Challenge B

Sebastian Mantilla and Valentina Narvaez Trujillo

December 8, 2017

GitHub link

Here's the link to the repository: https://github.com/heymantilla/Challenge-B-S.-Mantilla-and-V.-Narvaez

Task 1B - Predicting house prices in Ames, Iowa (continued)

Step 1

The Machine Learning technique method that we chose is ***random forest***. It is an algorithm that generates robust predictions by creating a "forest" with a number of decision sheets; the more decision sheets (so the more trees) the more the prediction will be robust.

Step 2

To use random forest, we start by transforming character values into factors and then we handle the missing data by removing the variables that have more than a 100 missing values. Afterwards, we remove the observations with missing values.

Then, using the package *'randomforest'*, we run a regression creating a forest of 50 trees (ntree=50). mtry=5 means that we use 3 of the 5 variables at each split to estimate SalePrice.

```
##
## Call:
##  randomForest(formula = SalePrice ~ MSZoning + LotArea + Neighborhood +      YearBuilt + OverallQual
##                Type of random forest: regression
##                      Number of trees: 50
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 1350724787
##                     % Var explained: 78.29
```

Step 3.

Using the model predicted by random forest we obtained the prediction table that follows:

```
##     Id SalePrice_predict
## 1 1461          143378.7
## 2 1462          167503.1
## 3 1463          175818.3
## 4 1464          183416.5
## 5 1465          206285.1
## 6 1466          179900.8
```

Now, we compared the prediction given by random forest with the linear regression performed using the same random variables.

By eyeballing the first 6 raws of the predictions, we observe they are different.

```
##      Id SalePrice
## 1 1461  110073.2
## 2 1462  171382.6
## 3 1463  142314.0
## 4 1464  171336.5
## 5 1465  295944.8
## 6 1466  169593.8
```

Linear regressions are used when the data has a linear shape, however, when OLS is unable to capture non linear features. Random Forest can capture the non linearity in the data, therefore the predictions obtained are not the same as the predictions with the OLS model.

Task 2B - Overfitting in Machine Learning (continued)

Step 1

First, we estimate the model by doing a low flexibility local linear model only in the training data. We use the package *np*. We call this model *ll.fit.lowflex*

