MATH 297 Assignment 2

Zongze Liu

06 March 2022

We first import important libraries and the data. We also define the important constants.

```
> library(tidyverse)
> require(lspline)
> require(splines)
> require(MASS)
> library(HoRM)
> tmp = read_csv("C:/Users/zongze liu/Desktop/tmp-20160413.csv")
> tmp = as.data.frame(tmp)
> names(tmp) = c("xx", "yy")
> dat = tmp
> attach(dat)
> n = length(xx)
> k = round(n/log(n))
> B = 200
```

1 Test for the best model

The idea is that we will estimate the risks for 100 sets of pseudo data for all four models with optimal parameters and construct a hypothesis testing to compare the risks of each pair of the models. The null hypothesis is that the risk of model A is equal to the risk of model B. If we can reject the null hypothesis then we can decide which model is better.
We want to first generate 100 sets of pseudodata. We fit a big linear spline model with high degrees of freedom and use the residue sum of squares of the big model to estimate the variance of the pseudo data (or parametric bootstrap).

```
> big_model = lm(yy ~ elspline(xx, k), data = dat)
> y_hat = fitted(big_model)
> estimated_var = sqrt(sum(resid(big_model)^2)/(n - k))
> pseudodata = matrix(0, nrow = n, ncol = 100)
> for(j in 1:100){
+   pseudodata[,j] = y_hat + rnorm(n, 0, estimated_var)
+ }
```

we consider the regressogram model with optimal number of bins = 42 from assignment1.We want to calculate the estimated risk for each set of pseudodata using parametric bootstrap as in Elfron's paper.

```
> regressogram_residue = function(x, y, nbins){
+   xy <- data.frame(x = x,y = y)
+   xy <- xy[order(xy$x),]
+   z <- cut(xy$x,breaks=seq(min(xy$x),max(xy$x),length=nbins+1),
+            labels=1:nbins,include.lowest=TRUE)
+   xyz <- data.frame(xy,z=z)
+   MEANS <- c(by(xyz$y,xyz$z,FUN=mean))
+   residue = 0
+   for (i in 1:n){
+     residue = residue + (xyz$y[i] - MEANS[xyz$z[i]])^2
+   }
+   return(residue)
+ }
> regressogram_fit = function(x, y, nbins){
+   xy <- data.frame(x = x,y = y)
+   xy <- xy[order(xy$x),]
+   z <- cut(xy$x,breaks=seq(min(xy$x),max(xy$x),length=nbins+1),
+            labels=1:nbins,include.lowest=TRUE)
+   xyz <- data.frame(xy,z=z)
+   MEANS <- c(by(xyz$y,xyz$z,FUN=mean))
+   return(MEANS[z])
+ }
> optimal_bins = 42
> regressogram_estimated_risk = rep(0, times = 100)
> for(j in 1:100){
+   curdata = pseudodata[,j]
+   cov_hat = rep(0, times = n)
+   mu_hat_star = matrix(0, nrow = n, ncol = B)
+   bootstrap = matrix(0, nrow = n, ncol = B)
+   bootstrap_data = data.frame(x = xx, y = curdata)
+   big_model_boot = lm(y ~ elspline(x, k), data = bootstrap_data)
+   y_hat_boot = fitted(big_model_boot)
+   sqaure_err = regressogram_residue(xx, curdata, optimal_bins)
+
+   for(b in 1:B){
+     bootstrap[,b] = y_hat_boot + rnorm(n, 0, estimated_var)
+     boot_dat = data.frame(x = xx, y = bootstrap[,b])
+     mu_hat_star[,b] = regressogram_fit(boot_dat$x, boot_dat$y, optimal_bins)
+   }
+
+   for(i in 1:n){
```

```
+     y_i_star  = sum(bootstrap[i,])/B
+     for(b in 1:B){
+       cov_hat[i] = cov_hat[i] + mu_hat_star[i,b]*(bootstrap[i,b] - y_i_star)/(B - 1)
+     }
+   }
+
+   regressogram_estimated_risk[j] = sqaure_err + 2*sum(cov_hat)
+ }
>
```

