)
    ypred <- predict(g1, newdata = data.frame(X_train=X_val))

    # ----- SSE: E_in -----x
    E_in_g1 <- sum(g1$residuals^2) / length(X_train)

    # ----- E_val-----x
    E_val_g1 <- sum((ypred - Y_val)^2) / length(X_val)


    # ----- functions g(x) -----x
    g1x <- function(xx) predict(object = g1, newdata = data.frame(X_train=xx))


    # ----- select g*(x) -----x
    g_star <- g1; E_val_gstar <- E_val_g1; E_in_gstar <- E_in_g1; g_starx <- g1x;
    b0 <-  g1$coefficients[1]; b2 <- g1$coefficients[2]

    # ----- store results -----
    results <- rbind(results, cbind(i, E_in_gstar, E_val_gstar, b0, b1))
  }
}




# for each split
# ave:  b0, b1
agg_b0 <- c()
agg_b1 <- c()
E_vals <- c()
E_ins  <- c()
for (i in unique(results[,1])) {
  sub     <- results[results[,1]==i,]
```

```
  agg_b0  <- c(agg_b0, mean(sub[,4]))
  agg_b1  <- c(agg_b1, mean(sub[,5]))
  E_vals  <- rbind(E_vals, cbind(i, mean(sub[,3])))
  E_ins   <- rbind(E_ins, cbind(i, mean(sub[,2])))
}


# ------ compute u(x): average model ------x
ux <- function(xx) mean(agg_b0) + xx*mean(agg_b1)

# ------ visualize u(x) ------x
plot(data, pch=4, col='#407294', frame=F, main='u(x) Average Model', font.main=1)
lines(x,y, 'l', lty=2, col='black')
lines(x, ux(x), col='#c6939b')
legend('topleft', legend = c('f(x): target', 'ux: ave_model'),
       lty=c(2,1), col = c('black', '#c6939b'), box.lwd = 0.1)



# ------ Compute Bias: bias^2 ------x
bias_func <- function(xx) (0.8*xx - ux(xx))^2
bias      <- integrate(f = bias_func, lower = -1, upper = 1)$value



# for each split

var  <- c()
for (i in 1:nrow(results)) {
  gx    <- function(xx) results[i,4] + results[i,5]*xx
  diff  <- function(xx) (gx(xx) - ux(xx))^2
  var   <- rbind(var, cbind(results[i,1], integrate(f=diff, lower = -1, upper = 1)$
       value))
}



# for each split
vars <- c()
for (i in unique(var[,1])) {
  sub  <- var[var[,1]==i,]
  vars <- rbind(vars, colMeans(sub))
}
# plot(vars[,1], vars[,2])


# ----- Out Sample Error -----x
E_out_g <- bias + vars[,2] + 1
E_vals


par(mfrow=c(1,1))
plot(5:25, E_out_g, 'l', col='#407294',
     ylim = c(0.8, 2.),
     main='Additional Experiment: y ~ a + bx', frame.plot = F,
     font.main = 1, xlab='K: validation set size', ylab = 'Expected Error', lty=2)
lines(E_vals[,1], E_vals[,2], 'l', col='#c16868', lty=2)
legend('topleft', legend = c('E_out[g*(x)]', 'E_val[g*(x)]'),
       col = c('#407294', '#c16868'), lty=2, box.lwd = 0.1)
```

Question 2 code

```
rm(list = ls())
# ----- simulate data ----x
simulate_data <- function(x_min, x_max, N=50) {
  x <- runif(N, min = x_min, max = x_max)
  x <- sort(x)
  y <- sin(pi*x) + rnorm(n=N,mean = 0,sd = 1)
  return(cbind(x,y))
}

# ----- generate data ----x
x <- seq(-1, 1, length.out = 1000)
fx <- sin(pi*x)
data <- simulate_data(x_min = -1, x_max = 1, N=50)