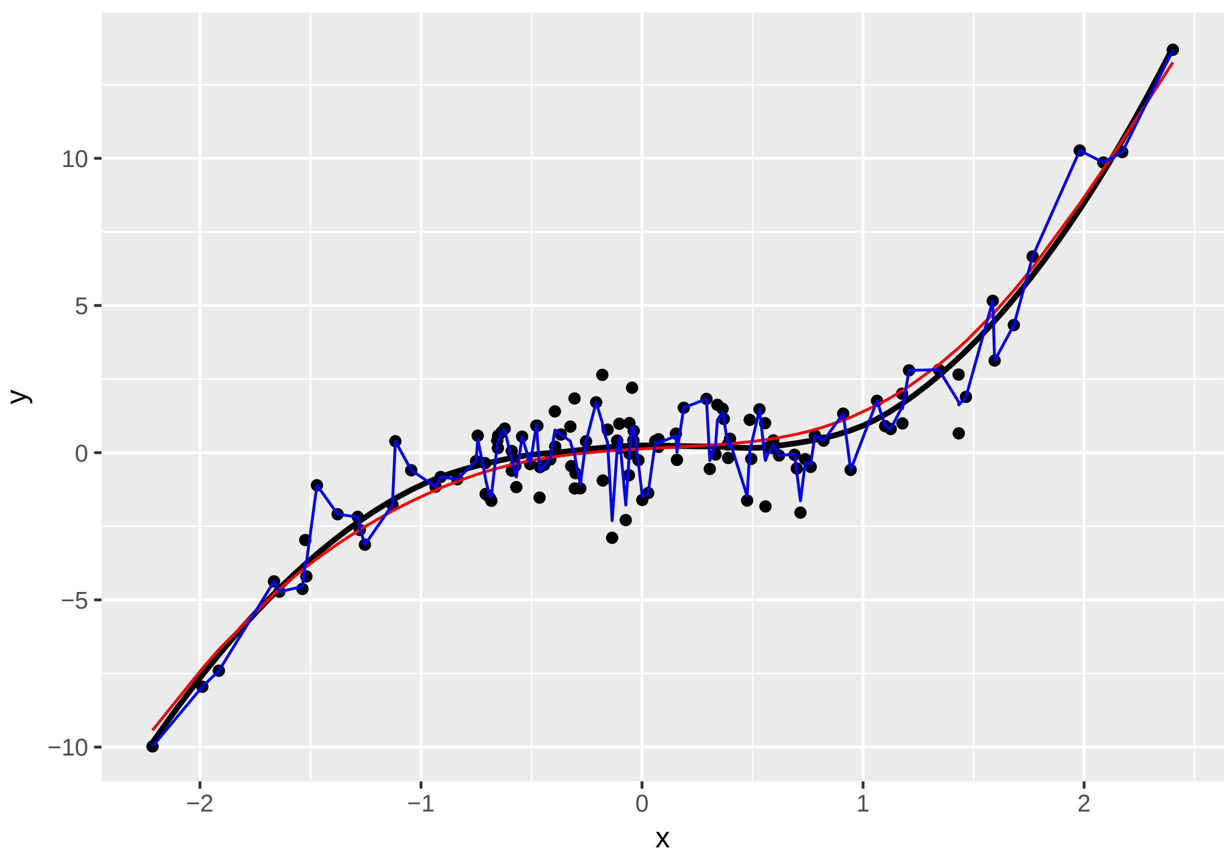
Step 2

Now, we estimate our model by doing a high flexibility local linear model only in the training data. We call this model *ll.fit.highflex*

Step 3

Here, we do a scatter plot of our x and y variables from our data (points in black), its true regression line (line in black), and the predictions we got in the previous 2 steps: the low flexibility prediction (in red) and the high flexibility prediction (in blue).

```
## `geom_smooth()` using method = 'loess'
```



