University of Cape Town

STA5090Z

Advanced Topics in Regression

Kernel Regression, Ridge, Lasso and Gibbs Sampling

Author:
Roger Bukuru

Student Number:
BKRROG001

August 11, 2024

Contents

1 Introduction

For this assignment we will be solving three problems. The first is a Kernel Regression Problem, the second includes Ridge and Lasso Regression and the third is a Markov Chain Monte Carlo Method (MCMC) problem.

2 Kernel Regression

2.1 Introduction and Aim

We start by exploring an application of Kernel Regression on monthly average temperature data at a particular location in the world. Our aim will be to:

- Use the Nadayara-Watson regression to show that the fitted values can be obtained using a linear filter in a particular form and as a result provide a smoother-matrix.
- Using the smoother-matrix we will provide a smoothed estimate of the temperature series assuming a Gaussian kernel.
- Finally we will use leave-one-out cross validation to obtain a suitable bandwidth parameter.

2.2 Nadayara-Watson Regression

We perform Kernel Regression using Nadayara-Watson regression, we start by showing that the fitted values ($\hat{y}$) obtained using Nadayara-Watson regression can be obtained using a linear filter of the form $\hat{y} = Sy$ where y is the observed data series and S is a *smoother* matrix.

Given a set of n observed data points $\{(x_i, y_i)\}_{i=1}^{n}$, the Nadaraya-Watson estimator for the value of y at a new point x is defined as [4]:

$$\hat{y}(x) = \frac{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) y_i}{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)} \tag{1}$$

Where:

- $K(\cdot)$ is the kernel function.
- h is the bandwidth parameter that controls the smoothness of the estimate.

To show that the fitted values $\hat{y}$ can be obtained using a linear filter, we express the estimator in matrix form. Let $K_{ij} = K\left(\frac{x_j - x_i}{h}\right)$. Then:

$$\hat{y}(x_j) = \sum_{i=1}^{n} w_{ij} y_i \tag{2}$$

where

$$w_{ij} = \frac{K\left(\frac{x_j - x_i}{h}\right)}{\sum_{k=1}^{n} K\left(\frac{x_j - x_k}{h}\right)} \tag{3}$$

Define the smoother matrix S with elements $S_{ji} = w_{ij}$. Then the fitted values can be written as:

$$\hat{y} = Sy \tag{4}$$

where S is the smoother matrix containing the weights w_{ij}.

2.3 Smoothing with a Gaussian Kernel

In order to perform Nadaraya-Watson regression we need to define a kernel, for this task we used a Gaussian kernel with bandwitdth h of the form:

$$K(x) = \frac{1}{\sqrt{2\pi h^2}} e^{-0.5(x/h)^2}. \tag{5}$$

substituting this kernel into equation 1 we obtained fitted values of the form:

$$\hat{y}(x) = \frac{\sum_{i=1}^{n} e^{-0.5\left(\frac{x-x_i}{h}\right)^2} y_i}{\sum_{i=1}^{n} e^{-0.5\left(\frac{x-x_i}{h}\right)^2}} \tag{6}$$

Using equation 6 we performed leave-one-out cross validation over the range $1 \leq h \leq 30$ in order to obtain the optimal bandwidth parameter, in figure 1, we observe that the optimal bandwidth was achieved at $h = 1$

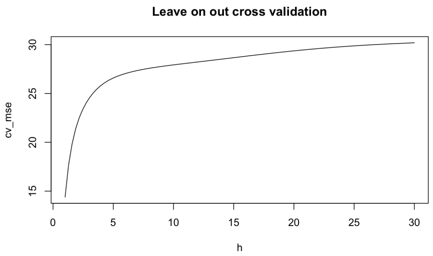

Figure 1: Leave on out cross validation

We proceeded to use this optimal parameter in order to calculate a smoothed estimate of the temperature and obtained the following results:

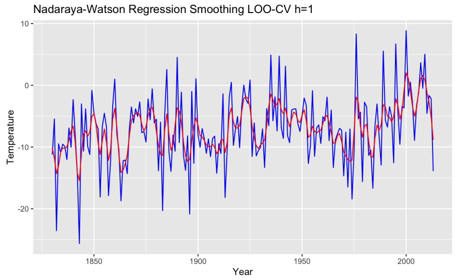

Figure 2: Smoothed Temperature Estimate $h = 1$

In figure 2 we observe that the smoother managed to capture the pattern of the data quite well whilst avoiding overfitting. This is important because the smaller the bandwidth parameter becomes, the more at risk we are of overfitting, in figure 3 we show that if we set $h = 0.1$ we end up overfitting.

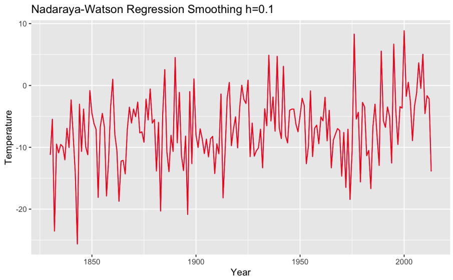

Figure 3: Overfitting for $h = 0.1$

3 Lasso and Ridge Regression

3.1 Introduction and Aim

In this section we will aim to minimize an objective function that represents the elastic net, that is an objective function that has both the Lasso and Ridge penalties, our aim will be to:

- Utilize the gradient descent optimization method to independently determine the optimal tuning parameters values for both the Lasso and Ridge penalties. We execute this using two different leave-one-out cross validations.
- Once we have the optimal parameters we compute the respective regression coefficients for each model and then compare these.
- Finally from the lasso model we analyse which regression coefficients are statistically different from zero.

