Challenge B - R Programming

Antonio Avila
Oriol Gonzalez

19/11/2017

TASK 1B: Predicting house prices in Ames, Iowa (continued)

Step 1: ML explanation

A feed-forward neural network is an artificial neural network wherein connections between the units do not form a cycle. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network, nor feedback connections in which outputs of the model are fed back into itself. The goal is to estimate the best function approximation f that maps inputs to outputs.

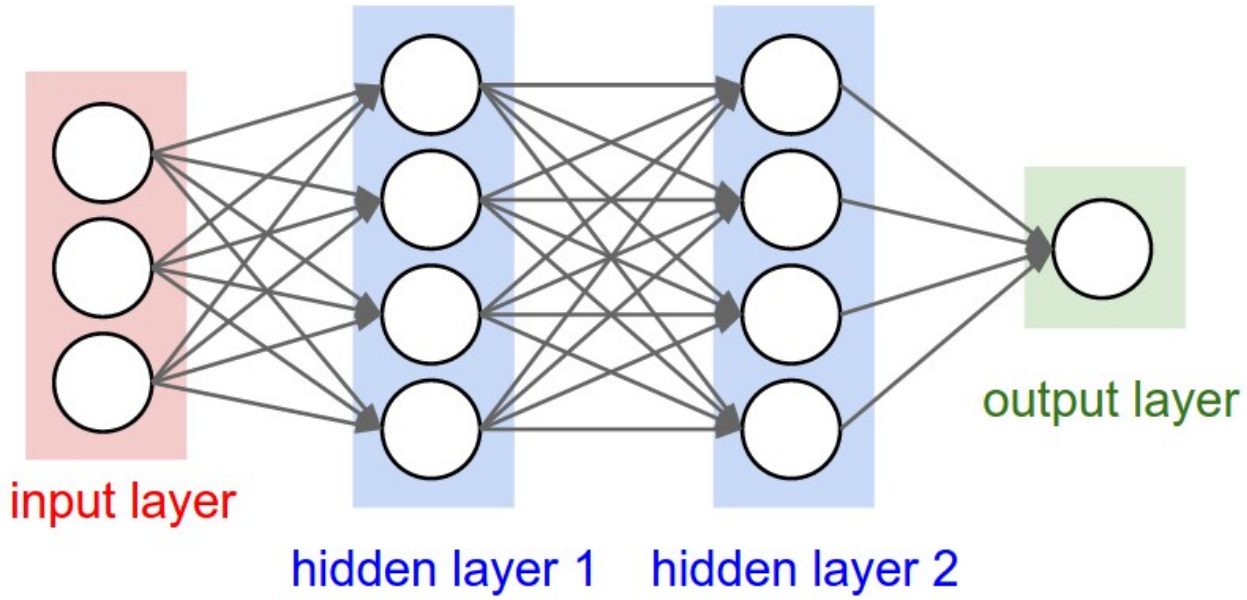


Figure 1: Feed-forward neural networks.

