Nonparametric Statistics Final Assignment

Danyu Zhang & Daniel Alonso

March 30, 2021

Exercises

Category A: Problem 6

- **Exercise 5.11**. The *challenger.txt* dataset contains dataset contains information regarding the state of the solidrocket boosters after launch for 23 shuttle flights prior the Challenger launch. Each row has, among others, the variables *fail.field* (indicator of whether there was an incident with the O-rings), *nfail.field* (number of incidents with the O-rings), and *temp* (temperature in the day of launch,measured in degrees Celsius).

a) Fit a local logistic regression (first degree) for *fails.field* ~ *temp*, for three choices of bandwidths: one that oversmooths, another that is somehow adequate, and another that undersmooths. Do the effects of *temp* on *fails.field* seem to be significant?

b) Obtain $\hat{h}_{LCV}$ and plot the LCV function with a reasonable accuracy.

c) Using $\hat{h}_{LCV}$, predict the probability of an incident at temperatures -0.6 (launch temperature of the Challenger) and 11.67 (specific recommendation by the vice president of engineers).

d) What are the local odds at -0.6 and 11.67? Show the local logistic models about these points, in spirit of Figure 5.1, and interpret the results.

Category B: Problem 4

- **Exercise 4.9**. Perform the following tasks:

a) Code your own implementation of the local cubic estimator. The function must take as input the vector of evaluation points x, the sample *data*, and the bandwidth h. Use the normal kernel. The result must be a vector of the same length as x containing the estimator evaluated at x.

We have implemented the local polynomial estimator for any $0 \leq p < 8$ (however, $p = 3$ is the default parameter as we are asked to implement the local cubic estimator).

Unfortunately, for large datasets + evaluation points (assuming both are of the same size as we are asked in part b of this problem) the function is quite inefficient ($> O(n^3)$).

Given that our focus was to simply implement it and not necessarily be particularly efficient, we have gone for the less efficient form of the local polynomial estimator (as described in this section of the Nonparametric Statistics notes).

Parameters

- **x**: vector of evaluation points
- **data**: sample dataset where the first column (*data[,1]* in R) are the *predictors* and the second column (*data[,2]* in R) are the *response*.
- **h**: bandwidth

Algorithm

The function goes through the following process in order to output the estimation:

1- Values are initialized

- The resulting vector (*result*, initialized as an empty vector)
- The predictors (*predictors*, taken from the first column of the *data* matrix)
- The response (Y, taken from the second column of the *data* matrix)
- A $(p+1) \times 1$ vector with its first entry as 1 and the rest as zero (*e_1*)

The X matrix is also initialized in this step and corresponds to:

$$\mathbf{X} := \begin{pmatrix} 1 & X_1 - x & \cdots & (X_1 - x)^p \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_n - x & \cdots & (X_n - x)^p \end{pmatrix}_{n \times (p+1)}$$

This is initialized as an empty list, where each entry in the list corresponds to the matrix X corresponding to the k-th value of the input vector x (values the user wants to predict).

2- Main loop steps

- k iterations are initiated (one per element in the input x vector)
- The element k of the X list of matrices X is created as an empty matrix with dimensions $n \times (p+1)$, where $n =$ amount of values in the predictor column of data and $p =$ degree (by default 3).
- i iterations are initiated (one iteration per row in $X[[k]]$) and inside j iterations (one iteration per degree + 1), the dimensions of $X[[k]]$ are used to determine the amount of iterations

- Inside the innermost loop (j-loop) we subtract the k-th element in the input vector from the i-th element in the predictor vector ($predictors[i]$) and we raise the result to the power of $j-1$ (as loops go, this corresponds to $p-3,\ldots,p$, for the case p=3)

3- Weights loop

- For the weights loop we use the normal kernel (*dnorm* in R) and apply it to the difference of the i-th predictor ($predictors[i]$) and the k-th element in the input vector ($x[k]$)

4- Result

- We apply the *diag* R function to the *weights* vector to make it a diagonal matrix with the values of *weights* in the main diagonal.
- We finally add the k-th value (result of $\mathbf{e}_1'(\mathbf{X'WX})^{-1}\mathbf{X'WY}$ for the k-th x)
- Return the result vector