Next we calculate the prediction errors of the pseudo data for the linear spline models with the optimal number of knots = 11.

```
> optimal_knots = 11
> splines_estimated_risk = rep(0, times = 100)
> for(j in 1:100){
+   curdata = pseudodata[,j]
+   cov_hat = rep(0, times = n)
+   mu_hat_star = matrix(0, nrow = n, ncol = B)
+   bootstrap = matrix(0, nrow = n, ncol = B)
+   bootstrap_data = data.frame(x = xx, y = curdata)
+   big_model_boot = lm(y ~ elspline(x, k), data = bootstrap_data)
+   y_hat_boot = fitted(big_model_boot)
+   lsp = lm(y ~ elspline(x, optimal_knots), data = bootstrap_data)
+   sqaure_err = sum(resid(lsp)^2)
+
+   for(b in 1:B){
+     bootstrap[,b] = y_hat_boot + rnorm(n, 0, estimated_var)
+     boot_dat = data.frame(x = xx, y = bootstrap[,b])
+     boot_lsp = lm(y ~ elspline(x, optimal_knots), data = boot_dat)
+     mu_hat_star[,b] = fitted(boot_lsp)
+   }
+
+   for(i in 1:n){
+     y_i_star  = sum(bootstrap[i,])/B
+     for(b in 1:B){
+       cov_hat[i] = cov_hat[i] + mu_hat_star[i,b]*(bootstrap[i,b] - y_i_star)/(B - 1)
+     }
+   }
+
+   splines_estimated_risk[j] = sqaure_err + 2*sum(cov_hat)
+ }
```

Next we calculate the estimated errors of the pseudo data for the kernel regression models with the optimal bandwidth = 0.25.

```
> optimal_kernel = 0.25
> kernel_estimated_risk = rep(0, times = 100)
> for(j in 1:100){
+   curdata = pseudodata[,j]
+   cov_hat = rep(0, times = n)
+   mu_hat_star = matrix(0, nrow = n, ncol = B)
+   bootstrap = matrix(0, nrow = n, ncol = B + 1)
+   bootstrap_data = data.frame(x = xx, y = curdata)
+   bootstrap_data = bootstrap_data[order(bootstrap_data$x),]
+   big_model_boot = lm(y ~ elspline(x, k), data = bootstrap_data)
+   y_hat_boot = fitted(big_model_boot)
+   kernel_model = ksmooth(bootstrap_data$x, bootstrap_data$y, 'normal', bandwidth = o
+   y_hat_kernel = kernel_model$y
+   square_err = sum((y_hat_kernel - bootstrap_data$y)^2)
+
+   for(b in 1:B){
+     bootstrap[,b] = y_hat_boot + rnorm(n, 0, estimated_var)
+     boot_dat = data.frame(x = xx, y = bootstrap[,b])
+     boot_dat = boot_dat[order(boot_dat$x),]
+     boot_kernel = ksmooth(boot_dat$x, boot_dat$y, 'normal', bandwidth = optimal_kern
+     mu_hat_star[,b] = boot_kernel$y
+   }
+   bootstrap[,B+1] = xx
+   bootstrap = bootstrap[order(bootstrap[,B+1]),]
+   for(i in 1:n){
+     y_i_star  = sum(bootstrap[i,])/B
+     for(b in 1:B){
+       cov_hat[i] = cov_hat[i] + mu_hat_star[i,b]*(bootstrap[i,b] - y_i_star)/(B - 1)
+     }
+   }
+
+   kernel_estimated_risk[j] = square_err + 2*sum(cov_hat)
+ }
```

Next we calculate the estimated errors of the pseudo data for the loess models with the optimal span = 0.2

```
> optimal_span = 0.2
> loess_estimated_risk = rep(0, times = 100)
> for(j in 1:100){
+   curdata = pseudodata[,j]
+   cov_hat = rep(0, times = n)
+   mu_hat_star = matrix(0, nrow = n, ncol = B)
+   bootstrap = matrix(0, nrow = n, ncol = B)
+   bootstrap_data = data.frame(x = xx, y = curdata)
+   big_model_boot = lm(y ~ elspline(x, k), data = bootstrap_data)
```

```
+   y_hat_boot = fitted(big_model_boot)
+   cur_loess = loess(y~x,bootstrap_data , span = optimal_span)
+   sqaure_err = sum((cur_loess$residuals)^2)
+
+   for(b in 1:B){
+     bootstrap[,b] = y_hat_boot + rnorm(n, 0, estimated_var)
+     boot_dat = data.frame(x = xx, y = bootstrap[,b])
+     boot_loess = loess(y~x,boot_dat , span = optimal_span)
+     mu_hat_star[,b] = fitted(boot_loess)
+   }
+
+   for(i in 1:n){
+     y_i_star  = sum(bootstrap[i,])/B
+     for(b in 1:B){
+       cov_hat[i] = cov_hat[i] + mu_hat_star[i,b]*(bootstrap[i,b] - y_i_star)/(B - 1)
+     }
+   }
+
+   loess_estimated_risk[j] = sqaure_err + 2*sum(cov_hat)
+ }
```