# ----- visualize data: Question 2.i ----x
par(mfrow=c(1,1))
plot(x, fx, frame=F, 'l', main='sin( x )', font.main=1, col='steelblue')
points(data[,1], data[,2], col='#800000')
legend('topleft', legend = c('target', 'data'), col=c('steelblue', '#800000'), lty =
      c(1, NA),pch = c(NA, 1), box.lwd = .1)



# ----- Question 2.ii ----x

# ----- Legendre polynomials ----x
Legendre <- function(x,n) {
  val=0
  for(i in 0:n) {
    val=val+((x^i)*choose(n,i)*choose((n+i-1)/2,n))
  }
  return((2^n)*val)
}


# ----- Question 2.i ----x
Z <- basis_funcs <- apply(matrix(0:10), MARGIN = 1, FUN = Legendre, x=data[,1])

# ----- closed form expression: with Regularization ----x
lambda <- 0
yhat <- Z %*% solve((t(Z) %*% Z) + lambda*diag(nrow(t(Z)))) %*% t(Z) %*% data[,2]
lambda <- 5
yhat2 <- Z %*% solve((t(Z) %*% Z) + lambda*diag(nrow(t(Z)))) %*% t(Z) %*% data[,2]

# ----- visualize data ----x
col1='#061db8'; col2='#810000'
par(mfrow=c(1,1))
plot(x, fx, frame=F, 'l', main=paste('q=', 10, sep = ''), font.main=1, col='
     steelblue', lty=2,
     ylim=c(min(yhat, yhat2, fx), max(yhat, yhat2, fx)))
points(data[,1], data[,2], col='#566d54', pch=4)
lines(data[,1], yhat, 'l', col=col1)
lines(data[,1], yhat2, 'l', col=col2)
legend('topleft',
       legend = c('target', 'data', '  :0', '  :5'), cex = 0.8,
       col=c('steelblue', '#566d54', col1, col2), lty = c(2,NA,1,1),pch = c(NA,4,NA,
            NA), box.lwd = .1)




# ----- Question 2.i: Many Qf ----x
compare_fits <- function(qq=0:9, lambdas=c(0,5), col1='#061db8', col2='#810000') {

  n <- length(qq)
  par(mfrow=c(2,n/2))
  for (q in qq) {
    Z <- basis_funcs <- apply(matrix(0:q), MARGIN = 1, FUN = Legendre, x=data[,1])

    # ----- closed form expression: with Regularization ----x
    lambda <- 0
    yhat <- Z %*% solve((t(Z) %*% Z) + lambda*diag(nrow(t(Z)))) %*% t(Z) %*% data
         [,2]
    lambda <- 5
    yhat2 <- Z %*% solve((t(Z) %*% Z) + lambda*diag(nrow(t(Z)))) %*% t(Z) %*% data
         [,2]


    # ----- visualize data ----x
    plot(x, fx, frame=F, 'l', main=paste('q=', q, sep = ''), font.main=1, col='
         steelblue', lty=2,
         ylim=c(min(yhat, yhat2, fx), max(yhat, yhat2, fx)))
    points(data[,1], data[,2], col='#566d54', pch=4)
    lines(data[,1], yhat, 'l', col=col1)
    lines(data[,1], yhat2, 'l', col=col2)
```

```
    legend('topleft',
           legend = c('target', 'data', '  :0', '  :5'), cex = 0.6,
           col=c('steelblue', '#566d54', col1, col2), lty = c(2,NA,1,1),pch = c(NA
                ,4,NA,NA), box.lwd = .1)
  }
}
compare_fits(qq=0:3)
compare_fits(qq=4:7)
compare_fits(qq=8:13)



# ---------------------------- Question 2.iii ------------------------------x

lambdas <- (seq(0.1, 10, length.out = 1000))


# 1 run of 10 fold
lambda <- 1
q <- 10
V <- 10
n <- 50
v_size <- n/V



results      <- c()
for (lambda in lambdas) {
  errors       <- c()
  for (v in 1:V) {

