CHAPTER 9: CRAFTMANSHIP

MODEL EXTENSION/STACKING

I. Overview

Chapter 8 is about "**craftmanship**". It recognizes the complexity of real-world problems, and highlight how we can modify or combine existing methods in integrative ways to solve a problem. Three examples are given, including the Kernel regression model that generalized the idea of linear regression, the conditional variance regression model that interestingly layers one regression model into another, and the tree-based quality monitoring method that converts traditional quality monitoring into classification framework.

II. Kernel Regression Model

II.1 Rationale and Formulation

Linear regression model looks intuitive, but it is built on very strong assumptions. One strong assumption is that, the linear regression model treats any data point in a global fashion. In other words, while a data point

is located in a particular geographical area, it has impact on the model's prediction on any other location in the space. This could be irrational, as in some applications the data point collected in a local area may only tell information about that area, not easily generalizable to the whole space. This risk is shown in Figure 9.1.

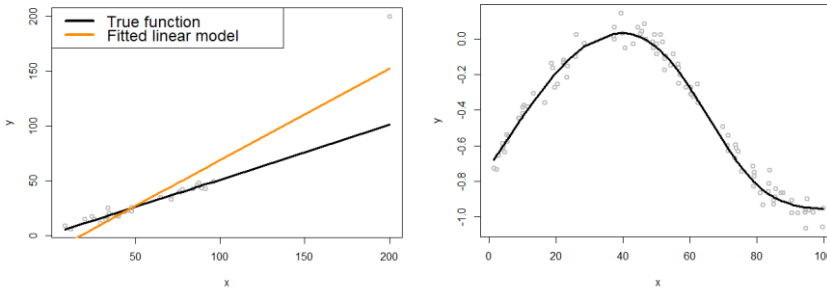

Figure 9.1: Risk of linear regression model as a global model. (left) A single outlier could impact the regression model as a whole; (Right) Many data problems call for localized regression models

The R code for conducting the experiment shown in the left figure in Figure 9.1 is shown in below.

```
# Write a nice simulator to generate dataset with one predictor a
nd one outcome
# from a polynomial regression model
require(splines)

seed <- rnorm(1)
set.seed(seed)
gen_data <- function(n, coef, v_noise) {
  eps <- rnorm(n, 0, v_noise)
  x <- sort(runif(n, 0, 100))
  X <- cbind(1,ns(x, df = (length(coef) - 1)))
  y <- as.numeric(X %*% coef + eps)
  return(data.frame(x = x, y = y))
}
```

```
n_train <- 30
coef <- c(1,0.5)
v_noise <- 3
tempData <- gen_data(n_train, coef, v_noise)
tempData[31,] = c(200,200)
# Fit the data using linear regression model
x <- tempData[, "x"]
y <- tempData[, "y"]
fit <- lm(y~x,data=tempData)
# Plot the data
x <- tempData$x
X <- cbind(1, x)
y <- tempData$y
plot(y ~ x, col = "gray", lwd = 2)
lines(x, X %*% coef, lwd = 3, col = "black")
lines(x, fitted(fit), lwd = 3, col = "darkorange")
legend(x = "topleft", legend = c("True function", "Fitted linear
model"), lwd = rep(4, 4), col = c("black", "darkorange"), text.wi
dth = 100, cex = 1.5)
```

So how to fix this problem, while on the other hand we don't want to derivate from the linear regression framework too far?

One approach we could utilize is to look at the linear regression model in a new perspective. Statisticians and data scientists who innovate on modeling use this approach of look-for-a-new-perspective all the time[1].

Let's look at the simple linear regression problem $y = \beta_0 + \beta_1 x$. Let's further simplify it by assuming that we know the mean of y is zero, so is the mean of x. This will lead to the model as $y = \beta_1 x$ and the estimator of β_1 as

$$\beta_1 = \frac{(\sum_{i=1}^{n} x_i y_i)}{\sum_{i=1}^{n} x_i^2}.$$

Thus, when we try to make prediction on a new data point with a given x^*, the prediction y^* will be

[1] Some examples for interested readers: Neal, R. *Bayesian learning for neural networks*, Springer Verlag 1996. Lee, K. and Kim, J. *On the equivalence of linear discriminant analysis and least squares*, AAAI 2005. Ye, J. *Least squares linear discriminant analysis*, ICML 2007. Li, F., Yang, Y. and Xing, E. *From LASSO regression to feature vector machine*, NIPS 2005.

$$y^* = x^* \frac{(\sum_{i=1}^{n} x_i y_i)}{\sum_{i=1}^{n} x_i^2}.$$

This could be further reformed as:

$$y^* = \sum_{i=1}^{n} y_i \frac{x_i}{\sum_{i=1}^{n} x_i^2} x^*,$$

which is equivalent with

$$y^* = \sum_{i=1}^{n} y_i \frac{x_i x^*}{n S_x^2}.$$

Now if we look closely at this formula, we can draw interesting observations how linear regression model works in prediction on a new location using its knowledge on other locations (e.g., the historical data points (x_i, y_i) for $i = 1,2, \dots, n$). It first evaluates the similarity between the new location with each of the knowing locations, as reflected in $\frac{x_i x^*}{n S_x^2}$, where $x_i x^*$ calculates the similarity and $n S_x^2$ is a normalization factor. Then, the prediction y^* is a weighted sum of y_i for $i = 1,2, \dots, n$ while the weight of y_i is proportional to the similarity between x_i and x^*. From this perspective, we see linear regression model as a very empirical prediction model that bears the same idea with those lazy learning methods such as k-nearest-neighbor regression model or local regression models. The difference here, in the linear regression model, is that a special similarity measure (i.e., $\frac{x_i x^*}{n S_x^2}$) is used, that means the weight of a data point depends on how far it is from the center of the data, not how far it is from the point at which we are trying to predict. Thus, for this similarity measure to work we need to hope that the underlying model is globally linear.

II.2 Theory and Method

We then pursue a generalized family of model, defined as:

$$y^* = \sum_{n=1}^{N} y_n w(x_n, x^*).$$

Here, $w(x_n, x^*)$ is the weight that characterizes the similarity between the point that will be predicted on (i.e., x^*) and the existing data points, x_n for $n = 1,2, \dots, N$. Roughly speaking, there are two types of methods to define this similarity metric.

One is the **K-nearest neighbor (KNN) smoother**:

$$w(x_n, x^*) = \begin{cases} \frac{1}{k}, & if x_n \text{ is one of the k nearest neighbors of } x^* \\ 0, & if x_n \text{ is NOT in the k nearest neighbors of } x^* \end{cases}.$$

Here, to define the neighbors of a data point, a distance function is needed, e.g., a popular one is the Euclidean distance function.

