Rcpp Portfolio

Rachel Wood

2023-03-27

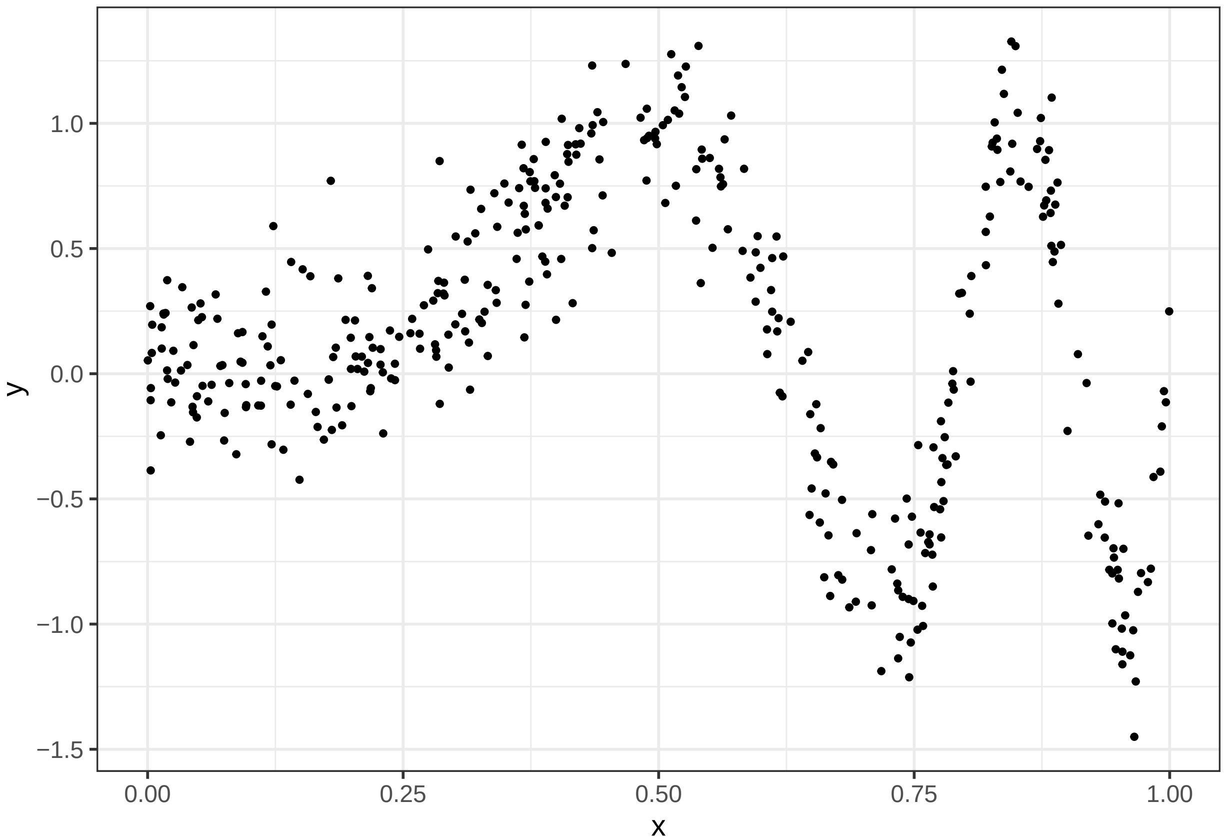
For this portfolio, we use Rcpp to fit an adaptive kernel smoothing regression model.

We first generate data according to the model

$$y_i = \sin(\alpha \pi x^3) + z_i \quad \text{with} \quad z_i \sim \mathcal{N}(0, \sigma^2)$$

In this case we take $\alpha = 4$ and $\sigma = 0.2$.

```r
library(dplyr)
library(ggplot2)
n <- 400
alpha <- 4
sigma <- 0.2

x <- runif(n)
y <- sin(alpha * pi * x^3) + rnorm(n, sd = sigma)

data <- tibble(x = x, y =y)

ggplot(data = data, aes(x, y))+
  geom_point(size = 0.8)
```



