Portfolio 2: Adaptive Kernel Regression Smoothing with Rcpp

Kieran Morris

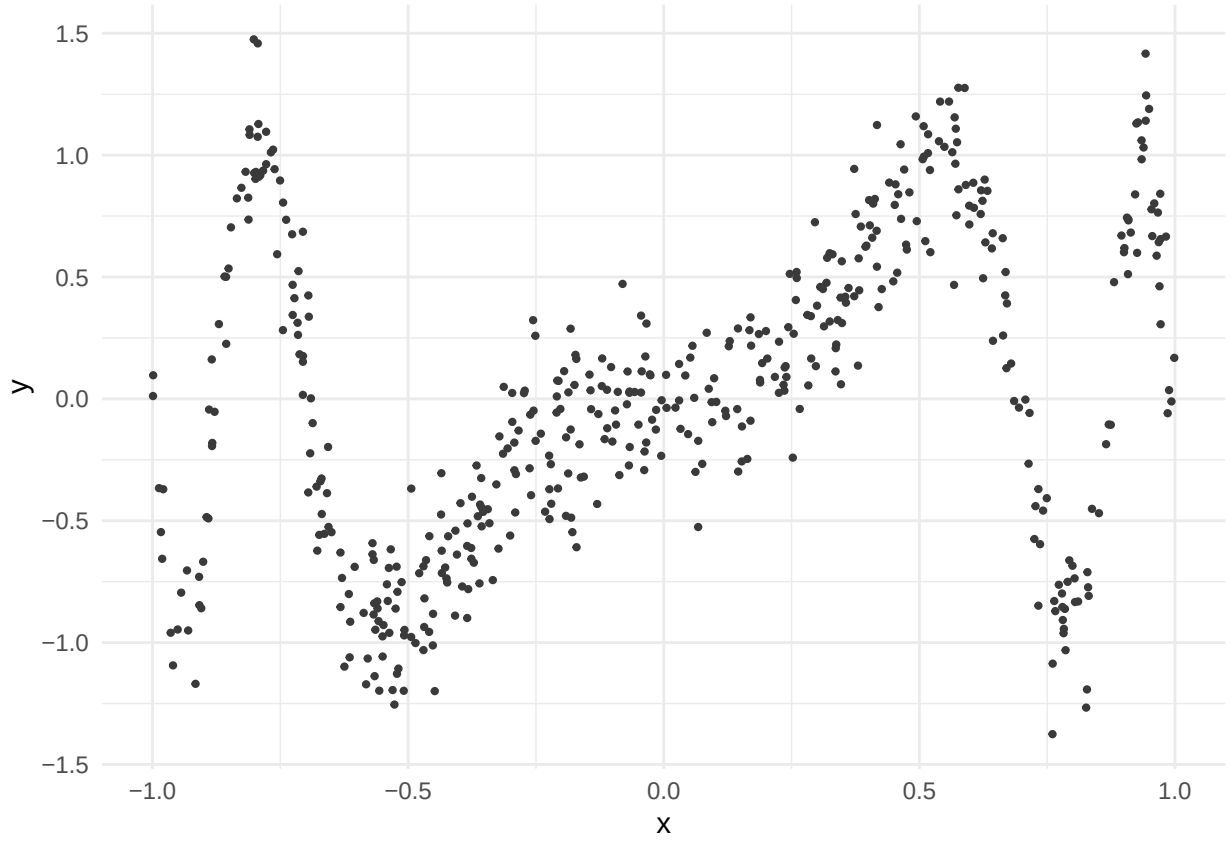
The Model

In this portfolio we will consider the following model:

$$y_i = \sin(a\pi x_i^3) + z_i, \quad z_i \sim N(0, \sigma^2)$$

where a and σ are hyperparameters. We will use adaptive kernel regression to estimate this function, imagining we don't know the function itself. We arbitrarily choose $\sigma = 0.1$ and $a = 3$, lets simulate this in `R` and see what it looks like.

```
library(ggplot2
)
set.seed(123)


n <- 500
x <- runif(n, -1, 1)
y <- sin(3 * pi * x^3) + rnorm(n, 0, 0.2)

df <- data.frame(x = x, y = y)

ggplot(df, aes(x = x, y = y)) + geom_point(size = 0.8,color = "#3a3a3a") + theme_minimal()
```



Definitely looks doable, lets begin by implimenting the adaptive kernel regression in R.