We will be making use of the prostate data set to perform our analysis and we only use the in sample observations for all our computations.

3.2 Elastic Net: Lasso and Ridge Regression

As highlighted the Elastic net is a linear model that contains both the Lasso and Ridge penalties. We define this as [4]:

$$A = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \tag{7}$$

where λ_1 and λ_2 are tuning parameters ($\lambda_1 \geq 0$ and $\lambda_2 \geq 0$).

Our aim will be to minimize equation 7 such that we can obtain regression coefficients, furthermore we will assume that the columns of $\mathbf{X}$ and $\mathbf{y}$ have been scaled to have a mean of 0 and a variance of 1.

3.3 Gradient Descent Minimization

In order to minimize equation 7, we made use of the gradient descent method. Gradient descent is a technique used to estimate the regression coefficients in equation 7. The approach involves keeping all but one of the regression coefficients fixed and then updating the remaining coefficient. This process is repeated for each coefficient in turn to minimize A for a given set of λ_1 and λ_2.

To compute the regression coefficients we have to first derive an update equation for $\hat{\beta}_l$ for $l = 1, \ldots, p$ in terms of λ_1, λ_2, and $\hat{\beta}_{-l}$, where $\hat{\beta}_{-l}$ is the vector of regression parameters excluding $\hat{\beta}_l$.

3.3.1 Derivation of update equation

Below we follow a step-by-step approach to derive the update equation similar to the work by [2].

1. Compute the Partial Derivative $\frac{\partial A}{\partial \beta_l}$

The objective function in equation 7 has three components: the residual sum of squares, the L_1 regularization term (Lasso), and the L_2 regularization term (Ridge).

Residual Sum of Squares:

$$\frac{\partial}{\partial \beta_l} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 = -2 \sum_{i=1}^{n} x_{il} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right)$$

Lasso Regularization $\left(\lambda_1 \sum_{j=1}^{p} |\beta_j|\right)$:

The derivative of the Lasso term is:

$$\frac{\partial}{\partial \beta_l} \lambda_1 \sum_{j=1}^{p} |\beta_j| = \lambda_1 \operatorname{sign}(\beta_l)$$

where $\operatorname{sign}(\beta_l)$ is 1 if $\beta_l > 0$, -1 if $\beta_l < 0$, and 0 if $\beta_l = 0$.

Ridge Regularization $\left(\lambda_2 \sum_{j=1}^{p} \beta_j^2\right)$:

The derivative of the Ridge term is:

$$\frac{\partial}{\partial \beta_l} \lambda_2 \sum_{j=1}^{p} \beta_j^2 = 2\lambda_2\beta_l$$

3. Combine the Derivatives: We combine all the derivatives:

$$\frac{\partial A}{\partial \beta_l} = -2 \sum_{i=1}^{n} x_{il} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right) + \lambda_1 \operatorname{sign}(\beta_l) + 2\lambda_2\beta_l \qquad (8)$$

4. Update Equation The optimal value for a given β_l is given when equation 8 is minimized, while holding other coefficients fixed. This leads to the following optimality condition:

$$0 = -2\sum_{i=1}^{n} x_{il}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j) + \lambda_1 \text{sgn}(\beta_l) + 2\lambda_2\beta_l \tag{9}$$

using the following residual notation:

$$r_i = y_i - \sum_{j \neq l}^{p} x_{ij}\beta_j \tag{10}$$

we isolate the term containing the updated value $\beta_l^{(t+1)}$ to derive the update equation given by:

$$\beta_l^{(t+1)} = \frac{\sum_{i=1}^{n} x_{il}(y_i - x_{i,-l}^T\beta_{-l}) - \frac{\lambda_1}{2}\text{sgn}(\beta_l^{(t)}) - \lambda_2\beta_l^{(t)}}{\sum_{i=1}^{n} x_{il}^2} \tag{11}$$

where $\beta_l^{(t+1)}$ represents the β_l value at the $(t+1)$ iteration and $\beta_l^{(t)}$ at the t_{th} iteration.

3.4 Analysis of Regression Coefficients

We computed the regression coefficients, by minimizing the objective function shown in equation 7 with the update equation shown in equation 11. The goal was to compare the regression coefficients obtained when undertaking Lasso and Ridge regression.

This was achieved by using two different leave-one-cross validations to estimate appropriate tuning parameter values for the two regression models. That is for the Ridge regression we set the Lasso parameter to 0 and similarly for the Lasso we set the Ridge parameter to 0. The cross-validation was computed across 100 points for $\lambda_1 \in [0, 5]$ and $\lambda_2 \in [0, 5]$ and the update equation was computed across 20 iterations with a convergence criteria of $1e^{-04}$.

In figure 4 we show the results of the leave-one-out-cross (LooCV) validation for each of the models.