The Kernel Smoother

We model $\mu(x) = \mathbb{E}(y|x)$ by

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

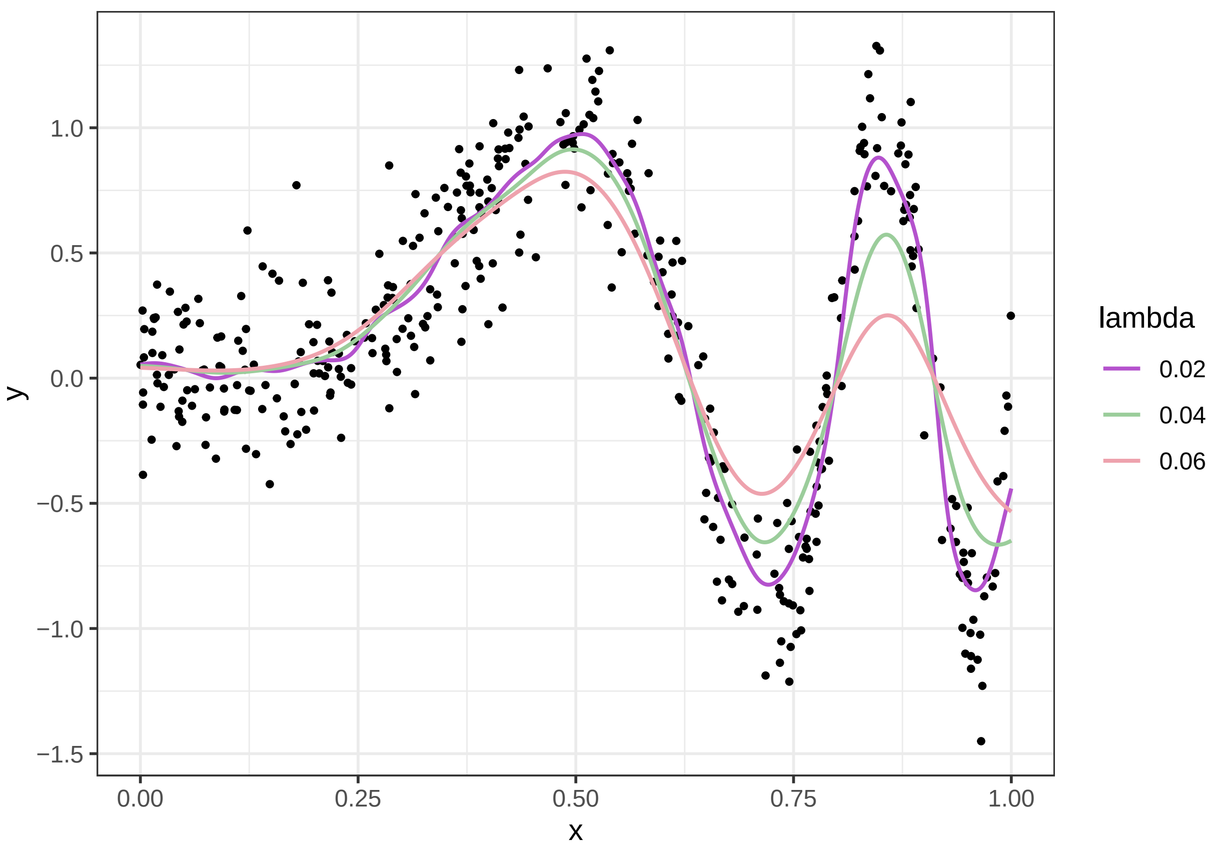
where we take κ_λ to be a Gaussian kernel with variance λ^2.

We implement this with the following function:

```
meanKRS <- function(x, y, xnew, lambda){
  n <- length(x)
  nnew <- length(xnew)

  mu <- numeric(nnew)

  for (i in 1:nnew){
    mu[i] <- sum(dnorm(x,xnew[i], lambda)*y)/ sum(dnorm(x,xnew[i], lambda))
  }

  return(mu)
}
```

We can now compare the fits for different values of λ:

```r
library(tidyr)
xnew <- seq(0,1, length.out = 1000)

smooth_large <- meanKRS(x, y, xnew, lambda = 0.06)
smooth_medium <- meanKRS(x, y, xnew, lambda = 0.04)
smooth_small <- meanKRS(x, y, xnew, lambda = 0.02)

plot_data <- tibble(x = xnew) %>%
  mutate("0.06" = smooth_large,
         "0.04" = smooth_medium,
         "0.02" = smooth_small) %>%
  pivot_longer(cols = c("0.06","0.04","0.02"),
               names_to = "lambda",
               values_to = "fitted") %>%
  mutate(lambda = as.factor(lambda))

ggplot() +
  geom_point(data = data,
             aes(x, y), size = 0.8) +
  geom_line(data = plot_data,
            aes(x, fitted, color = lambda), linewidth = 0.7)
```