Now we have found the risk estimates for 100 sets of pseudo data for all four models with optimal parameters.We want to find the largest α such that the $(1-\alpha)$ confidence interval that contains 0. Equivalently, we are finding the p-value of the hypothesis testing with null hypothesis $H_0 : riskA = riskB$ where riskA and riskB are the risks of model A and model B. we write a function to compute the p-value and set the significence level of the hypothesis testing to 0.05. The function will return 0 if we fail to reject the null hypothesis i.e the two models have the same risks.The function will return 1 if model A is better and return 2 if model B is better.

```
> calc_p_val = function(riskA, riskB){
+   diff = riskA - riskB
+   confidence_interval = sort(diff)
+   if(confidence_interval[1]>0){
+     return(2)
+   }
+   if(confidence_interval[100]<0){
+     return(1)
+   }
+   i = 1
+   j = 100
+   while(confidence_interval[i+1] < 0){
+     i = i + 1
+   }
+   while(confidence_interval[j-1] > 0){
+     j = j - 1
```

```
+   }
+   alpha = min(2 * i / 100, 1 - (2*j)/100)
+   if (alpha > 0.05){
+     return(0)
+   }
+   if(100 - j > i){
+     return(2)
+   }
+   else{
+     return(1)
+   }
+ }
```

Now we use the above function to compare the risks of each pair of models. There will be six pairs in total.

```
> compare_regress_spline = calc_p_val(regressogram_estimated_risk, splines_estimated_r
> #1: regressogram is better than linear splines
> compare_regress_kernel = calc_p_val(regressogram_estimated_risk, kernel_estimated_ri
> #1: regressogram is better than kernel regression
> compare_regress_loess = calc_p_val(regressogram_estimated_risk, loess_estimated_risk
> #1: regressogram is better than loess
> compare_spline_kernel = calc_p_val(splines_estimated_risk, kernel_estimated_risk)
> #1: linear splines is better than kernel regression
> compare_spline_loess = calc_p_val(splines_estimated_risk, loess_estimated_risk)
> #1: linear splines is better tahn loess
> compare_kernel_loess = calc_p_val(kernel_estimated_risk, loess_estimated_risk)
> #2: loess is better than kernel regression
```

In conclusion, the hypthesis testing shows that regressogram is better than linear spline, kernel regression and loess regression. And linear spline model is bettern than both kernel regression and loess regression and finally loess is better than kernel regression. Therefore, regressogram is the best model.

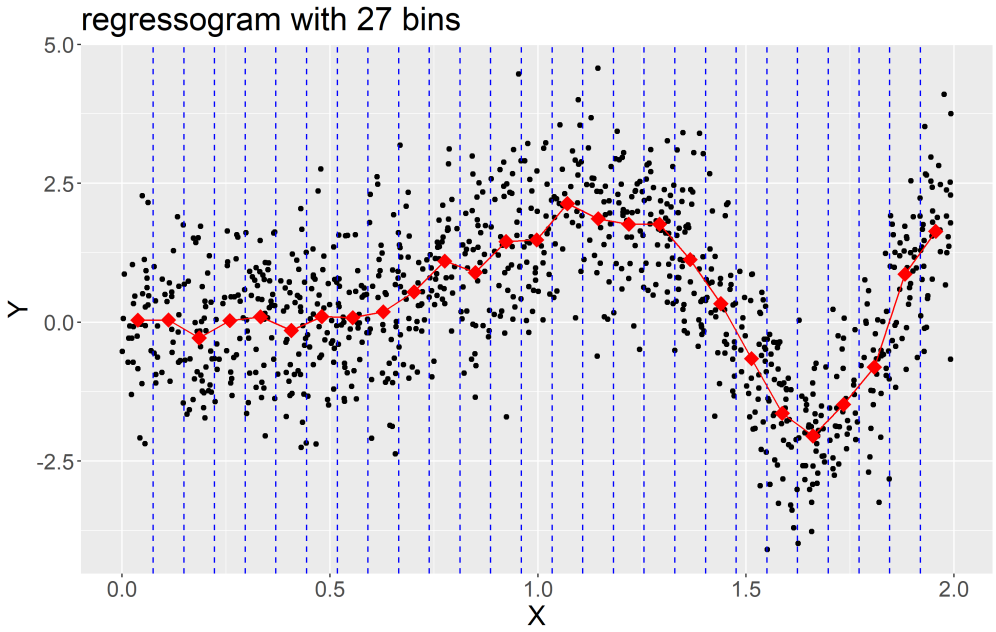
2 K-fold Cross-Validation for Finding Optimal Paremters

We apply 7−fold cross-validation to all four models to find the optimal paremeters. First we will import the required library and create the shuffled folds for cross validation.

```
> set.seed(7)
> library(caret)
> cv_folds = createFolds(dat$yy, k = 7, list = TRUE, returnTrain = FALSE)
```

Now we apply cross validation to the regressogram models.

```
> err_estimates_regress = rep(0, times = 80)
> max_x = max(dat$xx)
```