Adaptive Kernel Regression R

Consider the following kernel smoother:

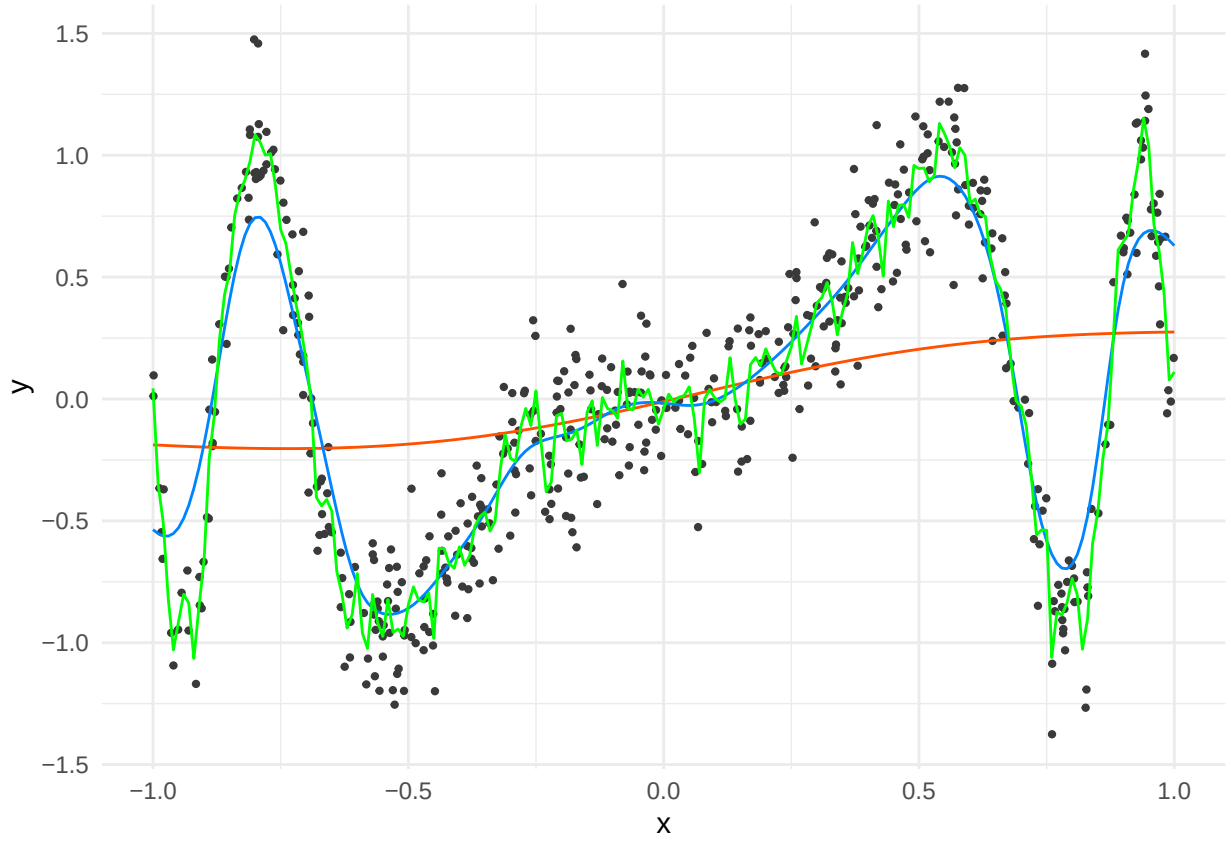
$$\hat{f}(x) = \frac{\sum_{i=1}^{n} K_\lambda(x, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x, x_i)}$$

where K_λ is the RBF kernel. With variance λ^2. Let's implement this in R.

```
meanKRS <- function(y, x, x0, lam){

 n <- length(x)
 n0 <- length(x0)

 out <- numeric(n0)
 for(ii in 1:n0){
  out[ii] <- sum( dnorm(x, x0[ii], lam) * y ) / sum( dnorm(x, x0[ii], lam) )
 }

 return( out )
}
```

Let's try this smoothing out with a few different values of λ.

```
x0 <- seq(-1, 1, 0.01)
y0 <- meanKRS(y, x, x0, 0.5)
y1 <- meanKRS(y, x, x0, 0.05)
y2 <- meanKRS(y, x, x0, 0.005)

df0 <- data.frame(x = x0, y = y0)
df1 <- data.frame(x = x0, y = y1)
df2 <- data.frame(x = x0, y = y2)

ggplot(df, aes(x = x, y = y)) +
    geom_point(size = 0.8,color = "#3a3a3a") +
    theme_minimal() +
    geom_line(data = df0, aes(x = x, y = y), color = "#ff5100") +
    geom_line(data = df1, aes(x = x, y = y), color = "#0084ff") +
    geom_line(data = df2, aes(x = x, y = y), color = "#00ff00")
```



I think the blue is my favourite!