Note that, a distinct feature of the KNN smoother is the discrete manner to define similarity between data points. Which is, a data point is either a neighbor of another data point, or not. Not like this, the **kernel smoother** is another approach that has the continuity in similarity between data points. A kernel smoother defines $w(x_n, x^*)$ in the following manner:

$$w(x_n, x^*) = \frac{K(x_n, x^*)}{\sum_{n=1}^{N} K(x_n, x^*)}.$$

There have been many kernel functions developed, for example, as shown in Table 9.1:

Table 9.1: Some kernel functions used in machine learning

Kernel function	Mathematical form	Parameters
Linear	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$	$null$
Polynomial	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j + 1)^q$	q
Gaussian radial basis	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|x_i - x_j\|^2}$	$\gamma \geq 0$
Laplace radial basis	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|x_i - x_j\|}$	$\gamma \geq 0$
Hyperbolic tangent	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\boldsymbol{x}_i^T \boldsymbol{x}_j + b)$	b
Sigmoid	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(a\boldsymbol{x}_i^T \boldsymbol{x}_j + b)$	a, b
Bessel function	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{bessel_{v+1}^n(\sigma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)}{(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)^{-n(v+1)}}$	σ, n, v
ANOVA radial basis	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\sum_{k=1}^{n} e^{-\sigma(x_i^k - x_j^k)}\right)^d$	σ, d

II.3 R Lab

Using the established framework in Chapter 5 to generate nonlinear dataset, here, we use the following R code to implement the KNN regression model.

```
# Simulate one batch of data
n_train <- 100
coef <- c(-0.68,0.82,-0.417,0.32,-0.68)
v_noise <- 0.2
n_df <- 20
df <- 1:n_df
tempData <- gen_data(n_train, coef, v_noise)
# Fit different KNN models
x <- tempData$x
X <- cbind(1, ns(x, df = (length(coef) - 1)))
y <- tempData$y
# install.packages("FNN")
require(FNN)

## Loading required package: FNN

xy.knn3<- knn.reg(train = x, y = y, k=3)
xy.knn10<- knn.reg(train = x, y = y, k=10)
xy.knn50<- knn.reg(train = x, y = y, k=50)
```

Then, we draw the true model (as a black curve) and the sampled data points using the following R code. Result is shown in Figure 9.2.

```
# Plot the data
plot(y ~ x, col = "gray", lwd = 2)
```

And we further layer the fitted KNN regression models with different choices on the parameter k onto the figure.

```
lines(x, X %*% coef, lwd = 3, col = "black")
lines(x, xy.knn3$pred, lwd = 3, col = "darkorange")
lines(x, xy.knn10$pred, lwd = 3, col = "dodgerblue4")
lines(x, xy.knn50$pred, lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("True function", "KNN (k = 3)",
"KNN (k = 10)", "KNN (k = 50)"),
       lwd = rep(3, 4), col = c("black", "darkorange", "dodgerblu
e4", "forestgreen"),
       text.width = 32, cex = 0.85)
```

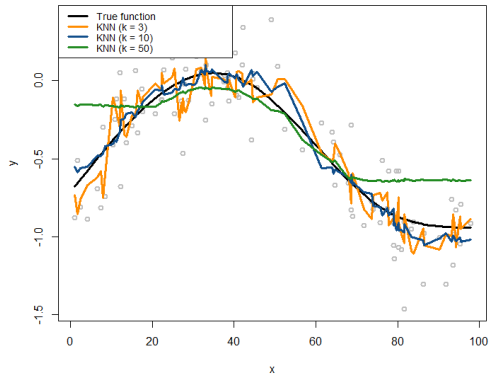

Figure 9.2: KNN regression models with different choices on the number of nearest neighbors

It can be seen that, with smaller number of nearest neighbors, the fitted curve by the KNN regression model is less smooth. That means, a KNN regression model with a smaller parameter k tends to predict on a data point by relying on only a few local data points, ignoring information provided by other data points that are far away. This is very different from the spirit of linear regression model, in which no matter how far away a data point is, it can change predictions on any other data point globally as it changes the regression line as a whole.

A related observation is, in terms of model complexity, the smaller the parameter k, the more complex the regression model. This is often taken as a counterintuitive conclusion.

Similarly, we can repeat the experiments introduced above for implementing the kernel smoother regression model. Here, we use the Gaussian radial basis kernel function in the kernel smoother regression model. Result is shown in Figure 9.3.

```
# Repeat the above experiments with kernel smoother
# Plot the data
plot(y ~ x, col = "gray", lwd = 2)
lines(x, X %*% coef, lwd = 3, col = "black")
lines(ksmooth(x,y, "normal", bandwidth=2),lwd = 3, col = "darkora
```

```
nge")
lines(ksmooth(x,y, "normal", bandwidth=5),lwd = 3, col = "dodgerb
lue4")
lines(ksmooth(x,y, "normal", bandwidth=15),lwd = 3, col = "forest
green")
legend(x = "topright", legend = c("True function", "Kernel Reg (b
w = 2)", "Kernel Reg (bw = 5)", "Kernel Reg (bw = 15)"),
       lwd = rep(3, 4), col = c("black", "darkorange", "dodgerblu
e4", "forestgreen"),
       text.width = 32, cex = 0.85)
```

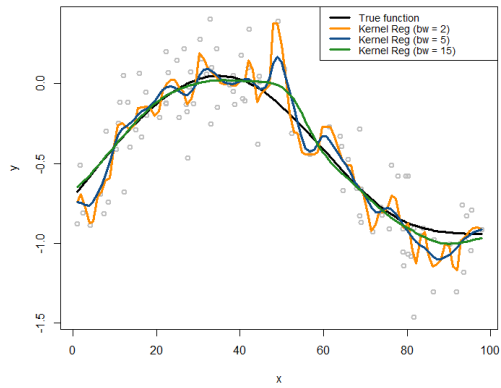

Figure 9.3: Kernel regression models with different choices on the `bandwidth` parameter of the Gaussian radial basis kernel function

As shown in Figure 9.3, the `bandwidth` parameter in the kernel smoother regression plays a similar role as the parameter k in KNN regression. The larger the bandwidth, the smoother of the regression curve. On the other hand, it can be seen that the curve of kernel smoother is smoother than the KNN curves. This observation corresponds to the note we mentioned in the beginning of this subsection that KNN is discretely parameterized while kernel smoother introduces smoothness and continuity into the definition of the neighbors of a data point (thus no hard thresholding is needed to classify whether a data point is a neighbor of another data point).

III. Conditional Variance Regression Model

II.1 Rationale and Formulation

Another common complication when applying linear regression model in real-world applications is that the variance of the response variable may also change. This phenomenon is called as **heteroscedasticity** in regression analysis. This complication can be taken care of by a conditional variance regression model that allows the variance of the response variable to be a (usually implicit) function of the input variables. This leads to the following model:

$$y = \boldsymbol{\beta}^T \boldsymbol{x} + \epsilon_{\boldsymbol{x}},$$

and $\epsilon_{\boldsymbol{x}}$ is the error term that is a normal distribution with varying variance:

$$\epsilon_{\boldsymbol{x}} \sim N(0, \sigma_{\boldsymbol{x}}^2).$$

The remaining issue is how to estimate the regression parameters.

II.2 Theory and Method

Known $\boldsymbol{\sigma_x^2}$: If we have known the $\sigma_{\boldsymbol{x}}^2$, this will lead to the following scheme for parameter estimation of the unknown regression parameters. The likelihood function is:

$$-\frac{n}{2}\ln 2\pi - \frac{1}{2}\sum_{n=1}^{N} \log \sigma_{x_n}^2 - \frac{1}{2}\sum_{n=1}^{N} \frac{\left(y_n - \boldsymbol{\beta}^T \boldsymbol{x}_n\right)^2}{\sigma_{x_n}^2}.$$

As we have known $\sigma_{\boldsymbol{x}}^2$, the parameters to be estimated only involve the last part of the likelihood function. Thus, we estimate the parameters that minimize

$$\frac{1}{2}\sum_{n=1}^{N} \frac{\left(y_n - \boldsymbol{\beta}^T \boldsymbol{x}_n\right)^2}{\sigma_{x_n}^2}.$$

This could be written in the matrix form as

$$\min_{\boldsymbol{\beta}} (\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})^T \mathbf{W} (\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta}),$$

where $\mathbf{W}$ is a diagonal matrix with its diagonal elements as $\mathbf{W}_{nn} = \frac{1}{\sigma_{x_n}^2}$.