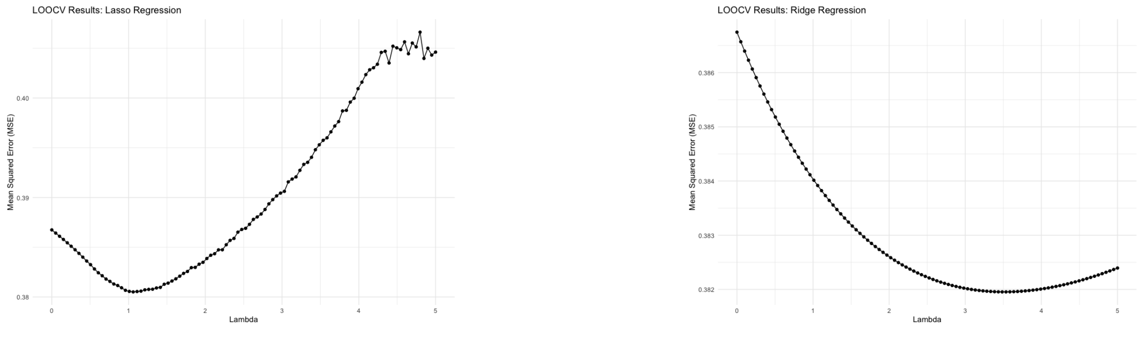

Figure 4: Lasso and Ridge LOOCV Results

We note that the optimal λ for the Lasso model was 1.0101 and the optimal λ for Ridge regression was 3.4343. This suggests that regularization was effective and that multicollinearity exists between the features as the λ parameter for Ridge was not zero, figure 5 below confirms this stance as for example we see that for the predictors gleason and pgg45 there exists high correlation of 0.76. Furthermore a non-zero value for Lasso indicates that certain predictors may not be important in predicting the response variable.

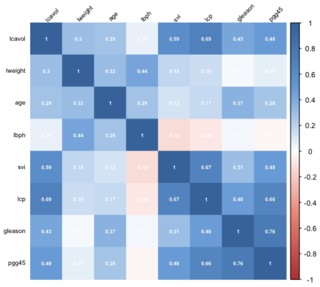

Figure 5: Predictor Correlation Analysis

We show the regression coefficients in table 1:

Predictor	**Lasso Coefficients**	**Ridge Coefficients**
lcavol	0.56923536	0.527830561
lweight	0.23815764	0.239068732
age	-0.10273133	-0.098759272
lbph	0.16689761	0.169576485
svi	0.24120637	0.241391108
lcp	-0.19010916	-0.159391458
gleason	-0.00630515	0.004474402
pgg45	0.19572855	0.184120639

Table 1: Coefficients for Lasso and Ridge Regression

Evaluating the coefficients above, we note that the coefficients are very similar with very minor differences. The relationship to the response variable is also consistent across the two models however the coefficient of the variable *gleason* is negative in the Lasso model whereby it's positive for the ridge model, indicating that this variable has opposite effects to the response variable between the two models.

3.5 Lasso Variable Selection

Our last objective in this section is to use the *boostrap* to test which Lasso regression coefficients are statistically different from zero. The boostrap was ran across 1000 iterations and therafter a 95% confidence interval was computed for each regression coefficient, the results are shown in table 2 below:

Predictor	**Lasso Coefficients**	**Lower**	**Upper**
lcavol	0.570041525	0.325635829	0.7635247034
lweight	0.237442453	0.057948344	0.4199194687
age	-0.100413985	-0.216830483	0.0509321832
lbph	0.167345205	0.004149456	0.3749541289
svi	0.239690611	0.007029293	0.4393423690
lcp	-0.190294295	-0.387586053	0.0004771296
gleason	-0.000223603	-0.207443099	0.1875128709
pgg45	0.191014756	0.014446424	0.4131771040

Table 2: Coefficients for Lasso and Ridge Regression

To determine which Lasso regression coefficient are statistically different from zero, we look for the confidence intervals that do not include zero. From table 2 we note that besides the coefficient associated with the predictor *age*, *lcp* and *gleason* all the other coefficients are statistically different from zero. Given the coefficient associated with the predictors *age*, *lcp* and *gleason* are not statistically significant from zero, these would be set to zero which is the result of the Lasso performing variable selection and in this case these variables would be excluded from predictions[4].

4 Markov Chain Monte Carlo: Gibbs Sampling

4.1 Introduction and Aim

For this final section we will be implementing a Markov Chain Monte Carlo Method (MCMC) technique in order to model the gametocyte density ($y_{i,j}$, for $i = 1, \ldots, n$, $j = 1, \ldots, 7$) present in patients over time from Distiller et al (2010). We will be working with 25 patient profile data where the gametocyte densities were measured on seven occasions namely, days 0, 3, 7, 14, 21, 28 and 42 (denoted as t_j).

One of the challenges we have is that on occasion a measurement ($y_{i,j}$) of 0 was recorded, in which case the observation was either removed from the analysis or set equal to a small positive value. Fill-in values were assigned in these instances. After this process the data was transformed to:

$$z_{i,j} = \log_2(1 + y_{i,j}) \text{ for all } i, j \tag{12}$$

Given this transformed data is a multivariate probability distribution which is not easily retractable our goal will be to conditionally model this patient data by undertaking the MCMC Gibbs sampling method.