We now use Rcpp to write a C++ version of `meanKRS()`:

```r
library(Rcpp)
```

```
#include <Rcpp.h>
#include <Rmath.h>
using namespace Rcpp;

// [[Rcpp::export]]

NumericVector meanKRS_Rcpp(const NumericVector x, const NumericVector y, const
↪  NumericVector xnew, const double lambda) {
  int n = x.size();
  int nnew = xnew.size();

  NumericVector mu(nnew);

  for (int i = 0; i < nnew; i++){
    mu[i] = sum(dnorm(x,xnew[i], lambda)*y)/ sum(dnorm(x,xnew[i], lambda));
  }

  return mu;
}
```

We check that this function produces the same output as the R version,

```
max(meanKRS(x, y, xnew, lambda = 0.06) - meanKRS_Rcpp(x, y, xnew, lambda = 0.06))
```

```
## [1] 2.220446e-15
```

and compare the performance of the two functions using the `microbenchmark()` function:

```
library(microbenchmark)
microbenchmark("R" = meanKRS(x, y, xnew, lambda = 0.06),
               "Rcpp" = meanKRS_Rcpp(x, y, xnew, lambda = 0.06))
```

```
## Unit: milliseconds
##   expr      min       lq     mean   median       uq      max neval
##      R 18.89753 19.74762 20.28494 19.86958 20.19374 25.80065   100
##   Rcpp 12.05555 12.92683 13.06372 13.00527 13.09450 17.98556   100
```

Cross-Validation

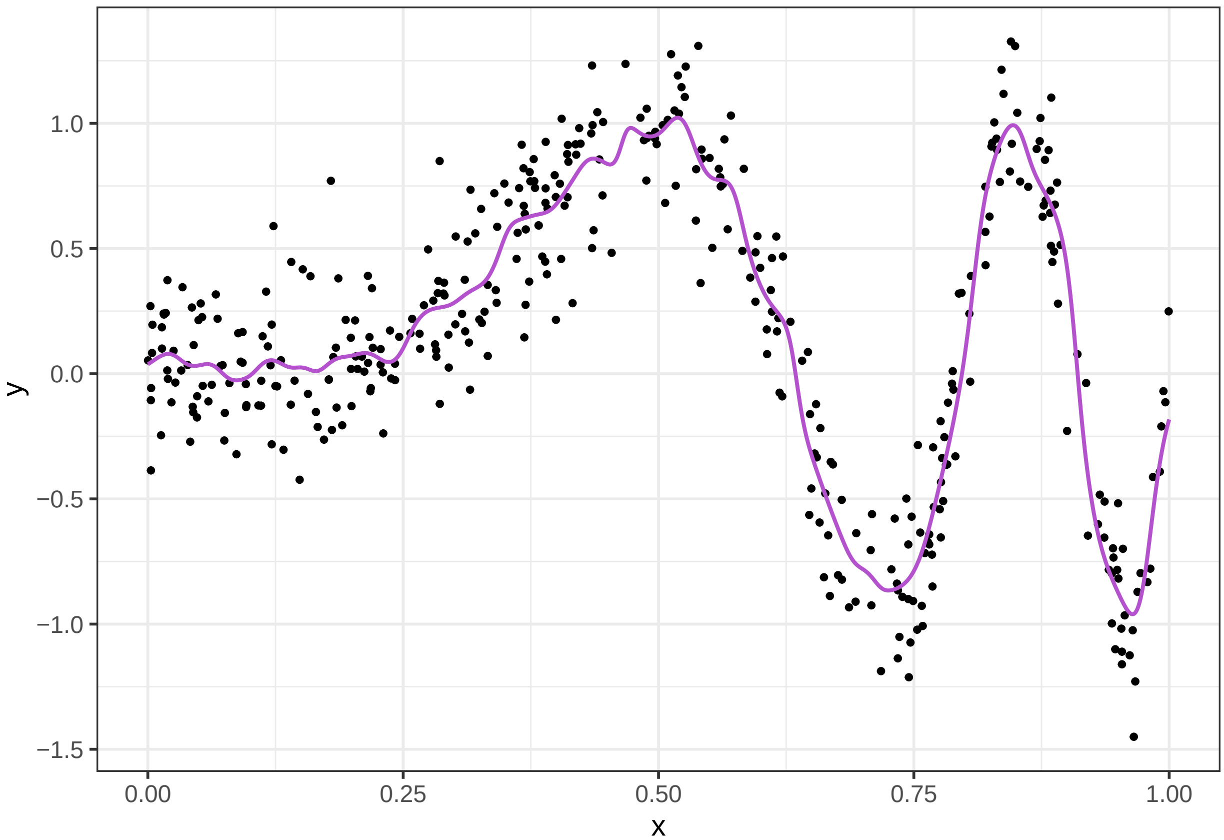
We now implement a cross-validation procedure for finding the optimal λ, using the mean squared error of the test set as the metric for determining the fit of our model. We first write the R version of this function:

```
mse_lambda <- function(log_lambda, x, y, x_new, y_new){
  lambda <- exp(log_lambda)

  fitted <- meanKRS(x, y, x_new, lambda)
  return(sum((fitted - y_new)^2))
}
```

```r
lambda_cv <- function(x, y, groups){
  n <- length(x)


  lambdas <- numeric(nfolds)
  mse <- numeric(nfolds)

  for (i in 1:nfolds){
    x_train <- x[groups != i]
    y_train <- y[groups != i]

    x_test <- x[groups == i]
    y_test <- y[groups == i]

    solution <- optim(par = 0.02, fn = mse_lambda, x = x_train, y = y_train, x_new =
↪   x_test, y_new =  y_test, method = "BFGS")
    lambdas[i] <- exp(solution$par)
    mse[i] <- solution$value

  }

 min_ind <- which.min(mse)
 return(lambdas[min_ind])
}
```