To solve this optimization problem, we can take the gradient of the objective function and set it to be zero:

$$\frac{\partial(\boldsymbol{Y}-\mathbf{X}\boldsymbol{\beta})^T\mathbf{W}(\boldsymbol{Y}-\mathbf{X}\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} = 0,$$

which gives rise to the equation:

$$\mathbf{X}^T\mathbf{W}(\boldsymbol{Y}-\mathbf{X}\boldsymbol{\beta}) = 0.$$

This leads to the weighted least square estimator of $\boldsymbol{\beta}$ as

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\boldsymbol{Y}.$$

Unknown $\boldsymbol{\sigma_x^2}$: A more complicated situation, also more realistic situation, is that we don't know σ_x^2. This means that we need to estimate σ_x^2. To do so, it is important to recognize that this problem bears a regression core in its formulation. It is to use the input variables $\boldsymbol{x}$ to predict a new outcome variable, σ_x^2. The only complication here is that, we don't have the "natural measurements" of the new outcome variable that is needed to apply the regression method. Since, here, the outcome variable σ_x^2 is not directly measuable. This is a **latent variable** in statistics.

To overcome this problem, we can estimate the measurements of the latant variable, denoted as $\hat{\sigma}_{x_n}^2$ for $n = 1,2,\dots,N$. This philosophy of taking some variables as latent variables and further using statistical estimation/inference to fill in the unseen measurements is popular and fertile in statistics that underlies many models such as the latent factor models, structural equation models, missing values imputation, EM algorithm, Gaussian mixture model, graphical models with latent variables, etc.

Thus, we propose the following steps:

1. Initialize $\hat{\sigma}_{x_n}^2$ for $n = 1,2,\dots,N$, by any reasonable approach including the random generation of values.
2. Build a regression model for the mean of the response variable using the weighted LS estimator. Estimate $\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\boldsymbol{Y}$ and get $\hat{y}_n = \widehat{\boldsymbol{\beta}}^T\boldsymbol{x}_n$.
3. Derive the residuals $\hat{\varepsilon}_n = y_n - \hat{y}_n$.

4. Build a regression model, e.g., using the kernel regression which is a nonparametric method, to fit $\hat{\varepsilon}_n^2$ using $\boldsymbol{x}_n$ for $n = 1,2,\dots,N$.
5. Predict $\hat{\sigma}_{\boldsymbol{x}_n}^2$ for $n = 1,2,\dots,N$ using the fitted regression model in Step 3.
6. Repeat Step 2 – Step 5 until convergence or satisfaction of a stopping criteria (could be a fixed number of iterations or small change of parameters).

We can see that the proposed conditional variance regression model is a composition of two regular linear regression models to work out the heteroscedasticity. This is a typical model stacking strategy to create new models based on existing models.

II.3 R Lab

We first simulate dataset to see how the proposed iterative procedure of the conditional variance regression model can work out the heteroscedasticity. The simulated data has one predictor and one outcome. The model parameters (including the intercept and regression coefficient) are assigned values as `coef <- c(1,0.5)`. The variance is a function of the predictor `x`, i.e., which equals to `0.5+0.8*x^2`.

```
# Conditional variance function
# Simulate a regression model with heterogeneous variance
gen_data <- function(n, coef, v_noise) {
  x <- rnorm(100,0,2)
  eps <- rnorm(100,0,sapply(x,function(x){0.5+0.8*x^2}))
  X <- cbind(1,x)
  y <- as.numeric(X %*% coef + eps)
  return(data.frame(x = x, y = y))
}
n_train <- 100
coef <- c(1,0.5)
v_noise <- 2.5
tempData <- gen_data(n_train, coef, v_noise)
```

While this data presents a typical heteroscedasticity problem, in what follows we apply a regular linear regression model with assumption of

homogenous variance. The fitted line is shown in Figure 9.4, indicating a significant derivation from the true regression model.

```
# Fit the data using linear regression model (OLS)
x <- tempData[, "x"]
y <- tempData[, "y"]
fit.ols <- lm(y~x,data=tempData)
# Plot the data and the models
x <- tempData$x
X <- cbind(1, x)
y <- tempData$y
plot(y ~ x, col = "gray", lwd = 2)
# Plot the true model
lines(x, X %*% coef, lwd = 3, col = "black")
# Plot the linear regression model (OLS)
lines(x, fitted(fit.ols), lwd = 3, col = "darkorange")
legend(x = "topleft", legend = c("True function", "Linear model
(OLS)"),
       lwd = rep(4, 4), col = c("black", "darkorange"), text.widt
h = 4, cex = 1)
```

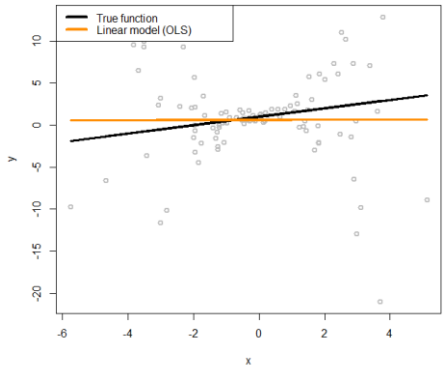

Figure 9.4: Linear regression model to fit a heteroscedastic dataset

We can generate the residuals based on the fitted regular linear regression model, which are plotted in Figure 9.5. A nonlinear regression model, the kernel smoother regression model implemented by `npreg()`, is fitted on these residuals. The true function of the variance is also shown as

the black line in Figure 9.5. It can be seen that the residuals encode the information for us to approximate the underlying true variance function.

```
# Plot the residual estimated from the linear regression model (OLS)
plot(x,residuals(fit.ols)^2,ylab="squared residuals",col = "gray", lwd = 2)
# Plot the true model underlying the variance of the error term
curve((1+0.8*x^2)^2,col = "black", lwd = 3, add=TRUE)
# Fit a nonlinear regression model for residuals
# install.packages("np")
require(np)

var1 <- npreg(residuals(fit.ols)^2 ~ x)

grid.x <- seq(from=min(x),to=max(x),length.out=300)
lines(grid.x,predict(var1,exdat=grid.x), lwd = 3, col = "darkorange")
legend(x = "topleft", legend = c("True function", "Fitted nonlinear model (1st iter)"),
       lwd = rep(4, 4), col = c("black", "darkorange"), text.width = 5, cex = 1.2)
```

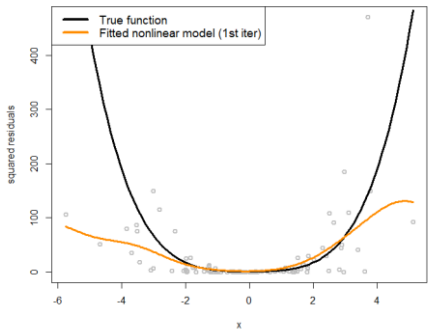

Figure 9.5: Nonlinear regression model to fit the residuals

Thus, we fit another linear regression model with weights of the data points assigned according to the inverse of the variance, i.e., `weights=1/fitted(var1)`, to penalize the influence of the data points that have larger variances on the fit of the regression line. The new regression model is added into Figure 9.4, which generates Figure 9.6. It can be seen

that, with this strategy, the new regression model (the green line) is much more closer with the true model than the regular linear regression model.

```
# Fit a linear regression model (WLS) with the weights specified
# by the fitted nonlinear model of the residuals
fit.wls <- lm(y~x,weights=1/fitted(var1))
plot(y ~ x, col = "gray", lwd = 2,ylim = c(-20,20))
# Plot the true model
lines(x, X %*% coef, lwd = 3, col = "black")
# Plot the linear regression model (OLS)
lines(x, fitted(fit.ols), lwd = 3, col = "darkorange")
# Plot the linear regression model (WLS) with estimated variance
function
lines(x, fitted(fit.wls), lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("True function", "Linear (OLS)",
 "Linear (WLS) + estimated variance"),
       lwd = rep(4, 4), col = c("black", "darkorange","forestgree
n"), text.width = 5, cex = 1)
```