4.2 Gibbs Sampling

Gibbs sampling is a techniques that involves sequentially update each variable x_i while keep all other variables fixed at their current values. This process forms a Markov Chain, and through repeated iterations the chain eventually converges to the joint probability distribution of all the variables, we will be undertaking this sampling with the transformed patient data.

In figure 6 we display the profiles ($z_{i,j}$) of such 25 such patients:

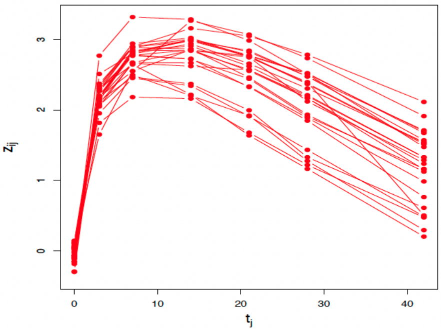

Figure 6: Patient Profiles

The transformed model described in equation 12, involves measurements taken from a patient as well as the mean profile of the $z_{i,j}$ measurement. More preciscely we assume that the probability density function of $z_{i,j}$ (for all i, j) is defined as

$$[z_{i,j}|\theta] = \mathcal{N}(f_i(t_j), \sigma_e^2) \tag{13}$$

where θ are parameters to be estimated. $f_i(t_j)$ is a non-linear function of time with

$$f_i(t_j) = \beta_{0,i} + \sum_{k=1}^{3} \tilde{\phi}_{k,i} \times b(|t_j - \xi_k|) \quad \text{for } j = 1, \ldots, 7$$

where $b(x) = x\log(x)$ and $\xi = [0.5, 10, 25]^T$.

Let

$$\phi_i = \left[\beta_{0,i}, \tilde{\phi}_{1,i}, \tilde{\phi}_{2,i}, \tilde{\phi}_{3,i}\right]^T = \begin{bmatrix}\beta_{0,i} \\ \tilde{\phi}_i\end{bmatrix}.$$

We assume that $\sigma_e^2 \sim IG(i_1, i_2)$ and $\sigma_\phi^2 \sim IG(i_3, i_4)$.

We want to sample from the multivariate distribution of $z_{i,j}$ described in equation 13, we will achieve this using a Gibbs sampler where we will start by first deriving the conditional distributions of and thereafter sample from the conditional distributions in turn [1].

4.2.1 Conditional distributions

The conditional distributions that we sampled from are as follow:

$$\sigma_e^2 | z, \phi_1, \ldots, \phi_{25}, \sigma_\phi^2 \tag{14}$$

$$\sigma_\phi^2 | z, \phi_1, \ldots, \phi_{25}, \sigma_\phi^2, \sigma_e^2 \tag{15}$$

$$\phi_i | z, \sigma_\phi^2, \sigma_e^2 \text{ for } i = 1, \ldots, 25. \tag{16}$$

We derived their distributions and this is presented below [1][3]:

a. Conditional Distribution for σ_e^2:

Given the inverse gamma prior $\sigma_e^2 \sim IG(i_1, i_2)$, the posterior distribution for σ_e^2 is:

$$\sigma_e^2 | z, \phi_1, \ldots, \phi_{25} \sim IG\left(i_1 + \frac{nT}{2}, i_2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{T}(z_{ij} - f_i(t_j))^2\right) \tag{17}$$

b. Conditional Distribution for σ_ϕ^2:

Given the inverse gamma prior $\sigma_\phi^2 \sim IG(i_3, i_4)$, the posterior distribution for σ_ϕ^2 is:

$$\sigma_\phi^2 | z, \sigma_e^2 \sim IG\left(i_3 + \frac{n(p+1)}{2}, i_4 + \frac{1}{2}\sum_{i=1}^{n} \phi_i^T \phi_i\right) \tag{18}$$

c. Conditional Distribution for ϕ_i:

The conditional distribution for ϕ_i is given by:

$$\phi_i | z, \sigma_e^2, \sigma_\phi^2 \sim \mathcal{N}(\mu_{\phi_i}, \Sigma_{\phi_i}) \tag{19}$$

where

$$\Sigma_{\phi_i} = \left(\frac{1}{\sigma_e^2} X_i^T X_i + \frac{1}{\sigma_\phi^2} I_{p+1}\right)^{-1}$$

and

$$\mu_{\phi_i} = \Sigma_{\phi_i}\left(\frac{1}{\sigma_e^2} X_i^T z_i\right).$$

Here, X_i is the design matrix for patient i.

Using equations 17, 18 and 19, we undertook Gibbs sampling by sampling from each conditional distribution in turn.

4.2.2 Sampling Results

We sampled by computing 30 000 iterations with a burn-in sample of 10 000 iterations, we ran one Markov chain and assumed the following

$$i_1 = 2, \quad i_2 = 0.1, \quad i_3 = 2, \quad \text{and} \quad i_4 = 0.001 \tag{20}$$

for equations 17 and 18. Furthermore n was 25 and T was 7. In the traceplots displayed in figure 7 below we see that both the σ_e^2 and σ_ϕ^2 conditional distributions reached convergence, as the trace plots are stationary. The ϕ_i also successfully converged across all the profiles, given there were too many plots to show here, some samples can be found in the appendix.