Function code with annotations

```r
lce <- function(x, data, h, p=3) {
    ## INITIALIZING VARIABLES
    # Resulting vector initilization
    result <- c()

    # Predictors initialization (copied from data col 1)
    predictors <- data[,1]

    # Response initialization (copied from data col 2)
    Y <- data[,2]

    # e_1 (vector of size p+1 x 1 of zeros with the exception of the first entry)
    e_1 <-  matrix(c(c(1), rep(0,p)),nrow=p+1,ncol=1)

    # X matrix
    X <- list()

    ## MAIN LOOP
    for (k in 1:length(x)){
        # k-th matrix of the list of X, whose size will correspond with the length
        # of the values the user wants to predict (length of x)
        X[[k]] <- matrix(,nrow=length(predictors),ncol=p+1)

        # Filling up the k-th X matrix entry by entry
        for (i in 1:dim(X[[k]])[1]) {
            for (j in 1:dim(X[[k]])[2]) {
                # the k-th X matrix's i,j-th positions are filled
                # subtracting the i-th predictor's value with the k-th
                # element in the values the user wants to evaluate
                # all to the power j-1, where the first value obtained
                # is a 0 in the exponent, rendering the first column of each
                # X matrix a column of ones, and the last one to the power
                # of the p defined by the user (default p=3)
                X[[k]][i,j] <- (predictors[i] - x[k])^(j-1)
            }
        }

        # Weights, these change in every loop, therefore must be initialized
        # in every loop.
        # Each weight corresponds to one element of the x vector provided
        # by the user
        weights <- c()
        for (i in 1:length(predictors)) {
            # We use the K_h normal kernel of the difference between
            # the i-th predictor and the k-th element of x
            weights[i] <- pnorm((predictors[i] - x[k])/h)/h
        }
        # We create a matrix where the entries of the weights vector
        # are in the diagonal of a matrix (where all other values are 0)
        weights <- diag(weights)

        # We calculate W_i^P (x[i]), that is the matrix product of:
        # the transpose of e_1 times the inverse of the matrix product
        # of the transpose of the k-th X times the weights times the k-th X
        # times the transpose of the k-th X times the weights times the response
        # We then assign this to the k-th element of the result vector initialized before
        result[k] <- t(e_1) %*% solve(t(X[[k]]) %*% weights %*% X[[k]]) %*% t(X[[k]]) %*% weights %*% Y
    }

    ## RETURNING VALUES
    # We return the estimator evaluated at the vector x
    return(result)
}
```

b) Test the implementation by estimating the regression function in the location model $Y = m(X) + \epsilon$, where $m(x) = (x-1)^2$, $X \sim N(1,1)$, and $\epsilon \sim N(0, 0.5)$. Do it for a sample of size $n = 500$.

We assign m to the function $m(x) = (x-1)^2$

```
# function to estimate
m = function(x) ((x-1)^2)
```

We select a bandwidth h:

```
# bandwidth
h = 0.5
```

We generate *pred* (predictor variable values) and *resp* (response variable values) and we add the ϵ for each response entry in a loop after applying the $m(x)$ function defined earlier to the values of *pred*

```
# data simulation
pred <- rnorm(500, mean=1, sd=1)
resp <- c()
for (i in 1:length(pred)) {
    resp <- c(resp, m(pred[i])) + rnorm(1, mean=0, sd=0.5)
}
```

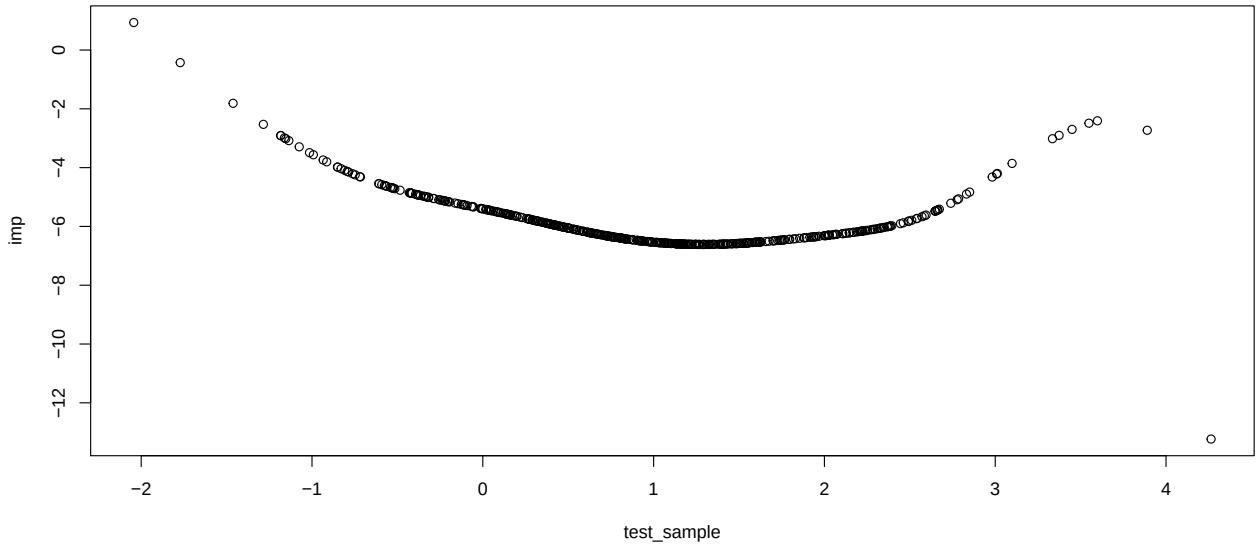
We create the variable *simulated_dataset* as a matrix with dimensions length(pred) $\times$ 2, the rows being amount of terms in *pred* and *resp* and columns being: 1- predictors and 2- response. This will facilitate the input to our *lce* function.

```
# appending to list object
simulated_dataset <- matrix(,nrow=length(pred),ncol=2)
simulated_dataset[,1] <- pred
simulated_dataset[,2] <- resp
```