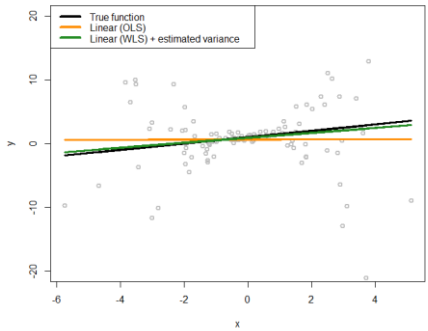

Figure 9.6: Fit the heteroscedastic dataset with two linear regression models using OLS and WLS (that accounts for the heteroscedastic effects with a nonlinear regression model to model the variance regression)

This process could proceed with updating the fitted variance function on the new residuals, as shown in Figure 9.7. Here, it seems that the use of the kernel smoother by `npreg` hits its limit as a local and data-driven model, that could not correctly fit the curve in the two ends. If we have known the form of the variance function as a second order polynomial function, we could use parametric regression model to attain this fitting.

```
# Plot the residual estimated from the linear regression model (O
LS)
plot(x,residuals(fit.ols)^2,ylab="squared residuals",col = "gray",
 lwd = 2)
# Plot the true model underlying the variance of the error term
curve((1+0.8*x^2)^2,col = "black", lwd = 3, add=TRUE)
# Fit a nonlinear regression model for residuals
# install.packages("np")
require(np)
var2 <- npreg(residuals(fit.wls)^2 ~ x)

grid.x <- seq(from=min(x),to=max(x),length.out=300)
lines(grid.x,predict(var1,exdat=grid.x), lwd = 3, col = "darkoran
ge")
lines(grid.x,predict(var2,exdat=grid.x), lwd = 3, col = "forestgr
een")
legend(x = "topleft", legend = c("True function", "Fitted nonline
ar model (1st iter)", "Fitted nonlinear model (2nd iter)"),
       lwd = rep(4, 4), col = c("black", "darkorange", "forestgre
en"), text.width = 6, cex = 1.2)
```

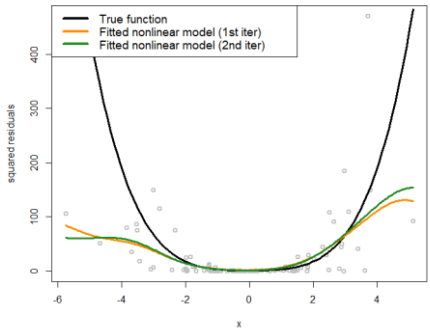

Figure 9.7: Nonlinear regression model to fit the residuals in the 2nd iteration

Now let's apply the conditional variance regression model on the AD dataset. As what we did in the simulated dataset, we first fit a regular linear regression model, then, use the kernel smoother regression model to fit the residuals, and further fit a weighted linear regression model with weights of the data points being assigned according to the inverse of the variance, i.e., `weights=1/fitted(var1)`, to penalize the influence of the data points that

have larger variances on the fit of the regression line. Results are shown in Figure 9.8.

```
AD <- read.csv('AD_bl.csv', header = TRUE)
str(AD)

# Fit the data using linear regression model (OLS)
x <- AD$HippoNV
y <- AD$MMSCORE
fit.ols <- lm(y~x,data=AD)

# Fit a linear regression model (WLS) with the weights specified
# by the fitted nonlinear model of the residuals
var1 <- npreg(residuals(fit.ols)^2 ~ HippoNV, data = AD)

fit.wls <- lm(y~x,weights=1/fitted(var1))

plot(y ~ x, col = "gray", lwd = 2)
# Plot the linear regression model (OLS)
lines(x, fitted(fit.ols), lwd = 3, col = "darkorange")
# Plot the linear regression model (WLS) with estimated variance
function
lines(x, fitted(fit.wls), lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("Linear (OLS)", "Linear (WLS) +
estimated variance"),
       lwd = rep(4, 4), col = c("darkorange","forestgreen"), text.
width = 0.2, cex = 1)
```

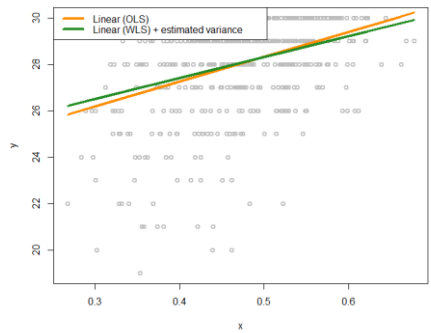

Figure 9.8: Fit the heteroscedastic AD dataset with two linear regression models using OLS and WLS (that accounts for the heteroscedastic effects with a nonlinear regression model to model the variance regression)

We can also visualize the fitted variance functions in Figure 9.9 via the following R code.

```
# Plot the residual estimated from the linear regression model (O
LS)
plot(x,residuals(fit.ols)^2,ylab="squared residuals",col = "gray",
 lwd = 2)
# Fit a nonlinear regression model for residuals
# install.packages("np")
require(np)
var2 <- npreg(residuals(fit.wls)^2 ~ x)

grid.x <- seq(from=min(x),to=max(x),length.out=300)
lines(grid.x,predict(var1,exdat=grid.x), lwd = 3, col = "darkoran
ge")
lines(grid.x,predict(var2,exdat=grid.x), lwd = 3, col = "forestgr
een")
legend(x = "topleft", legend = c("Fitted nonlinear model (1st ite
r)", "Fitted nonlinear model (2nd iter)"),
       lwd = rep(4, 4), col = c( "darkorange", "forestgreen"), te
xt.width = 0.25, cex = 1.2)
```

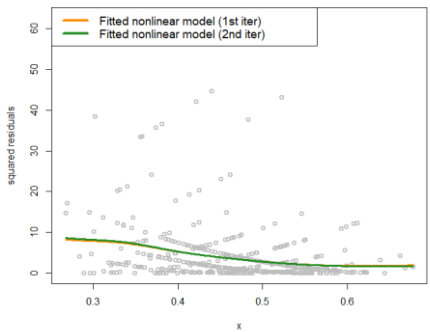

Figure 9.9: Nonlinear regression model to fit the residuals in the 2nd iteration for the AD data

It can be seen that, the heteroscedasticity problem is significant in this problem. Learning the variance function is helpful in this context in terms of at least two aspects. First, in terms of the statistical aspect, it improves the fitting of the regression line. Second, knowing the variance function

itself is important knowledge in healthcare, e.g., as variance implies unpredictability or low quality in healthcare operations.

II.4 Remark

For regression problems, the interest is usually on the modeling of the relationship between the mean of the outcome variable with the input variables. Thus, when there is heteroscedasticity in the data, a nonparametric regression method is recommended to estimate the latent variance information, more from a curve fitting perspective which is to smooth and estimate, rather than a modeling perspective to study the relationship between the outcome variable with input variables. But, of course, this usual tendency doesn't exclude the possibility that we can still study how the input variables affect the variance of the response variable explicitly. Specifically, as the linear regression as we know is a model to link the mean of Y with the input variables X, we can develop an analogical linear regression model to link the variance of Y with the input variables X. The iterative procedure developed above is still applicable here.