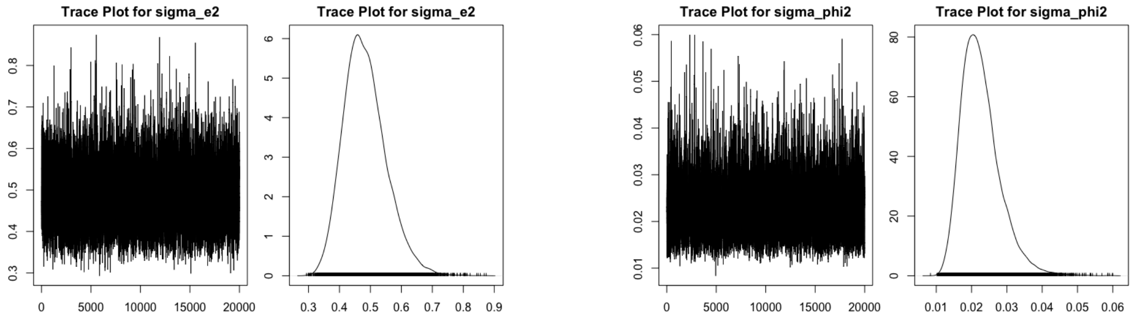

Figure 7: σ_e^2 and σ_ϕ^2 Trace Plots

4.3 Evaluating a patient profile

We plotted the 15^{th} observed profile and assessed how well it was fitted by our model by superimposing the posterior mean as well as the 95% credible interval of the fitted profile. Figure 8 shows that the fitted(predicted) profile closely follows the observed profile, which indicates that the model has captured the general trend of the patient's profile well.

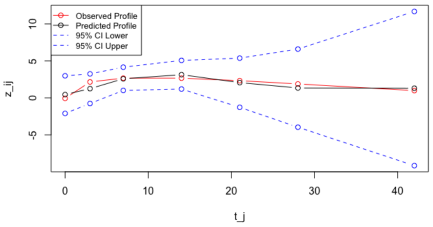

Figure 8: Credible Interval

In terms of the credibility intervals we see it vary over time, which indicates differing levels of uncertainty in the models predictions at different time points. We note that at earlier time points (0 to 10) the credibility intervals are relatively narrow, suggesting high confidence in the models predictions whereby as time progresses the credible intervals widen, indicating increased uncertainty in the model's prediction, this may be due to fewer data points, increased variability or even the model limitations at later time points. We do however note, that in this increased uncertainty band the fitted profile still follows the observed profile well.

References

[1] Prof Michael Jordan. Bayesian inference and gibbs sampling, 2010.

[2] Andrew Ng Tengyu Ma. Cs229 lecture notes, 2023.

[3] Christian P. Robert. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation.* Springer Series in Statistics. Springer Verlag, New York, 2007.

[4] Ryan Tibshirani. Kernel regression and local polynomial regression, 2023.

5 Appendix

5.1 Question 1:

5.1.1 b(i.)

```
library(ggplot2)
library(tidyverse)
library(dplyr)

filepath = "data/tempexam.csv"
temp_data = read.csv2(filepath)
summary(temp_data)

# Nadayara-Watson function
gaussian_kernel = function(x,h){
  return ((1/sqrt(2*pi*(h^2))) * exp(-0.5*(x/h)^2))
}

nw_estimator = function(x, observed_x, observed_y, h){
 weights = sapply(observed_x, function(xi) gaussian_kernel(abs(x-xi), h))
 weights = weights/sum(weights) # normalize weights
 y_hat = sum(weights * observed_y)
 return (y_hat)
}

x_train = as.matrix(temp_data$Dates)
y_train = as.matrix(as.numeric(temp_data$y))

y_hat = sapply(x_train, nw_estimator,observed_x = x_train,
observed_y = y_train, 0.5)
final_temp_data = data.frame(x= x_train, y = y_train, y_hat=y_hat)

# Plot Nadayara-Watson estimators against the original data
ggplot(final_temp_data, aes(x = x, y = y)) +
  geom_line(color = "blue") +
  geom_line(aes(y = y_hat), color = "red") +
  labs(title = "Nadaraya-Watson Regression Smoothing h=0.5",
       x = "Year",
       y = "Temperature")
```