    # ----- split data -----x
    slice      <- (n/V*v-v_size):(n/V*v)
    val_set    <- data[slice,]
    train_set <- data[-slice,]

    # ----- fit model -----x
    Z          <- apply(matrix(0:q), MARGIN = 1, FUN = Legendre, x=train_set[,1])
    betas      <- solve((t(Z) %*% Z) + lambda*diag(nrow(t(Z)))) %*% t(Z) %*% train_set
        [,2]
    yhat       <- Z %*% betas

    # ----- prediction on val set -----x
    Z_val      <- apply(matrix(0:q), MARGIN = 1, FUN = Legendre, x=val_set[,1])
    yhat_val <- Z_val %*% betas

    # ----- val error -----x
    err_val  <- mean((yhat_val - val_set[,2])^2)
    errors    <- c(errors, err_val)
  }
  # ----- validation error -----x
  E_val       <- mean(errors)
  results     <- rbind(results, cbind(E_val, lambda))
}




# ----- min lambda -----x
min_index <- which(results[,1]==min(results[,1]))
lam <- results[min_index,2]

# ----- visualize optimial lambda -----x
par(mfrow=c(1,1))
plot(results[,2], results[,1], 'l', frame=FALSE, col='#66cccc', main = 'optimal_  ',
      font.main=1,
     ylab='Error', xlab='  ')
abline(h=results[min_index,1], lty=2, col='#95497e')
legend('topleft',
       legend = c('E_val(  )', paste('  =', round(lam,3), sep='')),
       col=c('#66cccc', '#95497e'), lty = c(1,2), pch = c(NA,NA), box.lwd = .1)



# ----- fit again -----x
Z          <- apply(matrix(0:q), MARGIN = 1, FUN = Legendre, x=data[,1])
betas      <- solve((t(Z) %*% Z) + lam*diag(nrow(t(Z)))) %*% t(Z) %*% data[,2]
yhat       <- Z %*% betas

# ----- visualize data -----x
par(mfrow=c(1,1))
plot(x, fx, frame=F, 'l', main=paste('  =', round(lam,4), sep = ''),
     ylim=c(min(yhat, fx), max(yhat, fx)),
     font.main=1, col='steelblue', lty=2)
lines(data[,1], yhat, 'l', col='#810000')
points(data[,1], data[,2], col='#566d54', pch=4)
```

```
par(mfrow=c(2,4))
for (LL in c(-2,0,lam,1,2,5,10,50)) {
  Z        <- apply(matrix(0:q), MARGIN = 1, FUN = Legendre, x=data[,1])
  betas    <- solve((t(Z) %*% Z) + LL*diag(nrow(t(Z)))) %*% t(Z) %*% data[,2]
  yhat     <- Z %*% betas

  # ---- visualize data ----x

  plot(x, fx, frame=F, 'l', main=paste('  =', round(LL,4), sep = ''),
       ylim=c(min(yhat, fx), max(yhat, fx)),
       font.main=1, col='steelblue', lty=2)
  lines(data[,1], yhat, 'l', col='#810000')
  points(data[,1], data[,2], col='#566d54', pch=4)
}
```

Question 3 code

```
# ------ libraries ------x
setwd("~/Desktop/Machine_Learning/assignments/assignment_2")
library(pixmap)

# ------ import data ------x
faces          <- c()
faces_pixes    <- c()
faces_dir <- list.files('data/Faces')
for (dir in faces_dir) {
  face         <- read.pnm(paste('data/Faces/', dir, sep = ''))
  faces        <- c(faces, face)
  faces_pixes <- c(faces_pixes, getChannels(face))
}


# ------ 1 face ------x
# ------ hyper-parameters ------x
m     <- prod(face@size)
n     <- 400
xx    <- t(matrix(rev(face@grey), face@size[1], face@size[2]))
dims <- face@size

faces_dir <- list.files('data/Faces')
X     <- matrix(0,n,prod(face@size))
for (i in 1:n) {
  x      <- read.pnm(paste('data/Faces/', i, '.pgm', sep = ''))
  X[i,] <- rev(x@grey)
}