IV. System Monitoring as a Decision Tree Model

IV.1 Rationale and Formulation

Another method we'd like to introduce is an interesting framework that was proposed[1] to convert quality monitoring problem in statistical quality control into a classification problem. This is built on the following insight about quality monitoring as a statistical problem that does things with data. In a quality monitoring problem, we often collect the so-called "reference data" from the process in normal conditions. The objective of quality monitoring of this process is to trigger alerts if the new data that come in real time deviate from the reference data. It is hoped that the alerts could

[1] *Deng, H., Runger, G. and Tuv, E. System monitoring with real-time contrasts. Journal of quality technology, 2012*

correspond to real anomality happening in the process, with a small rate of false positive (i.e., alerts triggered when the process is actually normal).

Figure 9.10 shows a monitoring problem. At time 1 to time 40, the reference data are collected from a normal process. From time 41, we monitor the process with real-time data. As the process from time 41 to time 80 is under normal condition, we should not trigger any alert. From time 81, the process is abnormal and therefore it is expect that the quality monitoring system should trigger an alert as soon as possible.

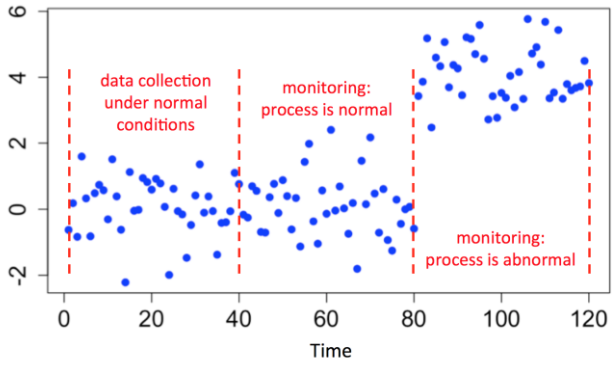

Figure 9.10: Illustration of the quality monitoring problem

In this 1-dimensional example, i.e., monitoring a process using one variable, it is easy to see that the data has an increased mean. However, when there are multiple dimensions, it is desirable to know which variables are causing the process abnormal. This so-called **fault diagnosis** problem is crucial. In this section, we discuss how we covert the problem to a classification problem where decision trees can be used for both monitoring and diagnosis.

IV.2 Theory/Method

Let $\boldsymbol{x}$ to be a p dimensional vector of the process. First, we collect a few data points under normal condition which form the **reference data**. Let $f_0(\boldsymbol{x})$ denote the distribution of the p process variables when the system is

under normal condition. Let $f_1(\boldsymbol{x})$ denote the distribution of the data points in monitoring. The goal of monitoring is to trigger an alert as quick as possible if $f_1(\boldsymbol{x})$ differs from $f_0(\boldsymbol{x})$, while at the same time, to reduce false alert when $f_1(\boldsymbol{x})$ is the same as $f_0(\boldsymbol{x})$.

One may notice that this is a typical scenario considered in multivariate quality monitoring literature. Indeed, if the variables are continuous as considered in most existing multivariate quality control charts, one may use traditional control charts like Hoteling's T^2 chart. However, these methods have difficulty handling more complex datasets, e.g., having both categorical and continuous variables. With the transformation of the monitoring problem into a stack of classification problems, we can overcome these data challenges as described in what follows.

Table 9.2: An exemplary time series dataset with 4 time points

Time	1	2	3	4
Value	2	1	3	3

Here we introduce the real-time contrasts method (RTC). The key idea of RTC method is to have a sliding window, with length of L, that includes the most recent data points to be compared with the reference data. Specifically, we label the reference data as one class, and the data points in the sliding window as another class, formulating a classification problem. The intuition is that, if the two data sets come from the same distribution, it is difficult to classify the two data sets and will result in a large classification error. But, if the training error is small, the real-time data may be different from the reference data, and will result in a small classification error. Therefore, the training error in building the classification model can be used as a metric indicating the difficulty classifying the two datasets.

Here we illustrate this intuition through a 1-dimenaional problem where the variable takes values either as 1 or 2. We also assume that, the reference data have been collected as {1,2}. The collected data for monitoring is shown in the Table 9.2.

To monitor the process, we use a window size of 2. That is say, the first monitoring action takes place at time 2 since we can collect two data points to compare with the reference data. At time 2, the 2 most recent data points are 1 and 2. The reference data set {1,2} is labeled as class 0, and the data points captured by the window with size of 2, i.e., {2,1}, are labeled as class 1. It can be seen that these two data sets are identical. Thus, the classification error rate is 0.5, which is very large.

At time 3, the sliding window now includes data points {1,3}. A classification rule "value <= 2, then class 0; else class 1" would achieve the best classification error rate as 0.25.

At time 4, the sliding window includes data points {3,3}. The same classification rule "value <= 2, then class 0; else class 1" can classify all example correctly with error rate of 0.

Through this example we can see that the classification error rate could be a **monitoring statistic** to guide the triggering of alerts. While we use a simple classification rule in this example since we only have one process variable with very simple value domains, in more complex problems, we can use other classification models such as Random forest models.

Actually, through in-depth research into this idea of directly using classification error rate as the monitoring statistic, a limitation soon reveals itself. Considering the number of data points in the monitoring window, which is L. The number of possible distinct classification error rate values are actually limited to be $L + 1$. This suggests that, while the monitoring statistic should be a continuum, the resolution of the classification error rate to reflect the continuum maybe limited if the window size is too small. This will result in gaps between the monitoring statistics, failing to capture changes that happen in the gaps which are blind zones.

As a remedy, the probability estimates of the data points can be used to replace the errors of the data points. The probability estimates are continuous indicators, while the errors are binary indicators. Then, the sum of the probability estimates from all data points in the sliding window can be used for monitoring, which is defined as:

$$p_t = \frac{\sum_{i=1}^{w} \hat{p}_1(\boldsymbol{x}_i)}{w}.$$

Here, $\boldsymbol{x}_i$ is the i^{th}data point in the sliding window, w is the window size, and $\hat{p}_1(\boldsymbol{x}_i)$ is the probability estimate of $\boldsymbol{x}_i$ belonging to $f_1(\boldsymbol{x})$. At each time point in monitoring, we can obtain a p_t. Following the tradition of control chart, we could chart the time series of p_t and observe the patterns to see if alerts should be triggered.

Besides this monitoring capacity, on the other hand, we could also use the classification model for fault diagnosis. Specifically, when the random forest is used for classification, the importance scores from random forests can be used for fault diagnosis. When process is under normal conditions and the classification errors are expected to be high, the importance scores are expected to be equal among process variables as none of them contribute to the classification problem. When process is abnormal, classification errors should be reduced, and the variables responsible for the process abnormality should now have larger importance scores. This gives us the foundation for using random forest for fault diagnosis.

Note that, under the RTC framework, the size of the sliding window is an important parameter. When the window is too long, the method requires a large number of real-time data in each monitoring epoch, which can delay the identification of abnormal patterns. In contrast, if the window is too short, the classifiers built on the small data sets may be unstable and are prone to more false positives. In our R lab, we will explore this phenomenon further.