Adaptive Kernel Regression C++

Now too be fair that was quite quick, but C++ is so much fun lets give it a go by using `Rcpp`. Below we use `ccpFunction` to use inline C++ code for smoothing.

```
library(Rcpp)

cppFunction('NumericVector meanKRS_C(NumericVector y, NumericVector x, NumericVector x0, double lam){

 int n = x.size();
 int n0 = x0.size();

 NumericVector out(n0);

 for(int ii = 0; ii < n0; ii++){
  double num = 0;
  double den = 0;
  for(int jj = 0; jj < n; jj++){
   num += exp(-pow(x[jj] - x0[ii], 2) / (2 * pow(lam, 2))) * y[jj];
   den += exp(-pow(x[jj] - x0[ii], 2) / (2 * pow(lam, 2)));
  }
  out[ii] = num / den;
 }

 return out;
}')
```

Below we check to make sure the output of our results are the same.

```
# Smoothing with C++

z0 <- meanKRS_C(y, x, x0, 0.5)
z1 <- meanKRS_C(y, x, x0, 0.05)
z2 <- meanKRS_C(y, x, x0, 0.005)
#Verify the results are the same
all.equal(y1, z1)
```

```
## [1] TRUE
```

Unfortunately the plots would look the exact same so we will skip those, lets compare their performance with the `microbenchmark` package.

```
library(microbenchmark)

microbenchmark("R" = meanKRS(y, x, x0, 0.005),
  "C++" = meanKRS_C(y, x, x0, 0.005))
```

```
## Unit: milliseconds
##  expr      min       lq     mean   median       uq      max neval
##     R 3.368155 3.782932 3.881390 3.796416 3.838220 7.350857   100
##   C++ 1.345164 1.352809 1.358034 1.356059 1.360326 1.398694   100
```

Pretty quick! Nice job C++.

Cross Validation in R and C++

We will create two cross validation functions, one in R and one in C++, in order to find the optimal λ value.

```r
# Cross Validation in R
cvKRS <- function(y, x, lam, k){
  n <- length(x)

  fold_size <- n / k
  mse <- 0

  for (i in 1:k) {
    test_indices <- ((i-1)*fold_size + 1):(i*fold_size)
    train_indices <- setdiff(1:n, test_indices)

    yhat <- meanKRS(y[train_indices], x[train_indices], x[test_indices], lam)
    mse <- mse + sum((y[test_indices] - yhat)^2)
  }

  return(mse / n)
}

# Cross Validation in C++
cppFunction('double cvKRS_C(NumericVector y, NumericVector x, double lam, int k){
  int n = x.size();

  int fold_size = n / k;
  double mse = 0;

  for(int i = 0; i < k; i++){
    int start = i * fold_size;
    int end = start + fold_size;

    for(int j = start; j < end; j++){
      double yhat = 0;
      double num = 0;
      double den = 0;
      for(int l = 0; l < n; l++){
        if(l < start || l >= end){
          num += exp(-pow(x[l] - x[j], 2) / (2 * pow(lam, 2))) * y[l];
          den += exp(-pow(x[l] - x[j], 2) / (2 * pow(lam, 2)));
        }
      }
      yhat = num / den;
      mse += pow(y[j] - yhat, 2);
    }
  }

  return mse / n;
}')
```

Verify that they are the same:

```r
all.equal(cvKRS(y, x, 0.005, 5), cvKRS_C(y, x, 0.005, 5))
```

```
## [1] TRUE
```