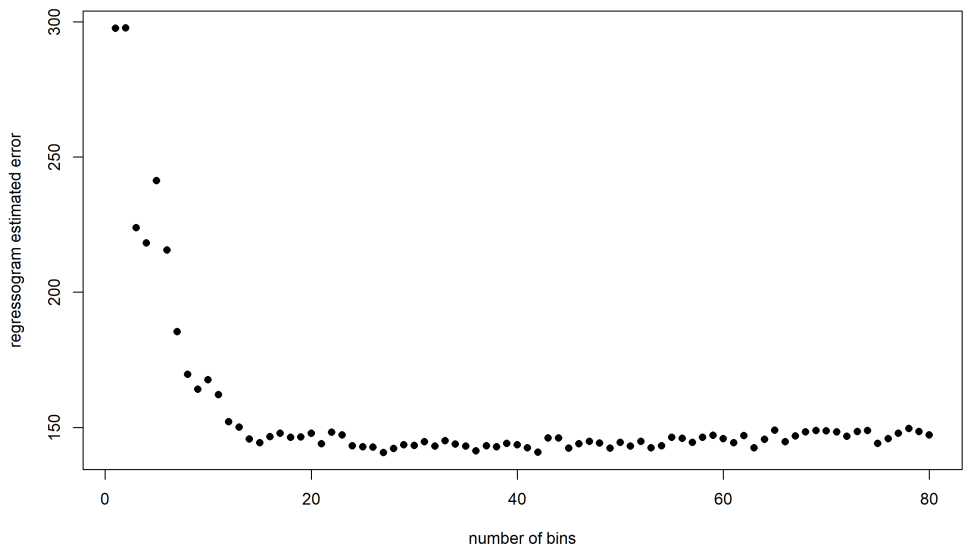

```
> min_x = min(dat$xx)
> for(j in 1:80){
+   for(i in 1:7){
+     validation_set = dat[cv_folds[[i]],]
+     training_set = dat[-cv_folds[[i]],]
+     xy = data.frame(x = training_set$xx, y = training_set$yy)
+     xy = xy[order(xy$x),]
+     z = cut(xy$x,breaks=seq(min_x,max_x,length=j+1),
+                   labels=1:j,include.lowest=TRUE)
+     xyz = data.frame(xy, z = z)
+     MEANS = c(by(xyz$y,xyz$z,FUN=mean))
+     predict_on_validation = cut(validation_set$xx,breaks=seq(min_x,max_x,length=j+1)
+                                 labels=1:j,include.lowest=TRUE)
+
+     err_estimates_regress[j] = err_estimates_regress[j] + sum((MEANS[predict_on_vali
+   }
+
+ }
> optimal_bins_cv = which.min(err_estimates_regress)
```

The optimal number of bins under cross validation is 27. The fitted curve of the optimal regressogram is similar to the optimal regressogram under Efron's method. The trend of prediction error of the regressograms with different number of bins under cross validation is similar to that under Efron's method.

Next we apply cross validation to the linear spline models.

```
> err_estimates_splines = rep(0, times = 40)
```

Figure 1: Number of bins in regressogram vs predicted errors




```
> for(j in 2:40){
+   for(i in 1:7){
+     validation_set = dat[cv_folds[[i]],]
+     training_set = dat[-cv_folds[[i]],]
+     lsp = lm(yy ~ elspline(xx, j), data = training_set)
+     lsp_prediction = predict(lsp, newdata = validation_set)
+     err_estimates_splines[j] = err_estimates_splines[j] + sum((validation_set$yy - l
+   }
+ }
> optimal_knots_cv = which.min(err_estimates_splines[2:40]) + 1
```

The optimal number of knots under cross validation is 11 which is the same as Efron's method. The trend of prediction error of the linear spline models under cross validation is the same as that of Efron's method.

Next we apply cross validation to the kernel regression models.

```
> err_estimates_kernel = rep(0, times = 20)
> bandwidths = seq(from = 0.05, to = 1, length.out = 20)
> for(j in 1:20){
+   for(i in 1:7){
+     validation_set = dat[cv_folds[[i]],]
+     training_set = dat[-cv_folds[[i]],]
+     validation_set = validation_set[order(validation_set$xx),]
+     training_set = training_set[order(training_set$xx),]
+     kernel_predict = ksmooth(training_set$xx, training_set$yy, 'normal', bandwidth =
```