IV.3 R Lab

We have written up the RTC method into the R function, `Monitoring()`, as shown in below. The Monitoring function takes two datasets as input, the first one being the reference data, and the second one being the real-time data points. The window size also should be provided. The function returns a few monitoring statistics for each real-time data point, and the importance score of each variable.

```
library(dplyr)
library(tidyr)
library(randomForest)
library(ggplot2)

theme_set(theme_gray(base_size = 15) )

# define monitoring function. data0: reference data; data.real.ti
me: real-time data; wsz: window size
Monitoring <- function( data0, data.real.time, wsz ){
  num.data.points <- nrow(data.real.time)
  stat.mat <- NULL
  importance.mat <- NULL

  for( i in 1:num.data.points  ){
    # at the start of monitoring, when real-time data size is sma
ll than the window size, combine the real-time data points and ra
ndom samples from the reference data to form a data set of wsz
    if(i<wsz){
      sample.size.from.reference <- wsz - i
      sample.reference <- data0[ sample(nrow(data0),sample.size.f
rom.reference,replace = TRUE), ]
      current.real.time.data <- rbind( sample.reference, data.rea
l.time[1:i,,drop=FALSE]   )
    }else{
      current.real.time.data <-  data.real.time[(i-wsz+1):i,,drop
=FALSE]
    }
    current.real.time.data$class <- 1
    data <- rbind( data0, current.real.time.data )
    colnames(data) <- c(paste0("X",1:(ncol(data)-1)),"Class")
    data$Class <- as.factor(data$Class)

    # apply random forests to the data
    my.rf <- randomForest(Class ~.,sampsize=c(wsz,wsz), data=data)

    # get importance score
```

```
    importance.mat <- rbind( importance.mat, t( my.rf$importance
) )
    # get monitoring statistics
    ooblist <- my.rf[5]
    oobcolumn=matrix(c(ooblist[[1]]),2:3)
    ooberrornormal= (oobcolumn[,3])[1]
    ooberrorabnormal=(oobcolumn[,3])[2]

    temp=my.rf[6]
    p1vote <- mean( temp$votes[,2][ (nrow(data0)+1) : nrow(data) ]
)

    this.stat <- c(ooberrornormal,ooberrorabnormal,p1vote)
    stat.mat <- rbind(stat.mat, this.stat)
  }
  result <- list(importance.mat = importance.mat,  stat.mat = sta
t.mat)
  return(result)
}
```

First, let's consider a 2-dimesional data. The reference data follow a normal distribution with mean of 0 and standard deviation of 1. The real-time data come from two distributions. The first 100 data points have the same distribution as the reference data, while the second 100 data points have the second variable changed with mean of 2. Note that, here we label the reference data with class 0 and the real-time data with class 1.

```
# data generation
# sizes of reference data, real-time data without change, and rea
l-time data with changes
length0 <- 100
length1 <- 100
length2 <- 100

# 2-dimension
dimension <- 2

# reference data
data0 <- rnorm( dimension * length0, mean = 0, sd = 1)
# real-time data with no change
data1 <- rnorm( dimension * length2, mean = 0, sd = 1)
# real-time data different from the reference data in the second
the variable
data2 <- cbind( V1 = rnorm( 1 * length1, mean = 0, sd = 1), V2 =
rnorm( 1 * length1, mean = 2, sd = 1) )
```

```
# convert to data frame
data0 <- matrix(data0, nrow = length0, byrow = TRUE) %>% as.data.
frame()
data1 <- matrix(data1, nrow = length2, byrow = TRUE) %>% as.data.
frame()
data2 <- data2 %>% as.data.frame()

# assign variable names
colnames( data0 ) <- paste0("X",1:ncol(data0))
colnames( data1 ) <- paste0("X",1:ncol(data1))
colnames( data2 ) <- paste0("X",1:ncol(data2))

# assign reference data with class 0 and real-time data with clas
s 1
data0 <- data0 %>% mutate(class = 0)
data1 <- data1 %>% mutate(class = 1)
data2 <- data2 %>% mutate(class = 1)

# real-time data consists of normal data and abnormal data
data.real.time <- rbind(data1,data2)
```

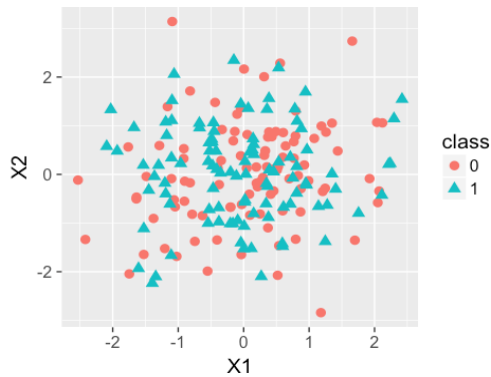

Figure 9.11: Scatterplot of the generated data points in the first 100 time points that come from the process under normal condition

Before we run the Monitor function, we show the scatterplot of the reference dataset and the dataset from the first 100 time points to obtain a visual sense of the data.

```
data.plot <- rbind( data0, data1 ) %>% mutate(class = factor(clas
s))
ggplot(data.plot, aes(x=X1, y=X2, shape = class, color=class)) +
  geom_point(size=3)
```

Then we can obtain Figure 9.11. It can be seen that the two sets of data points are similar.

We also show the scatterplot of the reference dataset and the dataset from the second 100 time points.

```
data.plot <- rbind( data0, data2 ) %>% mutate(class = factor(clas
s))
ggplot(data.plot, aes(x=X1, y=X2, shape = class, color=class)) +
  geom_point(size=3)
```

Then we can obtain Figure 9.12. It can be seen that for the real-time data set, X_2 has changed mean from 0 to 2.

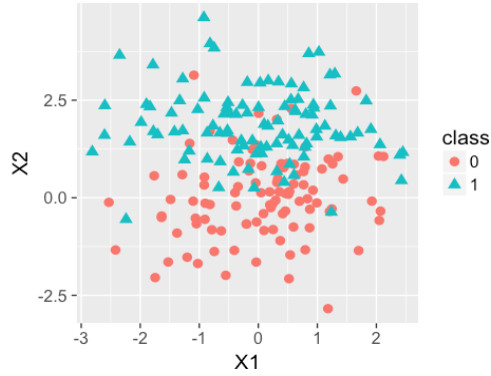

Figure 9.12: Scatterplot of the generated data points in the second 100 time points that come from the process under abnormal condition

Now we are ready to apply the RTC method. A window size of 10 is applied for monitoring. The error rates from the reference data, and the real-time data, and the probability estimates for the second class are shown in Figure 9.13 drew by the following R code.

```
wsz <- 10
result <- Monitoring( data0, data.real.time, wsz )
stat.mat <- result$stat.mat
importance.mat <- result$importance.mat

# plot different monitor statistics
stat.mat <- data.frame(stat.mat)
stat.mat$id <- 1:nrow(stat.mat)
colnames(stat.mat) <- c("error0","error1","prob","id")
stat.mat <- stat.mat %>% gather(type, statistics, error0,error1,p
rob)
ggplot(stat.mat,aes(x=id,y=statistics,color=type)) + geom_line(li
netype = "dashed") + geom_point() + geom_point(size=2)
```

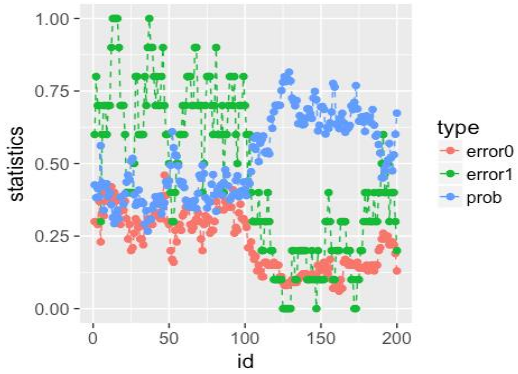

Figure 9.13: Chart of the monitoring statistics from time 1 to time 200. Three monitoring statistics are shown: `error0` denotes for the error rate in Class 0, `error1` denotes for the error rate in Class 1, and `prob` denotes for the probability estimates of the data points

As we have known that a process shift happened on X2 after the 100th data point, a good monitor statistic should significantly signal the process change after the 100th data point, the sooner the better. As we can see, there is a slight decrease for the error rates from the reference dataset, but the decrease is not substantial. The probability estimates of the data points in the reference data have more obvious increase. Similar observation can be made for the error rates from the dataset captured by the sliding window. However, the error rates from the reference data jump among a small

number of distinct values. As mentioned earlier, the number of distinct values would further reduce with a smaller sliding window. Thus, this experiment confirms that the probability estimates lead to smoother monitoring statistic and have a significant change during the process transition phase.