Step 2: Training the data

Need to install packages on the console

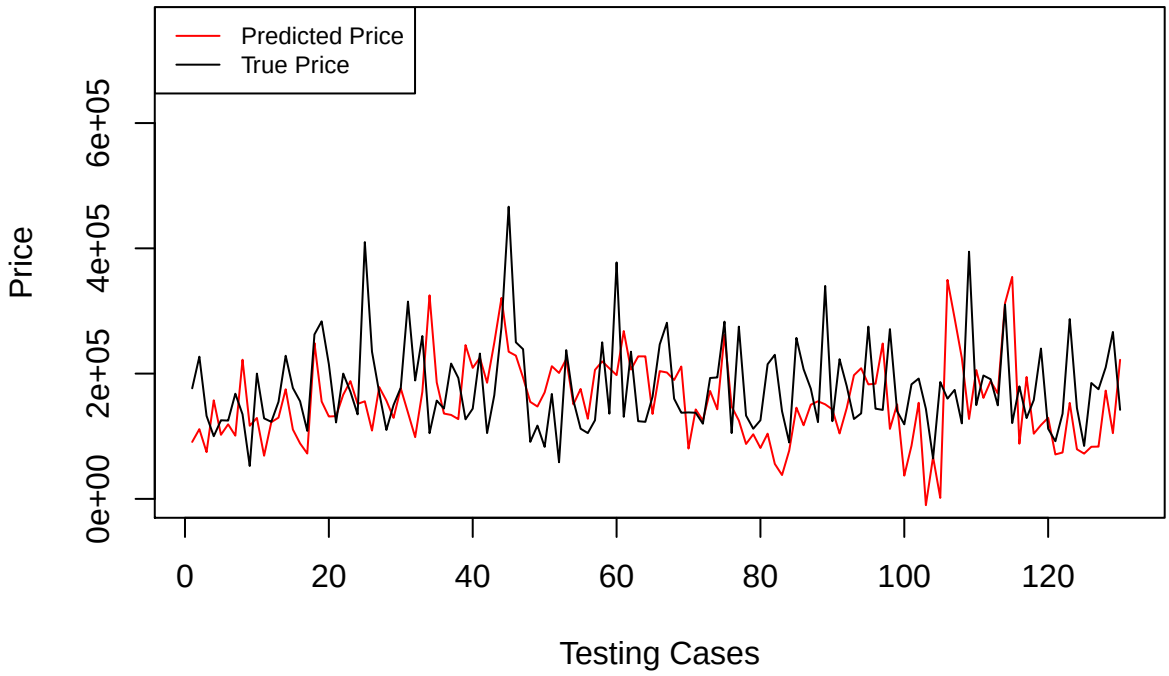
```
library(nnet) #load the non-parametric estimation package
training <- nnet(SalePrice~., traindf, size = 3, skip = TRUE, linout = TRUE)
```

```
## # weights:  931
## initial  value 58872611621566.625000
## iter  10 value 4087779545925.683105
## iter  20 value 3667172490783.154297
```

```
## iter  30 value 1626337334333.269775
## iter  40 value 1417854449415.617188
## iter  50 value 1134980955247.182617
## iter  60 value 928035038675.642822
## iter  70 value 813387865215.270752
## iter  80 value 751463926151.200195
## iter  90 value 722185739814.390625
## iter 100 value 687218170870.148560
## final  value 687218170870.148560
## stopped after 100 iterations
```