Figure 2: fitted curve of optimal linear spline model

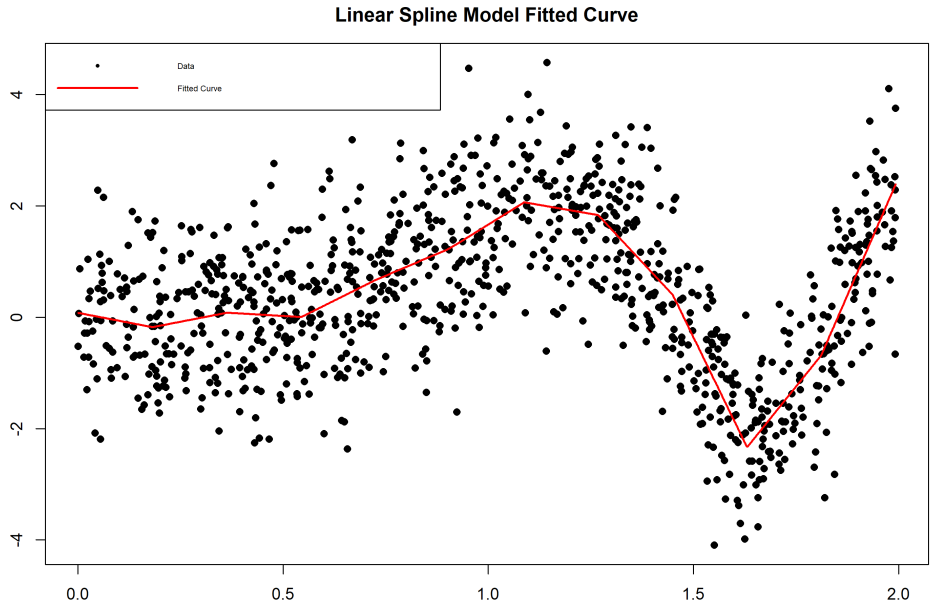


Figure 3: Number of knots in linear splines vs estimated errors

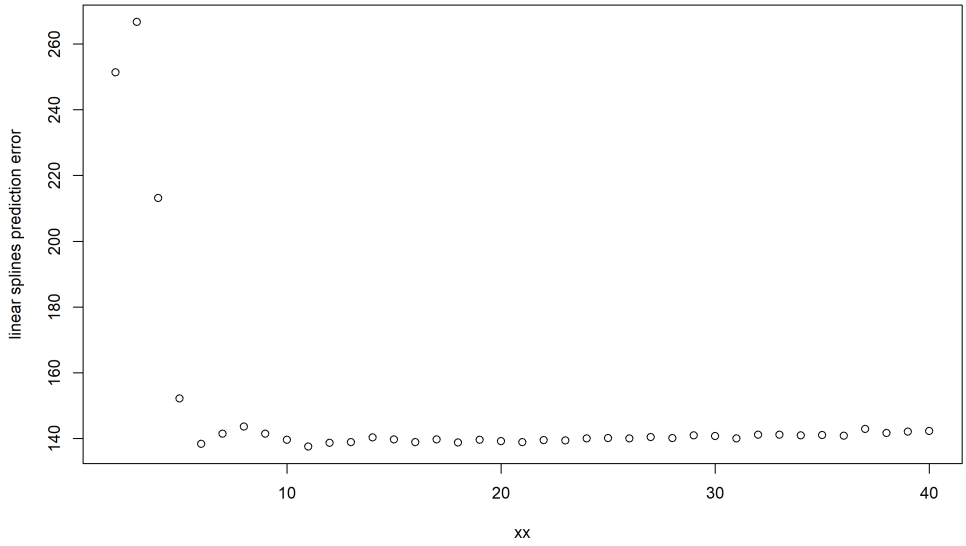
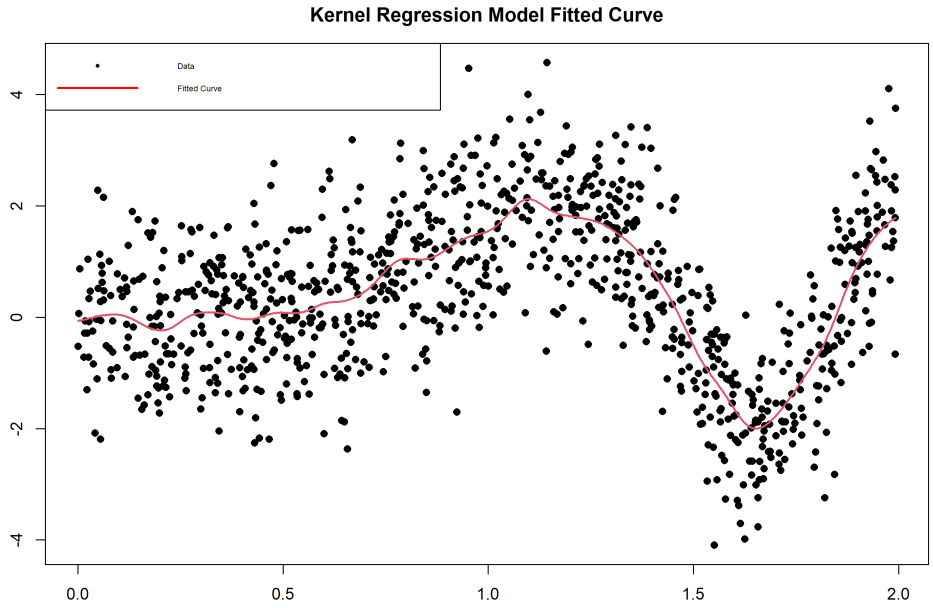