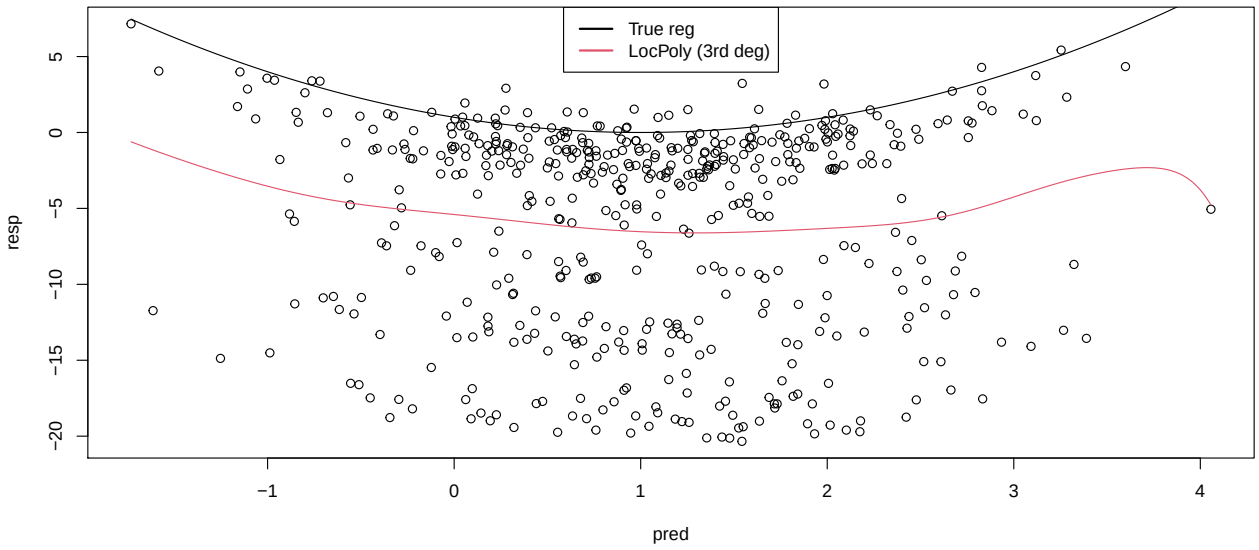
We run our custom implementation of the local cubic estimator for a sample size of $n = 500$, we generate the values from a normal distribution using *rnorm* with $\mu = 1$ and $\sigma = 1$.

```
# Running the custom implementation
test_sample <- rnorm(500, mean=1, sd=1)
imp <- lce(x=test_sample, data=simulated_dataset, h=h)
```

We plot the points obtained by our implementation vs the original values as follows



We run our implementation vs the true regression for a grid within the range of *pred*.



We can see that our implementation traces the true regression reasonably accurately with our chosen bandwidth $h = 0.5$

Category C: Problem 4

- **Exercise 3.30**. Load the *ovals.RData* file.

a) Split the dataset into the training sample, comprised of the first 2,000 observations, and the test sample (rest of the sample). Plot the dataset with colors for its classes. What can you say about the classification problem?

b) Using the training sample, compute the plug-in bandwidth matrices for all the classes.

c) Use these plug-in bandwidths to perform kernel discriminant analysis.

d) Plot the contours of the kernel density estimator of each class and the classes partitions. Use coherent colors between contours and points.

e) Predict the class for the test sample and compare with the true classes. Then report the successful classification rate.

f) Compare the successful classification rate with the one given by LDA. Is it better than kernel discriminant analysis?

g) Repeat f with QDA.