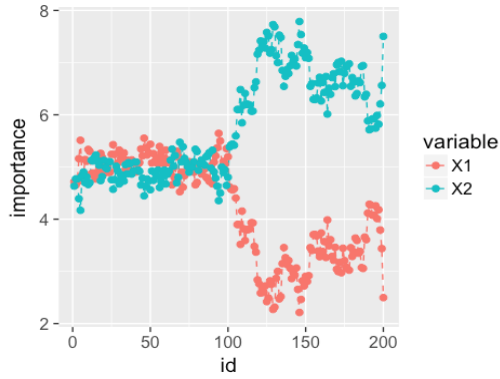

Figure 9.14: Chart of the importance score of the two process variables from time 1 to time 200

Next, let's consider fault diagnosis. Variable importance scores from the two variables from the random forests are shown in Figure 9.14 drew by the following R code.

```
# plot importance scores for diagnosis
importance.mat <- data.frame(importance.mat)
importance.mat$id <- 1:nrow(importance.mat)
colnames(importance.mat) <- c("X1","X2","id")
importance.mat <- importance.mat %>% gather(variable, importance,
X1,X2)

ggplot(importance.mat,aes(x=id,y=importance,color=variable)) + ge
om_line(linetype = "dashed")  + geom_point(size=2)
```

From Figure 9.14, we can see that the importance scores of X2 significantly increase after the 100th point. This indicates that X2 plays an

important role in improving the classification and may be responsible for the process change.

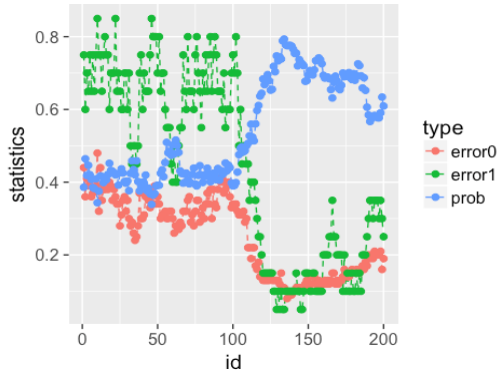

Figure 9.15: Chart of the monitoring statistics from time 1 to time 200 (window size increases to 20)

As we have mentioned, the size of the window for monitoring is an important parameter. Here, to see its effect, the window size is increased to 20. The monitoring statistics and importance scores are re-plotted in Figure 9.15 and Figure 9.16.

```
# change window size to 20
wsz <- 20
result <- Monitoring( data0, data.real.time, wsz )
stat.mat <- result$stat.mat
importance.mat <- result$importance.mat

# plot different monitor statistics
stat.mat <- data.frame(stat.mat)
stat.mat$id <- 1:nrow(stat.mat)
colnames(stat.mat) <- c("error0","error1","prob","id")
stat.mat <- stat.mat %>% gather(type, statistics, error0,error1,p
rob)
ggplot(stat.mat,aes(x=id,y=statistics,color=type))+ geom_line(lin
etype = "dashed") + geom_point() + geom_point(size=2)
```

```
# plot importance scores for diagnosis
importance.mat <- data.frame(importance.mat)
importance.mat$id <- 1:nrow(importance.mat)
colnames(importance.mat) <- c("X1","X2","id")
importance.mat <- importance.mat %>% gather(variable, importance,
X1,X2)

ggplot(importance.mat,aes(x=id,y=importance,color=variable)) + ge
om_line(linetype = "dashed")  + geom_point(size=2)
```

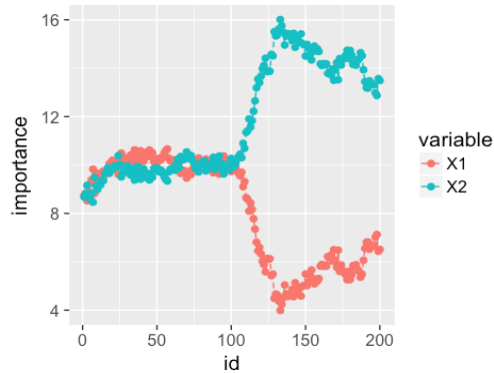

Figure 9.16: Chart of the importance score of the two process variables from time 1 to time 200 (window size increases to 20)

Compared to the previous results with window size of 10, the monitoring statistics on the changed real-time data points have a clearer increase from the un-changed real-time data. Similarly, the increase of the importance score of X2 is stronger. However, the change of the monitoring statistics and importance scores is slightly slower than the change with a smaller window. Therefore, a large window size can lead to more confident alert, but with a slower speed.

Now, let's change the window size to 5. The monitoring statistics and importance scores are re-plotted in Figure 9.17 and Figure 9.18.

```
# change window size to 5
wsz <- 5
result <- Monitoring( data0, data.real.time, wsz )
stat.mat <- result$stat.mat
importance.mat <- result$importance.mat

# plot different monitor statistics
stat.mat <- data.frame(stat.mat)
stat.mat$id <- 1:nrow(stat.mat)
colnames(stat.mat) <- c("error0","error1","prob","id")
stat.mat <- stat.mat %>% gather(type, statistics, error0,error1,p
rob)
ggplot(stat.mat,aes(x=id,y=statistics,color=type))+ geom_line(lin
etype = "dashed") + geom_point() + geom_point(size=2)
```

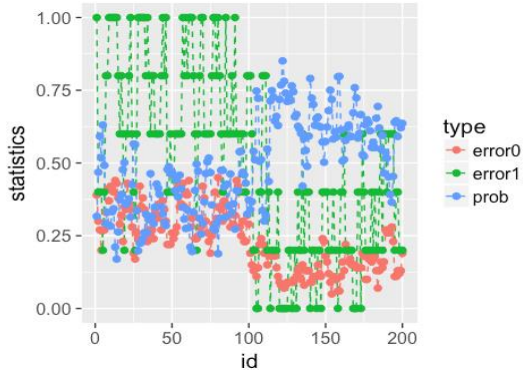

Figure 9.17: Chart of the monitoring statistics from time 1 to time 200 (window size decreases to 5)

```
# plot importance scores for diagnosis
importance.mat <- data.frame(importance.mat)
importance.mat$id <- 1:nrow(importance.mat)
colnames(importance.mat) <- c("X1","X2","id")
importance.mat <- importance.mat %>% gather(variable, importance,
X1,X2)

ggplot(importance.mat,aes(x=id,y=importance,color=variable)) + ge
om_line(linetype = "dashed")  + geom_point(size=2)
```

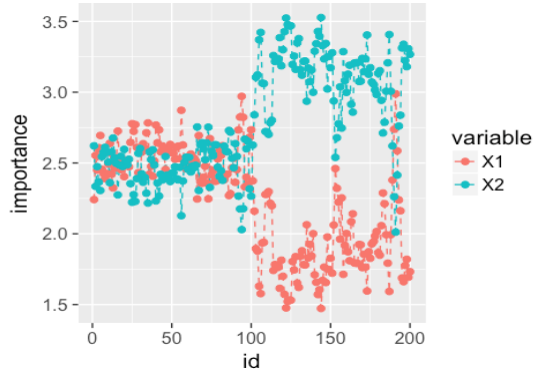

Figure 9.18: Chart of the importance score of the two process variables from time 1 to time 200 (window size decreases to 5)

Obviously, the monitoring statistic seems to be more noisy, producing less confident patterns, but it raises quicker alert at the 101th time point.