Figure 4: fitted curve of optimal kernel regression




```
+     err_estimates_kernel[j] = err_estimates_kernel[j] + sum((validation_set$yy - ker
+   }
+ }
> optimal_band_cv = bandwidths[which.min(err_estimates_kernel)]
```

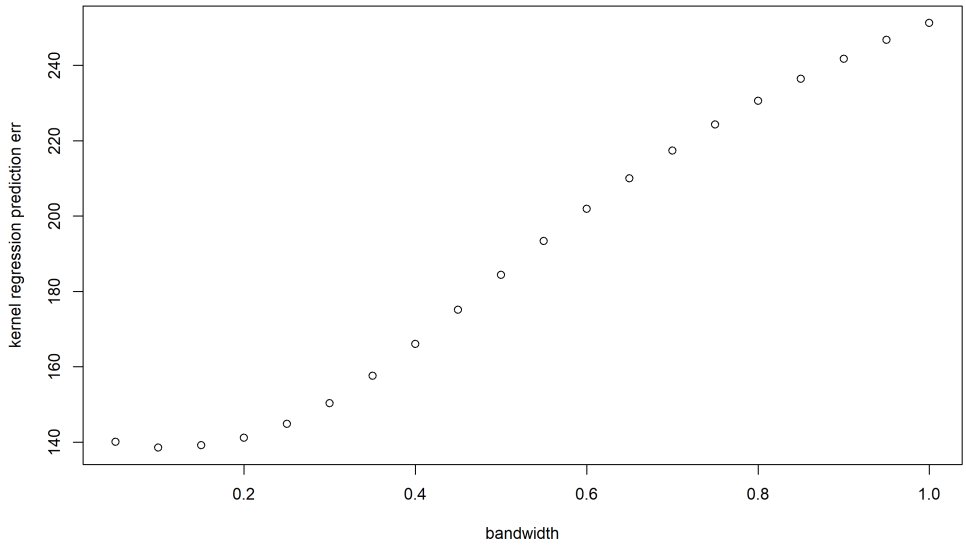
the optimal bandwidth under cross validation is 0.1 whereas the optimal bandwidth under Efron's method is 0.25. The fitted curved of the kernel regression with optimal bandwidth from cross validation is a bit wiggly, suggesting that the model may be slightly overfitted. We may conclude that Efron's method works better at giving optimal parameter for the kernel regression model.

Now we use k-fold cross-validation for loess regression

```
> err_estimates_loess = rep(0, times = 40)
> spans = seq(from = 0.05, to = 1, length.out = 40)
> for(j in 1:40){
+   for(i in 1:7){
+     validation_set = dat[cv_folds[[i]],]
+     training_set = dat[-cv_folds[[i]],]
+     loess_model = loess(yy ~ xx, training_set, span = spans[j], control = loess.cont
+     validation_predict = predict(loess_model, newdata = validation_set)
+     err_estimates_loess[j] = err_estimates_loess[j] + sum((validation_set$yy - valid
+   }
+ }
> optimal_span_cv = spans[which.min(err_estimates_loess)]
```

the optimal span under cross validation is 0.2449 where as the optimal span under Efron's

Figure 5: bandwidth of kernel regression vs estimated errors



method is 0.2. So the two methods do not differ very much. The trend of prediction error vs length of spans are also similar for cross-validation and Efron's method.

3 Paper Summary

The paper discusses the theory and the algorithm of Gaussian process regression. The paper starts with the more familiar weight space view of the Gaussian process regression. We start with a standard linear regression model with Gaussian noise

$$f(x) = \mathbf{x}^T\mathbf{w}, \qquad y = f(x) + \epsilon, \quad \epsilon \sim N(0, \sigma_n^2)$$

Following the Bayesian formalism, we specify a *prior* over the weights i.e. assuming that

$$\mathbf{w} \sim N(\mathbf{0}, \Sigma_p)$$

Then we can compute the posterior distribution over weights using Bayes'rule as

$$p(\mathbf{w}|X, y) \sim N(\bar{\mathbf{w}} = \frac{1}{\sigma_n^2}A^{-1}Xy, \quad A^{-1}), \quad where \quad A = \sigma_n^{-2}XX^T + \Sigma_p^{-1}$$