Step 3: Predictions

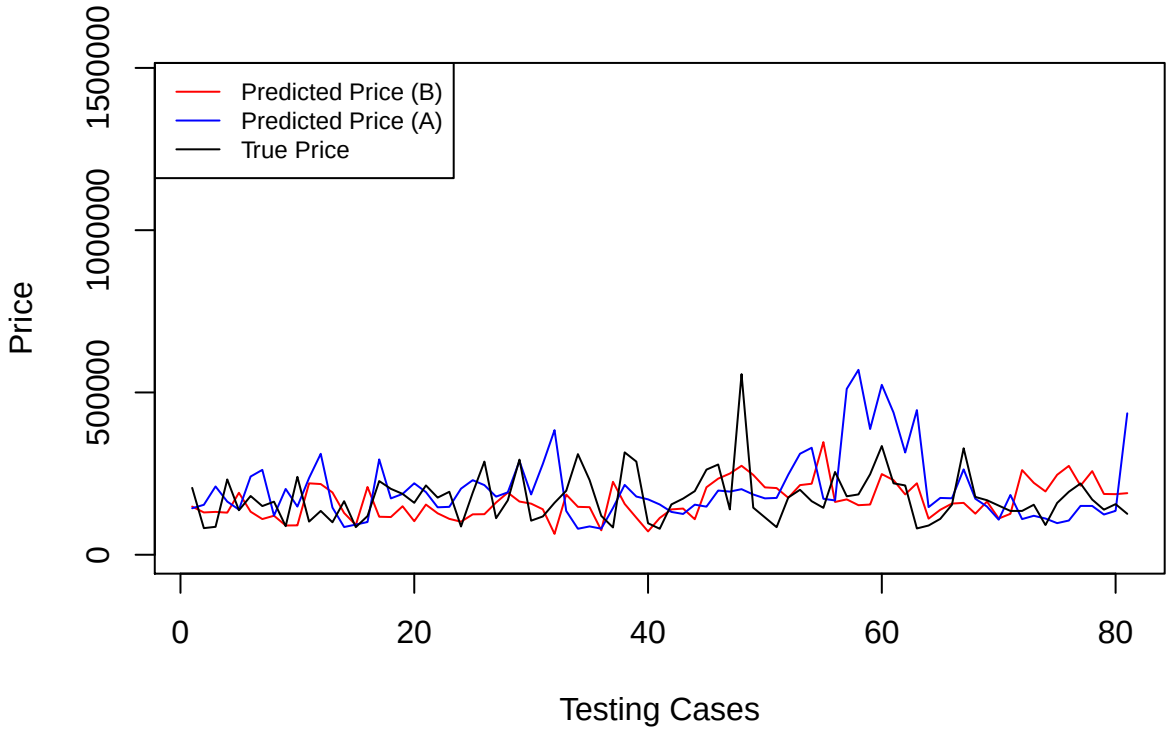
```
##      predicted actual
## 1461  108390.4 208500
## 1462  150542.0 181500
## 1463  181117.5 223500
## 1464  190230.7 140000
## 1465  204397.2 250000
## 1466  170304.2 143000
```




```
##      predicted.now predicted.before actual
## 1461      108390.4         126030.1 208500
## 1462      150542.0         158951.0 181500
## 1463      181117.5         186413.3 223500
## 1464      190230.7         198616.8 140000
## 1465      204397.2         198232.7 250000
## 1466      170304.2         169510.9 143000
```

```
l = 1000:1080
plot(NA, xlim = c(1, length(predictionsa[l, 1])), ylim = c(0, max(predictionsa[, 2])),
     xlab = "Testing Cases", ylab = "Price")
```

```
lines(predictionsa[1, 1], col = "red")
lines(predictionsa[1, 2], col = "blue")
lines(predictionsa[1,3])
legend("topleft", cex = 0.75, lty = 1, c("Predicted Price (B)",
  "Predicted Price (A)", "True Price"), col = c("red", "blue", "black"))
```



TASK 2B: Overfitting in Machine Learning (continued)

Step 1: Estimatng a low-flexibility local linear model on the training data

```
ll.fit.lowflex <- npreg(y~x, bws = 0.5, train, regtype="ll")
summary(ll.fit.lowflex)
```

```
##
## Regression Data: 122 training points, in 1 variable(s)
##                  x
## Bandwidth(s): 0.5
##
## Kernel Regression Estimator: Local-Linear
## Bandwidth Type: Fixed
## Residual standard error: 1.085438
## R-squared: 0.8540956
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

Step 2: Estimatng a high-flexibility local linear model on the training data

```
ll.fit.highflex <- npreg(y~x, bws = 0.01, train, regtype="ll")
summary(ll.fit.highflex)
```

```
##
## Regression Data: 122 training points, in 1 variable(s)
##                  x
## Bandwidth(s): 0.01
##
## Kernel Regression Estimator: Local-Linear
## Bandwidth Type: Fixed
## Residual standard error: 0.5070779
## R-squared: 0.9680171
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

Step 3: Scatterplot of x-y, along with the predictions of ll.fit.lowflex and ll.fit.highflex, on only the training data