# -------- Plot Mean Image ---------x
library(viridis)
xm <- colMeans(X)
xm <- matrix(xm, dims[1], dims[2])
par(mfrow=c(1,1))
image(t(xm), main='Mean_Image', font.main=1, col=viridis::viridis(10)[1:10])


# -------- Plot Std Image ---------x
xs <- apply(X,2,sd)
xs <- matrix(xs, dims[1], dims[2])
par(mfrow=c(1,1))
image(t(xs), main='Standard_Deviation_Image', font.main=1, col=viridis::viridis(10)
      [1:10])


# ------ Image 168 -------x
x <- X[168,]
x <- matrix(x, dims[1], dims[2])
image(t(x), main='Unscalled_Image_168', font.main=1, col=viridis::magma(10))
mean(x)
sd(x)

# ------ Scaled Image 168 -------x
xz <- X[168,]
xz <- (xz-colMeans(X))/apply(X,2,sd)
xz <- matrix(xz, dims[1], dims[2])
image(t(xz), main='Scaled_Image_168', font.main=1, col=viridis::magma(10))
mean(xz)
sd(xz)


# ------ scale data ------x
X     <- apply(X,2,scale)
x     <- X

# ------ Plot all the Faces ------x
cols <- colorRampPalette(c('grey90','grey50','grey10'))
windows()
par(mfrow = c(3,3))
for(j in 1:39)
{
  sss = 1:9 + j*10
  Sys.sleep(0.25)
  for( i in sss){
    image(t(matrix(X[i,],dims[1],dims[2])),col = cols(20),main = paste0('Individual_
          ',j))
    }
}


# ------------------ Question 2.ii ------------------x


# Calculate Eigenvalues:
N    = dim(x)[1]
SIG = 1/N*(x %*% t(x))
```

```
sv   = eigen(SIG)

# Construct components:
temp   = t(x) %*% sv$vectors
temp   = temp*matrix(1/sqrt(sv$values),dim(X)[2],dim(X)[1],byrow = T)

# windows()
par(mfrow = c(3,3))
for(i in 1:9)
{
  image(t(matrix(temp[,i],dims[1],dims[2])),col = cols(20))
}


# ------ Question 2.ii first 10 Eigen Faces ------x
par(mfrow = c(2,5))
for(i in 1:10)
{
  image(t(matrix(temp[,i],dims[1],dims[2])),col = cols(20))
}


#windows()
# ------ Question 2.iii: number of components? -------x
# How much information?
par(mfrow=c(1,1))
plot(cumsum(sv$values)/sum(sv$values),type = 'l')
abline(h = 0.95, lty = 3)

ss = c(10,50,100,200,350)
abline(v = ss,lty = 3)

ii = 36*10+1 #26

#windows()
# Compare construction to full image:
par(mfrow = c(2,5))
for(i in ss)
{
  Sys.sleep(0.01)
   # Yhat = ...
  Xhat = (x[ii,] %*% temp[,1:i]) %*% t(temp[,1:i])
  image(t(matrix(Xhat,dims[1],dims[2])),col = cols(20))
  Sys.sleep(0.01)
}
for( i in ss){
  image(t(matrix(X[ii,],dims[1],dims[2])),col = cols(20)) }


# Compare construction to full image:

ss <- c(5,50,200)
par(mfrow = c(2,3))
ii <- 115
for(i in ss)
{
  Sys.sleep(0.01)
  # Yhat = ...
  Xhat = (x[ii,] %*% temp[,1:i]) %*% t(temp[,1:i])
  image(t(matrix(Xhat,dims[1],dims[2])),col = blues9)
  Sys.sleep(0.01)
}
for( i in ss){
  image(t(matrix(X[ii,],dims[1],dims[2])),col = blues9) }
```