More generally we can use a function $\phi(\mathbf{x})$ to map a D-dimensional input vector $\mathbf{x}$ into a N dimensional feature space. Then we have a model

$$f(\mathbf{x}) = \phi(\mathbf{x})^T\mathbf{w}$$

Figure 6: fitted curve of optimal loess

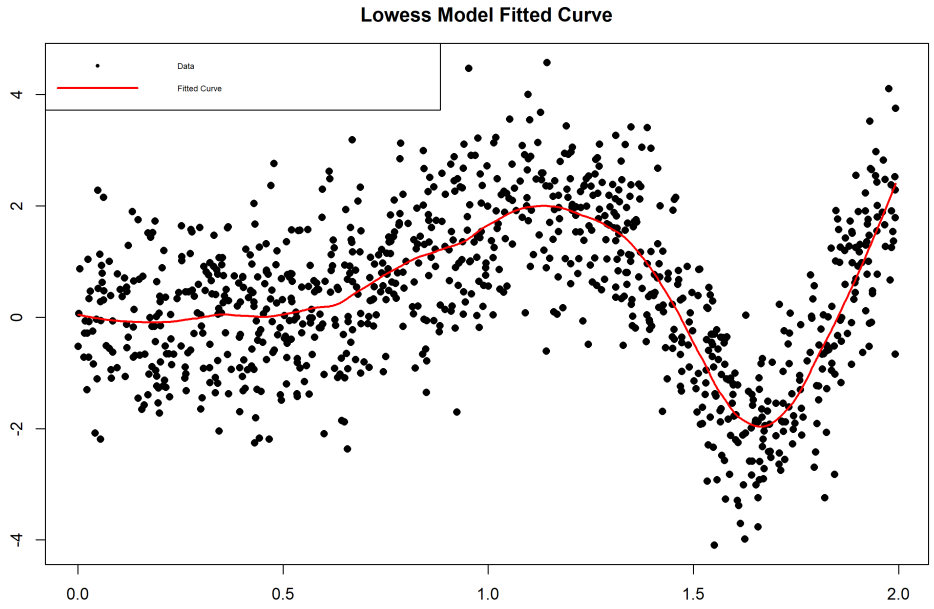
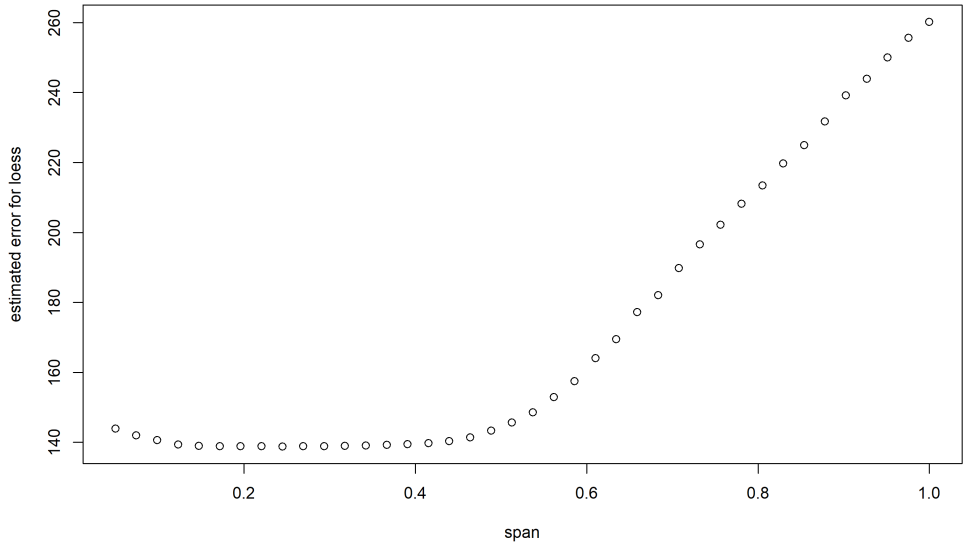


Figure 7: Spans for loess vs predicted errors



We let $\mathbf{x}_*$ be new test data and let $f_* := f(x_*)$ be the prediction of the new test data. Using the above formula, we have a formual for predictive posterior distribution:

$$\mathbf{f}_*|\mathbf{x}_*, X, y \quad \sim N(\frac{1}{\sigma_n^2}\phi(\mathbf{x}_*)^T A^{-1}\Phi(X)y, \quad \phi(\mathbf{x}_*)^T A^{-1}\phi(\mathbf{x}_*)) \quad where \quad A = \sigma_n^{-2}\Phi(X)\Phi(X)^T + \Sigma_p^{-1}$$