5.1.2 b(ii.)

```
# Leave one-out-cross-validation

loo_cv = function(data,bandwidths){
 cv_grid = matrix(data=NA, nrow=nrow(data), ncol = length(bandwidths)+1)
 cv_error_grid = matrix(data=NA, nrow=nrow(data), ncol = length(bandwidths)+1)

 for (i in 1:nrow(data)){
   training_set = data[-i, ]
   validation_set = data[i, ]
   x_train = as.matrix(data$Dates)
   y_train = as.matrix(as.numeric(data$y))
   x_validation = as.matrix(validation_set$Dates)
   y_validation = as.matrix(as.numeric(validation_set$y))
   cv_grid[i,1] = x_validation
   cv_error_grid[i, 1] = x_validation

   for( j in 1:length(bandwidths)){
     h = bandwidths[j]
     y_hat = sapply(x_validation, nw_estimator,
     observed_x = x_train, observed_y = y_train, h)
     cv_grid[i,j+1] = y_hat
     cv_error_grid[i, j+1] = (y_validation - y_hat)^2
   }
 }
 colnames(cv_grid) = c("x", paste0("h", 1:length(bandwidths)+1))
 mse = apply(cv_error_grid[,-1], 2, mean)
 return (list(fitted_values = cv_grid, errors = cv_error_grid, mse=mse))
}

bandwidths = seq(1,30, length.out=101)

loo_cv_result = loo_cv(temp_data,bandwidths)
fitted_vals = loo_cv_result$fitted_values
cv_errors = loo_cv_result$errors
cv_mse = loo_cv_result$mse
plot(x= bandwidths, y=cv_mse, type = "l",
main="Leave on out cross validation", xlab="h")
optimal_bandwith = bandwidths[which.min(cv_mse)]

# Use optimal bandwith to plot estimate
y_hat = sapply(x_train, nw_estimator,observed_x = x_train,
observed_y = y_train, optimal_bandwith)
final_temp_data = data.frame(x= x_train, y = y_train, y_hat=y_hat)

# Plot Nadayara-Watson estimators against the original data
ggplot(final_temp_data, aes(x = x, y = y)) +
  geom_line(color = "blue") +
  geom_line(aes(y = y_hat), color = "red") +
  labs(title = "Nadaraya-Watson Regression Smoothing LOO-CV h=1",
       x = "Year",
       y = "Temperature")
```

6 Question 2:

6.0.1 b(i.)

```
regopt = function(tuning, Y, X, nitermax=20){
  lambda1 = tuning[1]
  lambda2 = tuning[2]

  p = ncol(X)
  beta  = rep(0, p)
  errortol = 1e-4

  A = NULL
  if(lambda1 == 0 && lambda2 == 0){
    # compute OLS Beta
    beta = ols_beta(Y,X)

  }else {
     for (i in 1:nitermax) {
       # Store previous beta for convergence check
       beta_prev = beta
       # Update each coefficient in turn
       for (l in 1:p){
          beta[l] = beta_gradient_desc(tuning, Y, X, beta,l)
       }
       # Check for convergence
       if( max(abs(beta - beta_prev)) < errortol){
         break
       }
     }
  }
  A = elastic_net_obj_function(tuning, Y,X, beta)
  return(list(errortol= errortol, beta = beta, A = A))

}

ols_beta = function(Y,X){
  beta_hat = solve(t(X)%*%X)%*%t(X)%*%Y
  return (beta_hat)
}

compute_A <- function(Y, X, beta, lambda1, lambda2) {
  residual_sum_squares <- sum((Y - X %*% beta)^2)
  lasso_penalty <- lambda1 * sum(abs(beta))
  ridge_penalty <- lambda2 * sum(beta^2)
  A <- residual_sum_squares + lasso_penalty + ridge_penalty
  return(A)
}
```

```
elastic_net_obj_function = function(tuning, Y,X, beta){
  lambda1 = tuning[1]
  lambda2 = tuning[2]
  one_transpose = t(matrix(1, nrow=length(beta), ncol=1))
  rss = t((Y-X%*%beta))%*%(Y-X%*%beta)
  lasso_penalty = lambda1 * one_transpose %*% abs(beta)
  ridge_penalty = lambda2 * t(beta)%*%beta
  A = rss + lasso_penalty + ridge_penalty
  return(A)
}

beta_gradient_desc = function(tuning, Y,X, beta, l){
  lambda1     = tuning[1]
  lambda2     = tuning[2]
  r           = Y - X[, -l]%*%beta[-l] # exclude effect of beta_l
  numerator   = sum(X[, l] * r) - (lambda1 / 2) * sign(beta[l]) - lambda2 * beta[l]
  denominator = sum(X[, l]^2)
  beta_new    = numerator / denominator
  return (beta_new)
}
```