Now, let's consider a 10-dimensional dataset. In this example, two variables' means change from 0 to 2 in the second 100 data points.

```
# 10-dimensions, with 2 variables being changed from the normal c
ondition
dimension <- 10
wsz <- 5
# reference data
data0 <- rnorm( dimension * length0, mean = 0, sd = 1)
# real-time data with no change
data1 <- rnorm( dimension * length1, mean = 0, sd = 1)
# real-time data different from the reference data in the second
the variable
data2 <- c( rnorm( (dimension - 2) * length2, mean = 0, sd = 1),
rnorm( (2) * length2, mean = 20, sd = 1))


# convert to data frame
data0 <- matrix(data0, nrow = length0, byrow = TRUE) %>% as.data.
frame()
data1 <- matrix(data1, nrow = length1, byrow = TRUE) %>% as.data.
frame()
data2 <- matrix(data2, ncol = 10, byrow = FALSE) %>% as.data.fram
e()
```

```
# assign reference data with class 0 and real-time data with clas
s 1
data0 <- data0 %>% mutate(class = 0)
data1 <- data1 %>% mutate(class = 1)
data2 <- data2 %>% mutate(class = 1)

# real-time data consists of normal data and abnormal data
data.real.time <- rbind(data1,data2)
```

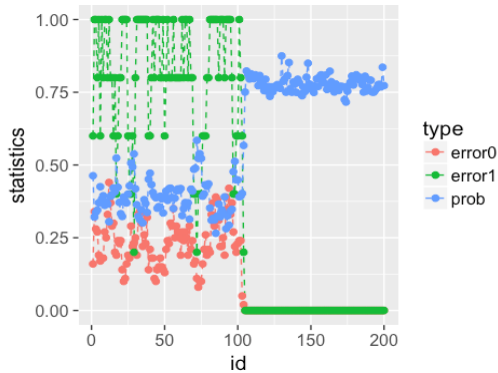

Figure 9.19: Chart of the monitoring statistics from time 1 to time 200 (window size is 10)

10 dimensions are difficult to visualize and monitor. Encouragingly, the monitoring statistics shown in Figure 9.19 shows that the RTC method is still capable of capturing the changes. It is clear in Figure 9.19 that all the monitoring statistics change after the 101th time point, and the importance scores in Figure 9.20 also indicate the change is due to X9 and X10. The following R codes generated Figure 9.19.

```
result <- Monitoring( data0, data.real.time, wsz )
stat.mat <- result$stat.mat
importance.mat <- result$importance.mat

# plot different monitor statistics
stat.mat <- data.frame(stat.mat)
stat.mat$id <- 1:nrow(stat.mat)
```

```
colnames(stat.mat) <- c("error0","error1","prob","id")
stat.mat <- stat.mat %>% gather(type, statistics, error0,error1,p
rob)
ggplot(stat.mat,aes(x=id,y=statistics,color=type))+ geom_line(lin
etype = "dashed") + geom_point() + geom_point(size=2)
```

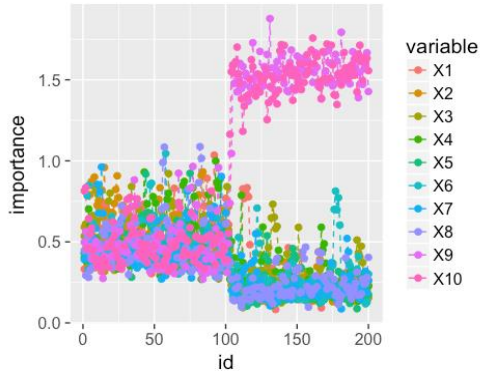

Figure 9.20: Chart of the importance score of the ten process variables from time 1 to time 200 (window size is 10)

The following R codes generated Figure 9.20.

```
# plot importance scores for diagnosis
importance.mat <- data.frame(importance.mat)
importance.mat$id <- 1:nrow(importance.mat)
# colnames(importance.mat) <- c("X1","X2","id")
importance.mat <- importance.mat %>% gather(variable, importance,
X1:X10)
importance.mat$variable <- factor( importance.mat$variable, level
s = paste0( "X", 1:10  ) )
# levels(importance.mat$variable) <- paste0( "X", 1:10  )
ggplot(importance.mat,aes(x=id,y=importance,color=variable)) + ge
om_line(linetype = "dashed")  + geom_point(size=2)
```

IV.4 Remarks

There is an important technical aspect to implement the RTC method. Realistically, to capture the normal conditions of a process, many data

points are needed to define the reference dataset. On the other hand, to capture process changes more sensitively, the window size for online monitoring should not be too large. Thus, comparing with the sample size in the reference dataset, an effective window size is typically substantially smaller than the size of reference data. Therefore, this usually leads to a highly imbalanced classification problem.

As a remedy, the random forest model can handle class imbalance by under-sampling the reference data to be the same size as the sliding window data for each tree. That is to say, instead of sampling uniformly the data points for each tree (as shown in the left figure in Figure 9.21), the same number of samples are selected from the reference data and the sliding window data (as shown in the right figure in Figure 9.21).

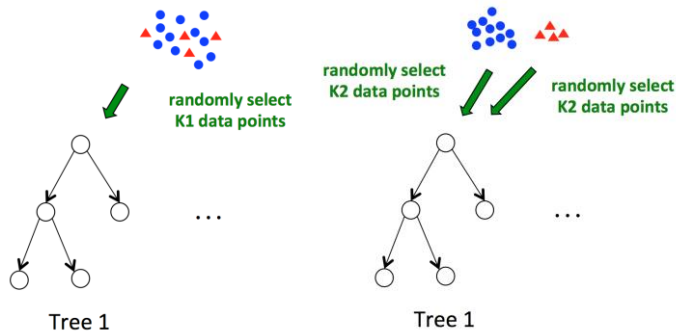

Figure 9.21: Regular sampling (left) and purposeful equal sampling (right) by random forest to grow its trees

IV. Exercises

Data analysis

1. Find 10 regression datasets from the UCI data repository or R datasets. Build kernel regression model, KNN regression model, and linear regression models. Conduct model selection and validation. Use cross-validation to select the best models. Compare the models and comment on their applications on these datasets.

2. Identify 3 datasets from the UCI data repository or R datasets that the heteroscedasticity may be a problem. Build the conditional variance regression model on these datasets. Compare the conditional variance regression model with linear regression model.

Programming

3. Use bootstrap to show the conditional variance regression model is significantly better than linear regression model on the datasets you have selected that have the problem of heteroscedasticity. Write your own R script to implement this idea. Make sure that, for datasets that have no concern of heteroscedasticity, your approach would not always advocate for the use of conditional variance regression model.
4. Implement the tree-based system monitoring method on a high-dimensional dataset with more than 100 variables. You can simulate such a dataset following the R lab in this chapter. See if the tree-based system monitoring method can lead to quick detetion of process change, and accurate fault diagnosis (i.e., make sure in your data, only a few variables are responsible for the process change).