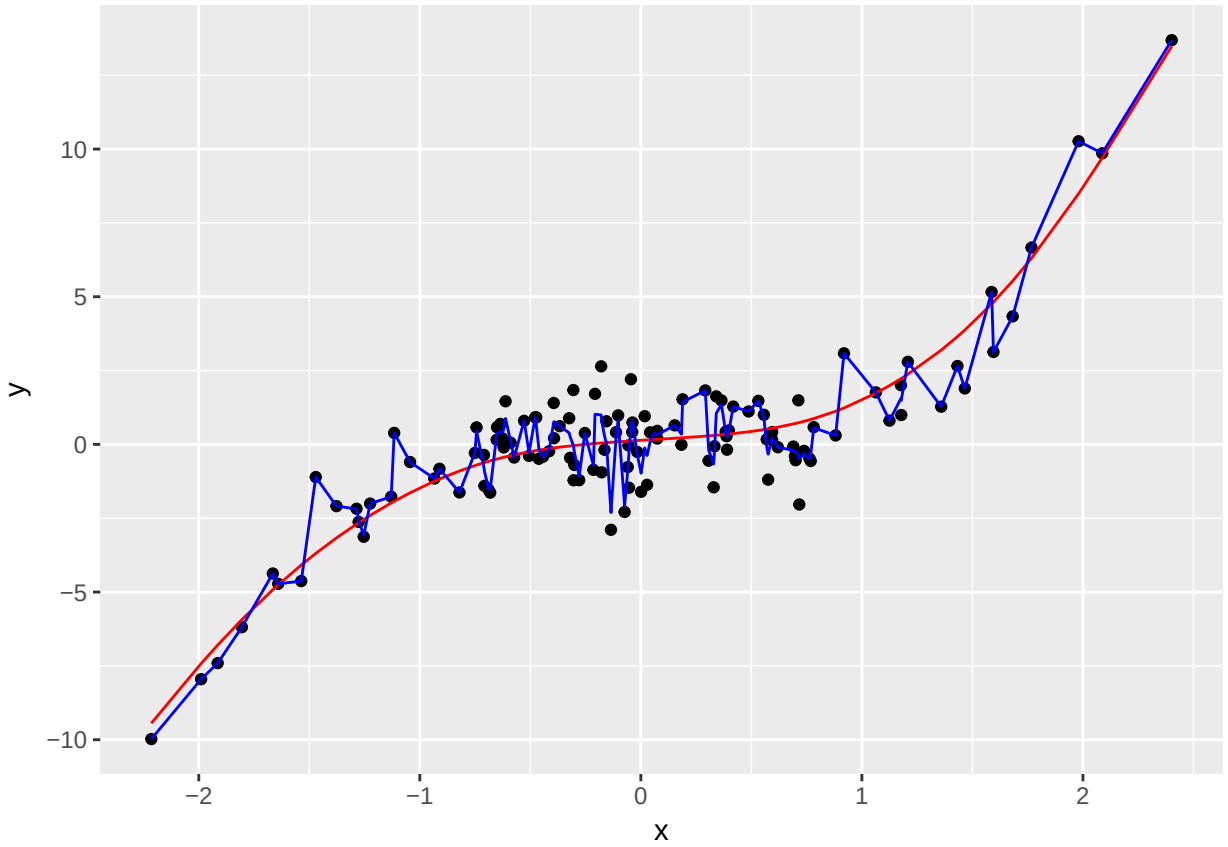


Figure 2: Step 3 - Predictions of ll.fit.lowflex and ll.fit.highflex on training data.

Step 4: Interpretation of the predictions

We can see that there is a trade-off in the predictions between little variance and little bias. On one hand, the predictions of ll.fit.lowflex are less variable, but have more bias, i.e. propably underfitted. On the other hand, predictions of ll.fit.highflex are more variable but have less bias, i.e. probably overfitted.

Step 5: Scatterplot of x-y, along with the predictions of ll.fit.lowflex and ll.fit.highflex now using the test data

```
predictlow <- predict(ll.fit.lowflex, newdata=test) #We save the predictions
predicthigh <- predict(ll.fit.highflex, newdata=test)
```