We now plot the smooth for the returned value λ to see if this seems reasonable:

```r
nfolds <- 5
groups <- sample(rep(1:nfolds, length.out = n), size = n)
hat_lambda <- lambda_cv(x, y, groups)
opt_smooth <- meanKRS(x, y, xnew, hat_lambda)
opt_data <- tibble(xnew = xnew,
         fitted = opt_smooth)
ggplot() +
  geom_point(data = data, aes(x, y), size = 0.8) +
  geom_line(data = opt_data, aes(x = xnew,y = fitted),
            color = "mediumorchid3", linewidth = 0.7)
```



We now write the equivalent function in Rcpp:

```
// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::depends(roptim)]]

#include <cmath>
#include <cstddef>

#include <algorithm>

#include <RcppArmadilloExtensions/sample.h>
#include <RcppArmadillo.h>
#include <roptim.h>
#include <functional>
using namespace Rcpp;
using namespace arma;
using namespace roptim;


#include <roptim.h>
#include <functional>
using namespace Rcpp;
using namespace arma;
using namespace roptim;


NumericVector meanKRS_Rcpp_I(const NumericVector x, const NumericVector y, const
↪  NumericVector xnew, const double lambda) {
```

```
  int n = x.size();
  int nnew = xnew.size();

  NumericVector mu(nnew);

  for (int i = 0; i < nnew; i++){
    mu[i] = sum(dnorm(x,xnew[i], lambda)*y)/ sum(dnorm(x,xnew[i], lambda));
  }

  return mu;
}

double mse_lambda(const double lambda, const NumericVector x,const NumericVector y,const
↪  NumericVector x_new, const NumericVector y_new){


  NumericVector fitted = meanKRS_Rcpp_I(x, y, x_new, lambda);
  return sum(pow(fitted - y_new, 2));
}


NumericVector x_train, y_train, y_test, x_test;


// [[Rcpp::export]]


NumericVector lambda_cv_Rcpp(const NumericVector x, const NumericVector y, const
↪  NumericVector groups) {

    int n = x.size();

    int nfolds = unique(groups).size();

    NumericVector lambdas(nfolds);
    NumericVector mse(nfolds);

    for (int i =1; i <= nfolds; i++){
        x_train = x[groups != i];
        y_train = y[groups != i];

        x_test = x[groups == i];
        y_test = y[groups == i];


    class Mse : public Functor {
      public:
      double operator()(const arma::vec& log_lambda) override {
        double lambda = exp(log_lambda[0]);
        // std::function<double(const double log_lambda)> mse = { mse_lambda(log_lambda,
        ↪  x_train, y_train, x_test, y_test)};
        return  mse_lambda(lambda, x_train, y_train, x_test, y_test);
      }
```

```
    };

    Mse fun;
    Roptim<Mse> opt("BFGS");

    arma::vec initial = {0.02};
    opt.minimize(fun, initial);


    arma::vec par = opt.par();
    lambdas[i] = par[0];
    mse[i] = opt.value();

  }

  int min_ind = which_min(mse);

  return lambdas;

}
```

```
hat_lambda <- lambda_cv_Rcpp(x, y, groups)
```

Lambda as a function of x

We can see merely from looking at the plot, the shape of the function changes at approximately $x = 0.5$, and so these two sections of the function will need different values of λ. We address this by modelling $\lambda = \lambda(x)$. We do this by fitting the model as before for a fixed λ (for this we can use the cross-validated value of λ) and consider the residuals $r_1, \dots, r_n$.

We can then model these under another KRS with the same λ - producing estimates of the absolute values of the residuals $\hat{v}_1, \dots, \hat{v}_n$. We can then fit

$$\hat{\mu}(x)$$