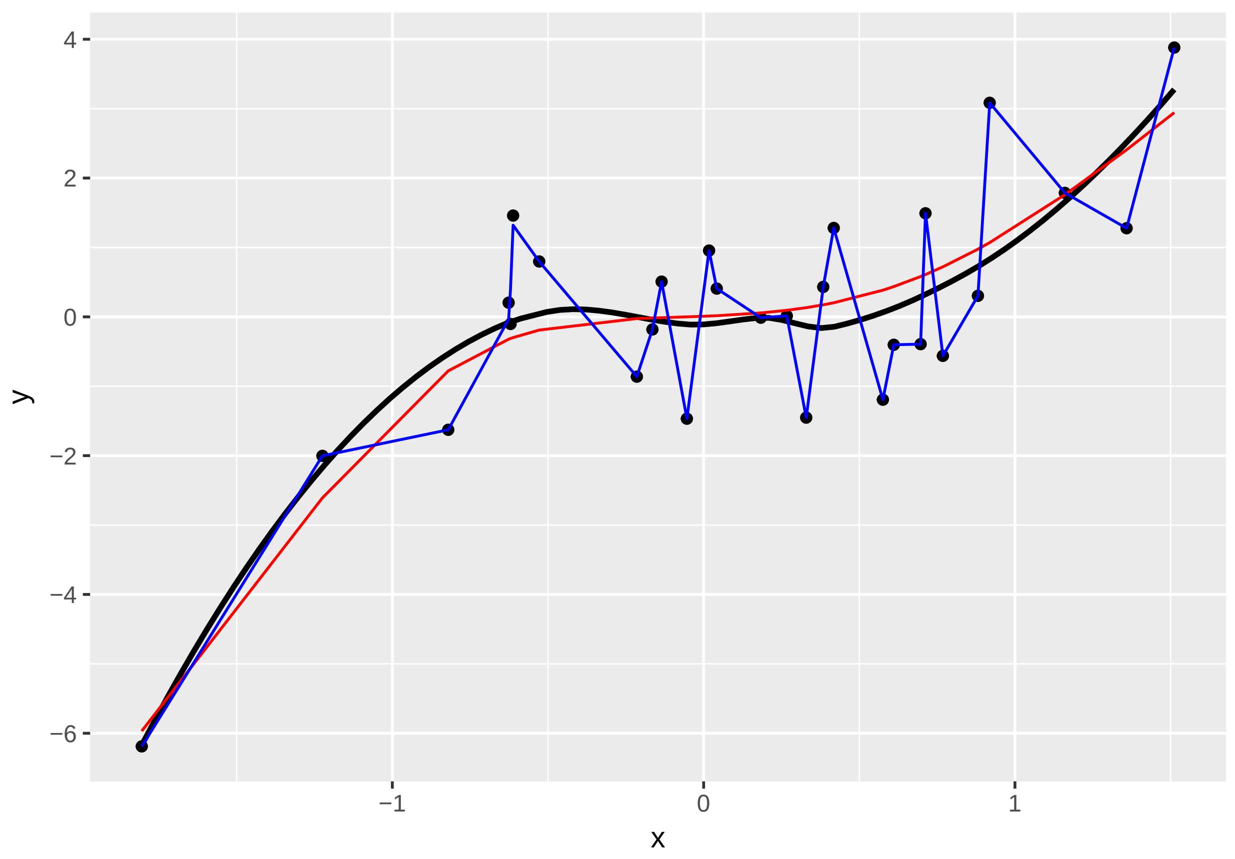
Step 4

Between the 2 models, the predictions from the high flexibility are more variable. The low flexibility prediction has the least bias.

Step 5

In this step, we have to do the same plot as in Step 3, but on the test data. For this, we have to estimate our model by doing a high and a low flexibility local linear regression on the test data only. Then, we can do our scatter plot:a scatter plot of our x and y variables from our data (points in black), its true regression line (line in black), the low flexibility prediction (in red) and the high flexibility prediction (in blue).

```
## `geom_smooth()` using method = 'loess'
```



Between the 2 models, the predictions from the high flexibility are more variable.

The low flexibility prediction has the least bias.

Step 6

Here, we simply create a vector of bandwidth going from 0.01 to 0.5, with a step of 0.001. We can see that our vector indeed starts with 0.01 and finishes with 0.5. Plus, we see there are 491 elements in our vector. We will need this number in future steps.

```
## [1] 0.010 0.011 0.012 0.013 0.014 0.015
```

```
## [1] 0.495 0.496 0.497 0.498 0.499 0.500
```

```
## [1] 491
```

Step 7

In this step, we need to perform a local linear regression on the training data using each of the bandwidth in the vector we created in the previous step. This means we have to perform a local regression with bandwidth equal to **0.01**, another one with **0.011**, and so on until **0.5**. You can see in the code that we performed a loop for this. The output, being extremely large, was not included in our report.

Step 8

In this step, we have to compute, for each bandwidth, the MSE on the **training data**.

After creating a vector called MSE1 in which each MSE value will be stored, we run the loop we did in *Step 7*. For each bandwidth, we extract the residuals of the model and them compute the MSE value, which will be stored in a vector we called MSE1.

As you can see below, our vector MSE1 has all the 491 MSE values, for each bandwidth.

```
## [1] 491
```

```
## [1] 0.2302086 0.2522588 0.2751926 0.2983326 0.3209526 0.3426808
```

Step 9

In this step, we have to compute, for each bandwidth, the MSE on the **test data**.

After creating a vector called MSE1 in which each MSE value will be stored, we run the loop we did in *Step 7*. For each bandwidth, we extract the residuals of the model and them compute the MSE value, which will be stored in a vector we called MSE2.