6.0.2 b(ii.)

```
loo_cv = function(Y,X, lambdas, lasso_regression=TRUE){

  n = nrow(Y)
  p = ncol(X)
  # Setup up cross-validation grid matrix
  cv_grid = matrix(data=NA, nrow=nrow(X), ncol = length(lambdas))
  # Setup cross validation error grid matrix
  cv_error_grid = matrix(data=NA, nrow=nrow(X), ncol = length(lambdas))

  for( i in 1:n){
    # Leave one observation out
    x_training_set = X[-i,]
    x_validation_set = X[i,]
    y_training_set  = Y[-i,]
    y_validation_set = Y[i,]
    for (j in 1: length(lambdas)){
      result = NA
      lambda = lambdas[j]
      if(lasso_regression){
        tuning = c(lambda, 0)
        result = regopt(tuning, y_training_set, x_training_set, nitermax = 20)
      }else{
        tuning = c(0, lambda)
        result = regopt(tuning, y_training_set, x_training_set, nitermax = 20)
      }
       y_hat = x_validation_set%*%result$beta
       cv_grid[i,j] = y_hat
       cv_error_grid[i, j] = (y_validation_set - y_hat)^2
    }
  }
  colnames(cv_grid) = c(paste0("lambda", c(lambdas)))
  colnames(cv_error_grid) = c(paste0("lambda", c(lambdas)))
  mse = apply(cv_error_grid, 2, mean)
  return (list(fitted_values = cv_grid, errors = cv_error_grid, mse=mse))
}

# Lambda range [0,5] across 100 points
#lambdas = 10^seq(0,log10(5), length.out=100)
lambdas = seq(0,5, length.out=100)

# Cross-validation for Lasso Regression
lasso_loo_cv_result = loo_cv(Y_train,X_train,lambdas,lasso_regression=TRUE)
lasso_reg_fitted_vals = lasso_loo_cv_result$fitted_values
lasso_reg_errors = lasso_loo_cv_result$errors
lasso_reg_mse = lasso_loo_cv_result$mse
optimal_lasso_lambda = lambdas[which.min(lasso_reg_mse)]
optimal_lasso_lambda

# Lasso Coefficients
tuning = c(optimal_lasso_lambda,0 )
lasso_result <- regopt(tuning, Y_train, X_train, nitermax = 20)
lasso_coefficients = lasso_result$beta
lasso_coefficients
plot(x= lambdas, y=lasso_reg_mse, type = "l",
main="Lasso Regression LOOCV", xlab="lambda", ylab="CV MSE")

# Cross-validation for Ridge Regression
ridge_loo_cv_result = loo_cv(Y_train,X_train,lambdas,lasso_regression=FALSE)
ridge_reg_fitted_vals = ridge_loo_cv_result$fitted_values
ridge_reg_errors = ridge_loo_cv_result$errors
ridge_reg_mse = ridge_loo_cv_result$mse
optimal_ridge_lambda = lambdas[which.min(ridge_reg_mse)]
optimal_ridge_lambda

# Ridge Coefficients
tuning = c(0,optimal_ridge_lambda )
ridge_result <- regopt(tuning, Y_train, X_train, nitermax = 20)
ridge_coefficients = ridge_result$beta
ridge_coefficients
plot(x= lambdas, y=ridge_reg_mse, type = "l",
main="Ridge Regression LOOCV", xlab="lambda", ylab="CV MSE")
```

6.0.3 b(iii.)

```
library(boot)
set.seed(123)
bootstrap_lasso_fit = function(data, indices){
 data = data[indices,]
 Y_train = as.matrix(data$lpsa)
 colnames(Y_train) = "lpsa"
 X_train = as.matrix(data[, -9])
 tuning = c(optimal_lasso_lambda,0 )
 lasso_result <- regopt(tuning, Y_train, X_train, nitermax = 20)
 lasso_coefficients = lasso_result$beta
 return(lasso_coefficients)
}

boot_results = boot(data = in_sample_obs_std,
statistic = bootstrap_lasso_fit, R = 1000)
bootstrap_estimates <- boot_results$t
boot_results


compute_ci <- function(bootstrap_estimates, level = 0.95) {
  ci_lower <- apply(bootstrap_estimates, 2, quantile,
  probs = (1 - level) / 2)
  ci_upper <- apply(bootstrap_estimates, 2, quantile,
  probs = 1 - (1 - level) / 2)
  return(data.frame(Lower = ci_lower, Upper = ci_upper))
}

# Compute 95% confidence intervals for each coefficient
confidence_intervals <- compute_ci(bootstrap_estimates)
print(confidence_intervals)
```