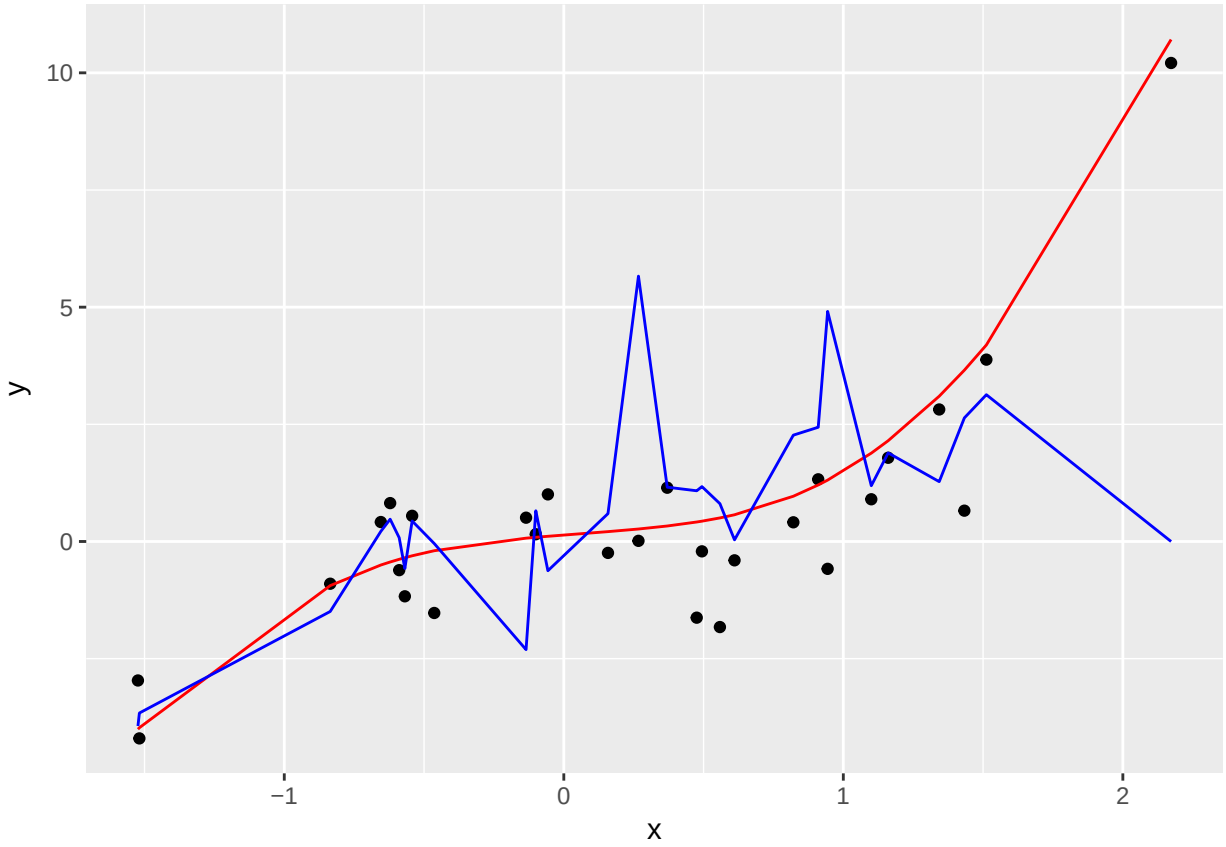


Figure 3: Step 5 - Predictions of ll.fit.lowflex and ll.fit.highflex on test data.

Again, the predictions for the ll.fit.highflex model are more variable. The bias of the least biased model has increased, since before it matched the exact point of the actual data, but now it is matching the predictions.

Step 6 - Create a vector of bandwidth going from 0.01 to 0.5 with a step of 0.001

```
bandwith <- seq(0.01,0.5,0.001)
```