As you can see below, our vector MSE2 has all the 491 MSE values, for each bandwidth.

```
## [1] 491
```

```
## [1] 0.01182457 0.01254792 0.01312352 0.01358735 0.01397096 0.01432541
```

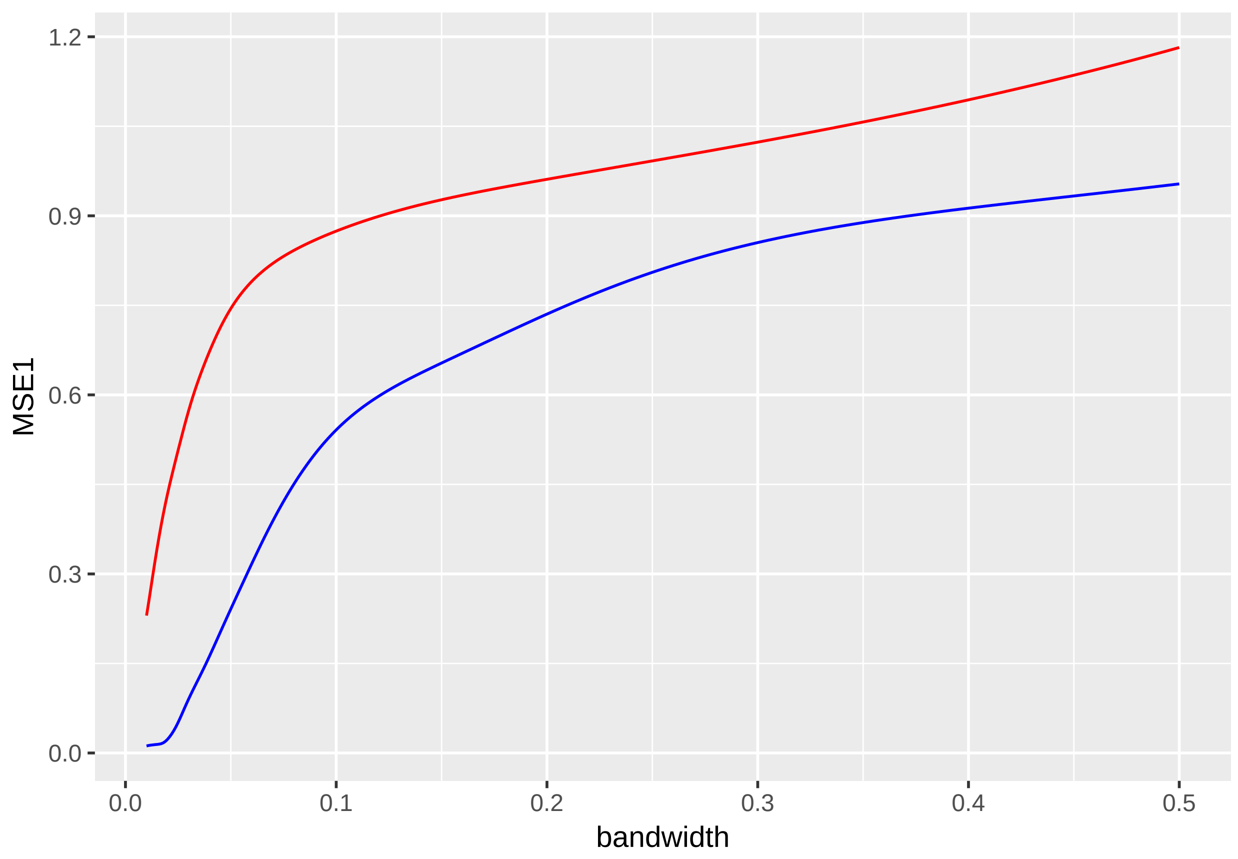
Step 10

In this final step, we draw, on the same plot, how the MSE on the training dta and the MSE on the test data change when the bandwidth increases.

We first create a data set in which we store every bandwidth value from *0.01* to *0.5*, and each bandwidth value is associated to its corresponding MSE value.

```
##   bandwidth      MSE1       MSE2
## 1     0.010 0.2302086 0.01182457
## 2     0.011 0.2522588 0.01254792
## 3     0.012 0.2751926 0.01312352
## 4     0.013 0.2983326 0.01358735
## 5     0.014 0.3209526 0.01397096
## 6     0.015 0.3426808 0.01432541
```

We draw the plot. The MSE values on the training data are in red and the ones on the test data are in blue (although, it should be noted that the graph isn't exactly the one we had to find).



Task 3