6.1 Question 3

6.1.1 Gibbs Sampling

```
library(MASS)
library(coda)

patient_profiles = read.csv2(file="data/profiles2024.csv", header=TRUE)
patient_profiles = as.matrix(patient_profiles[1:25,])
```

```
# Initial Settings
n = 25
T = 7
p = 3
iterations = 30000
burn_in = 10000


# Chain Starting values
sigma_e2 = 1
sigma_phi2 = 2
phi = matrix(0, nrow=n, ncol=4)

# prior values
i1 = 2
i2 = 0.1
i3 = 2
i4 = 0.001

# Store initial chain values

sigma_e2_chain   = numeric(iterations)
sigma_phi2_chain = numeric(iterations)
phi_chain        = array(0, dim=c(n, 4, iterations))
# b(x) design matrix
b = function (x) { x* log(x) }
eta_k = c(0.5, 10, 25)
t_j   = c(0,3,7,14,21,28,42)
X_i = matrix(0, nrow=T, ncol=4)
X_i[,1] = 1

for ( j in 1:T){
  X_i[j, 2:4] = b(abs(t_j[j]-eta_k))
}

#-------------------------- Perform Gibbs sampling------------------------------

for (k in 1:iterations){

  # Update sigma_e2
  patient_sum_of_squares = 0
  for ( i in 1:n){
    patient_sum_of_squares = patient_sum_of_squares +
    sum((patient_profiles[i, ]- X_i%*%phi[i,])^2)
  }
  sigma_e2 = 1/ rgamma(1, i1 + n*T/2, i2+ patient_sum_of_squares/2)

  # Update sigma_phi2
  sum_phi_squares = sum(phi^2)
  sigma_phi2 = 1 / rgamma(1, i3 + n * (p+1)/ 2, i4 + sum_phi_squares)

  # update phi_i
  for ( i in 1:n ){
    Sigma_phi <- solve((1/sigma_e2) * t(X_i) %*% X_i + diag(1/sigma_phi2, 4))
    mu_phi <- Sigma_phi %*% ((1/sigma_e2) * t(X_i) %*% patient_profiles[i, ])
    phi[i, ] <- mvrnorm(1, mu_phi, Sigma_phi)
  }
  # Store Markov Chain Draws
  sigma_e2_chain[k] <- sigma_e2
  sigma_phi2_chain[k] <- sigma_phi2
  phi_chain[, , k] <- phi
}

# ------------------ Trace Plots and Convergence Statistics --------------------
# Remove burn-in samples
sigma_e2_chain <- sigma_e2_chain[(burn_in+1):iterations]
sigma_phi2_chain <- sigma_phi2_chain[(burn_in+1):iterations]
phi_chain <- phi_chain[, , (burn_in+1):iterations]

# Convergence diagnostics
par(mfrow = c(2, 2), mar = c(2, 2, 2, 1))
plot(as.mcmc(sigma_e2_chain), main = "Trace Plot for sigma_e2",
xlab = "Iteration", ylab = "Value")
plot(as.mcmc(sigma_phi2_chain), main = "Trace Plot for sigma_phi2",
xlab = "Iteration", ylab = "Value")


# Convergence diagnostics for phi parameters
par(mfrow = c(25, 4), mar = c(2, 2, 2, 1), oma = c(4, 4, 2, 1), cex.main = 0.7)
for (i in 1:25) {
  for (j in 1:4) {
    plot(as.mcmc(phi_chain[i, j, ]),
         main = paste("phi[", i, ",", j, "]", sep = ""),
         xlab = "Iteration",
         ylab = "Value",
         col.main = "blue",
         cex.main = 0.7)
  }
}
par(mfrow = c(1, 1))  # Reset to default



# Extract posterior means and 95% credible intervals for patient 15
posterior_means <- apply(phi_chain[15, , ], 1, mean)
credible_intervals <- apply(phi_chain[15, , ], 1,
```

```r
quantile, probs = c(0.025, 0.975))

#------------------- Analyses of 15th Observed Profile -------------------------

# Plot the 15th observed profile
observed_profile <- patient_profiles[15, ]
predicted_profile <- X_i %*% posterior_means
predicted_profile_lower <- X_i %*% credible_intervals[1, ]
predicted_profile_upper <- X_i %*% credible_intervals[2, ]

plot(t_j, observed_profile, type = 'b', col = 'red',
ylim = c(min(observed_profile, predicted_profile_lower), max(observed_profile,
predicted_profile_upper)),
ylab = 'z_ij', xlab = 't_j')
lines(t_j, predicted_profile, type = 'b', col = 'black')
lines(t_j, predicted_profile_lower, type = 'b', col = 'blue', lty = 2)
lines(t_j, predicted_profile_upper, type = 'b', col = 'blue', lty = 2)
legend("topleft", legend = c("Observed Profile", "Predicted Profile",
"95% CI Lower",
"95% CI Upper"), col = c("red", "black", "blue", "blue"),
lty = c(1, 1, 2, 2), pch = c(1, 1, NA, NA), pt.cex = 1, cex = 0.8)
```

6.1.2 ϕ_i Sample Trace Plots

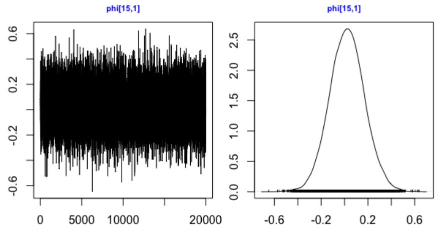

Figure 9: 15^{th} profile $\phi_i = 1$ Trace Plot

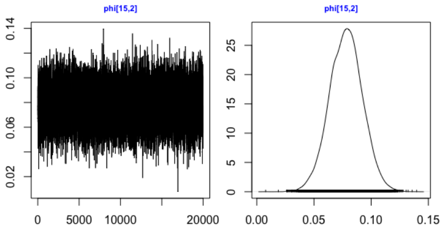

Figure 10: 15^{th} profile $\phi_i = 2$ Trace Plot

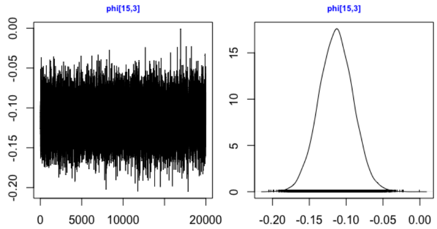

Figure 11: 15^{th} profile $\phi_i = 3$ Trace Plot

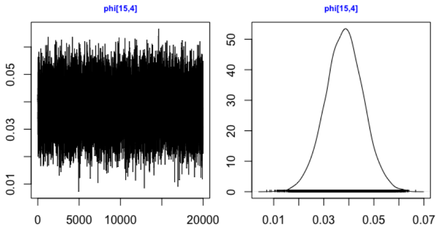

Figure 12: 15^{th} profile $\phi_i = 4$ Trace Plot

Figure 13: 15^{th} profile ϕ_i Trace Plots