We note that the covariance in the above formula can be rewritten as a kernel or covariance function that will correspond to the function space view.
The function-space view is equivalent to the weight-space approach and considers inference directly in function space. A Gaussian process is a collection of random variables such that any finite number of random variables in the collection have a joint Gaussian distribution. A Gaussian process is completely determined by its mean function and covariance function. We define the mean function $m(\mathbf{x})$ and the covariance function $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$ as

$$m(\mathbf{x}) = E[f(\mathbf{x})],$$
$$k(\mathbf{x}, \mathbf{x}') = E[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$$

If we use *squared exponential* as our covariance functionn, then the SE covariance function specifies the covariance between pairs of random variables:

$$cov(f(\mathbf{x_p}), f(\mathbf{x_q})) = k(f(\mathbf{x_p}), f(\mathbf{x_q})) = exp(-\frac{1}{2}|\mathbf{x_p}, \mathbf{x_q}|^2)$$

It is known that squared exponential covariance function corresponds to a Bayesian linear regression model with infinite number of basis function. Moreover, the specification of the covariance function of a Gaussian process gives a distribution over functions.
Finally, we consider a model with noise $y = f(\mathbf{x}) + \epsilon$ where ϵ is an iid Gaussian noice with variance σ_n^2. We have the covariance funcrtion

$$cov(y_p, y_q) = k(\mathbf{x}, \mathbf{x}') + \sigma_n^2\delta_{pq} \quad or \quad cov(\mathbf{y}) = K(X, X) + \sigma_n^2 I$$

We can write the join distribution of the observed target values and function values at the test locations under the prior as

$$\begin{pmatrix} y \\ f_* \end{pmatrix} \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} K(X,X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{pmatrix}\right).$$

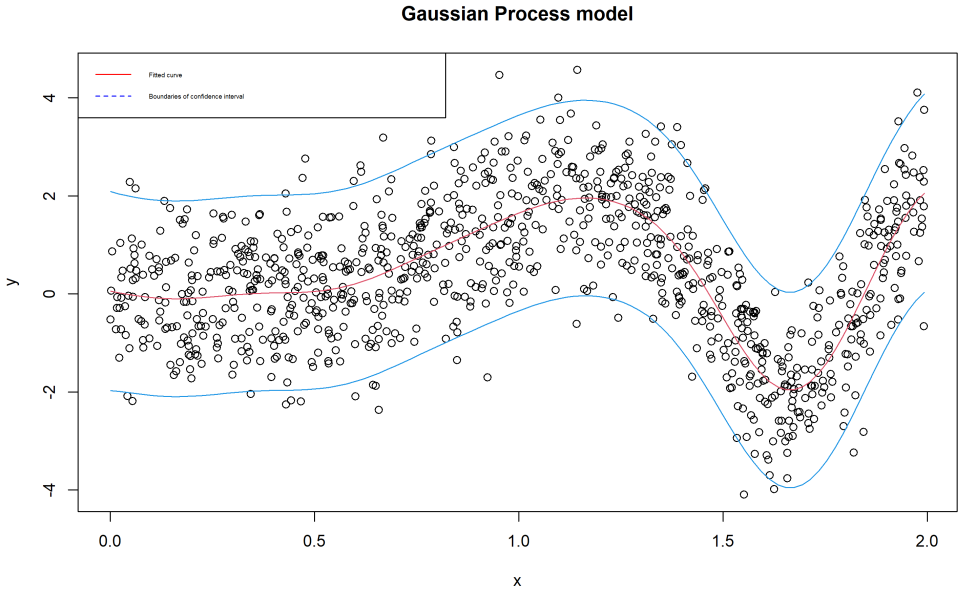
To get the posterior distribution over functions we need to restrict this joint distribution to contain only those functions which agree with the observed data points. We do this by taking the conditioning the distribution on the observations. Then we get the key predictive equations for Gaussian process regression

$$\mathbf{f}_*|X, \mathbf{y}, X_* \sim N(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)),$$

where

$$\bar{\mathbf{f}}_* = E[\mathbf{f}_*|X, \mathbf{y}, X_*] = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y},$$
$$\text{cov} = K(X_*, X_*) - K(X_*, X)\left[K(X, X) + \sigma_n^2 I\right]^{-1} K(X, X_*).$$

Figure 8: Gaussian Process model



4 Gaussian Process Regression

We fit the Gaussian process regression model to the data. We will use the package 'GauPro'.

```
> library(GauPro)
> gauss_process_model = GauPro(dat$xx, dat$yy, parallel=FALSE)
```

Finally we plot the fitted curve of the Gaussian regression model and the 95% confidence interval. We see that most of the data fits inside the confidence interval and thus the model works very well.