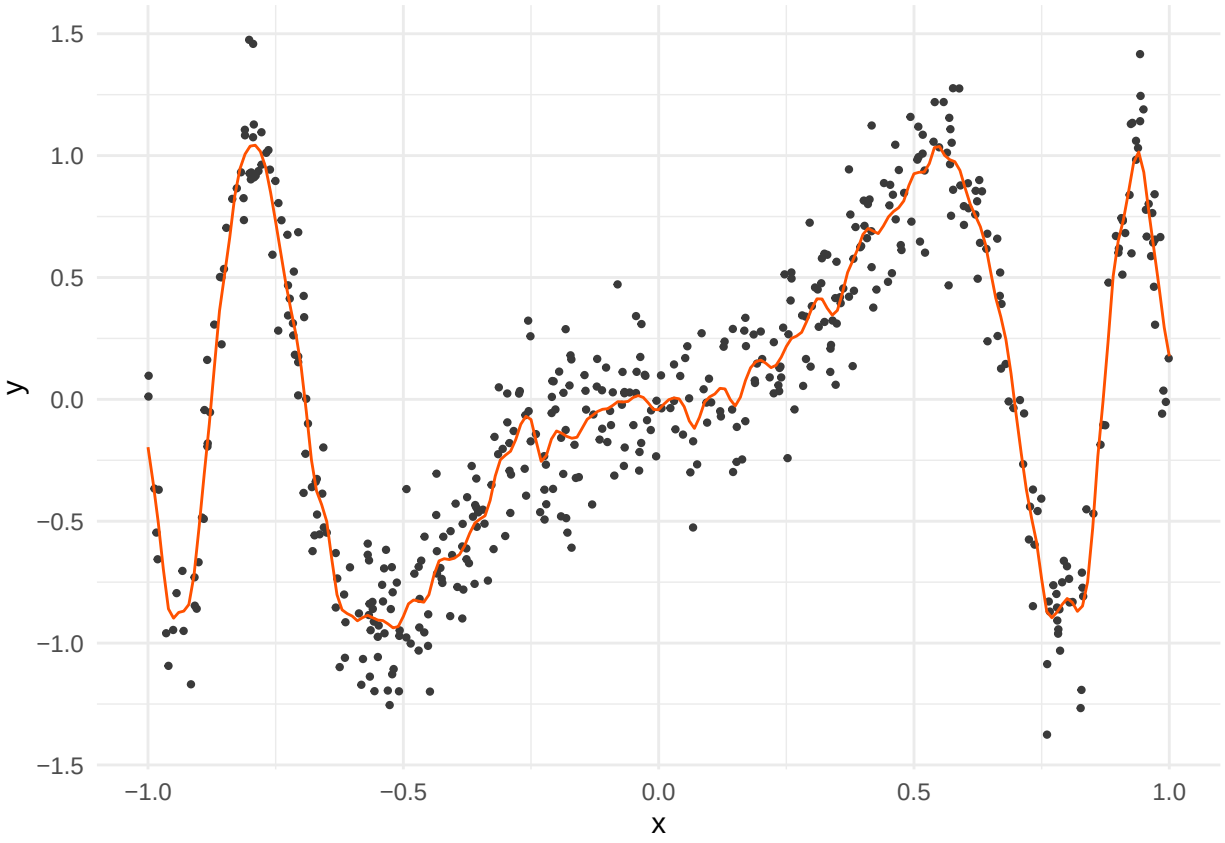
Now we will use these functions to find the optimal λ value.

```
lams <- seq(0.001, 0.5, 0.001)

cv_vals <- sapply(lams, function(lam) cvKRS_C(y, x, lam, 5))

optimal_lam <- lams[which.min(cv_vals)]

optimal_lam
```

```
## [1] 0.013
```

Let's plot the fit with this optimal λ value.

```
y_opt <- meanKRS_C(y, x, x0, optimal_lam)

df_opt <- data.frame(x = x0, y = y_opt)

ggplot(df, aes(x = x, y = y)) +
    geom_point(size = 0.8,color = "#3a3a3a") +
    theme_minimal() +
    geom_line(data = df_opt, aes(x = x, y = y), color = "#ff5100")
```



What a beauty! We used the C++ version of cross validation, since we assumed it would be quicker, but lets use `microbenchmark` again to check.

```r
microbenchmark("R" = cvKRS(y, x, 0.005, 5),
  "C++" = cvKRS_C(y, x, 0.005, 5))
```

```
## Unit: milliseconds
##  expr      min       lq     mean   median       uq       max neval
##     R 7.165759 7.197638 7.413231 7.221350 7.259007 10.232671   100
##   C++ 3.037250 3.056694 3.080913 3.068822 3.078863  3.375651   100
```

Quite a lot faster I'd say!