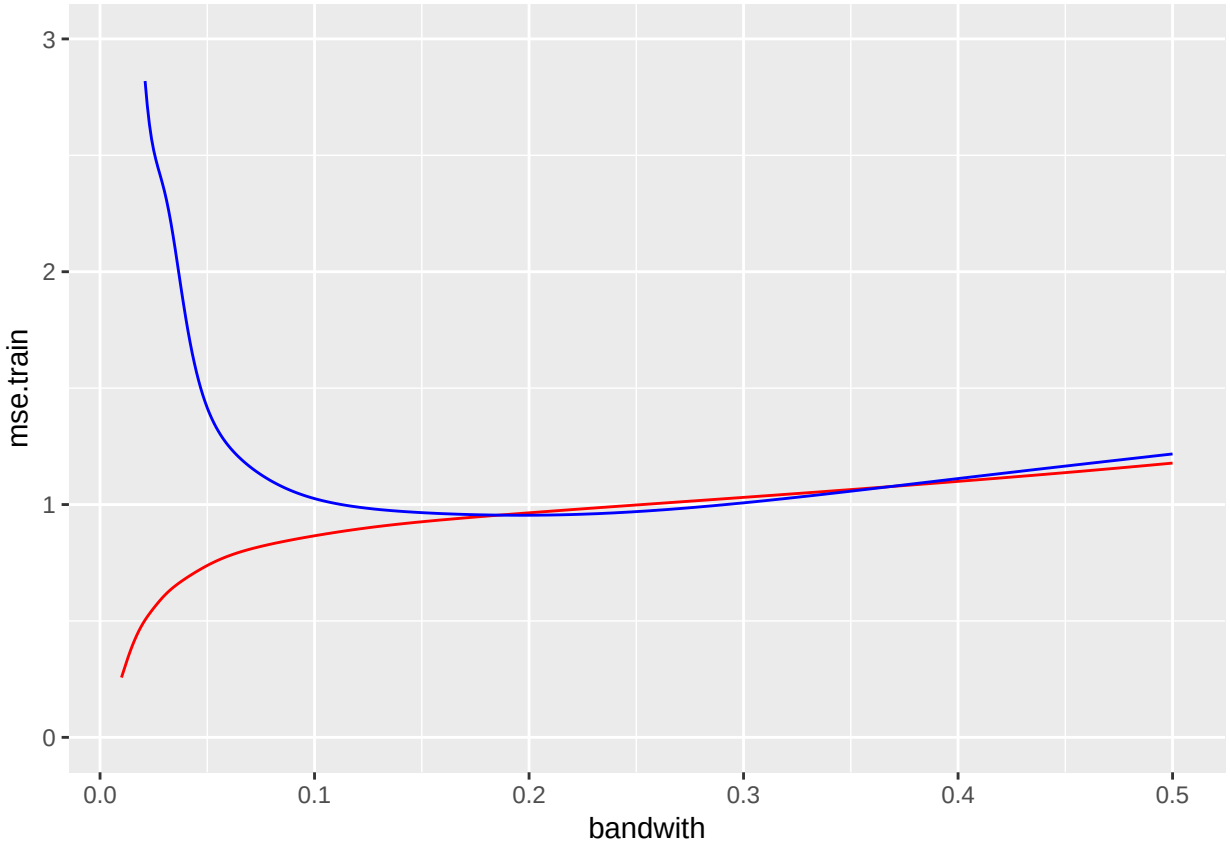
We create a sequence on numbers that will be the different bandwidths of our models

Step 7, 8 and 9

```
set.seed(1)
mse <- matrix(nrow = length(bandwith), ncol = 2)
for(i in 1:length(bandwith)){
  reg <- npreg(y~x, bws = bandwith[i], train, regtype="ll")
  predreg1 <- fitted.values(reg)
  predreg2 <- predict(reg, newdata=test)
  mse[i,1] <- mean( (predreg1-train$y)^2 )
  mse[i,2] <- mean( (predreg2-test$y)^2  )
}
mse <-data.frame(mse.train=mse[,1],mse.test=mse[,2])
```

We do steps 7,8 and 9 by regressing each model, doing the predictions and then saving the Mean Square Error in a dataframe.

Step 10 - Draw on the same plot how the MSE on training data, and test data, change when the bandwidth increases. Conclude.



The plot of MSE gives the usual result with
- a MSE of the sample monotonicaly increasing with commplexity (given the model can adjust less to the training data) and
- a MSE of the test with a convex form (given the bias-variance tradeoff) and a minimum when bandwith is close to 0.2.

TASK 3B: Privacy regulation compliance in France

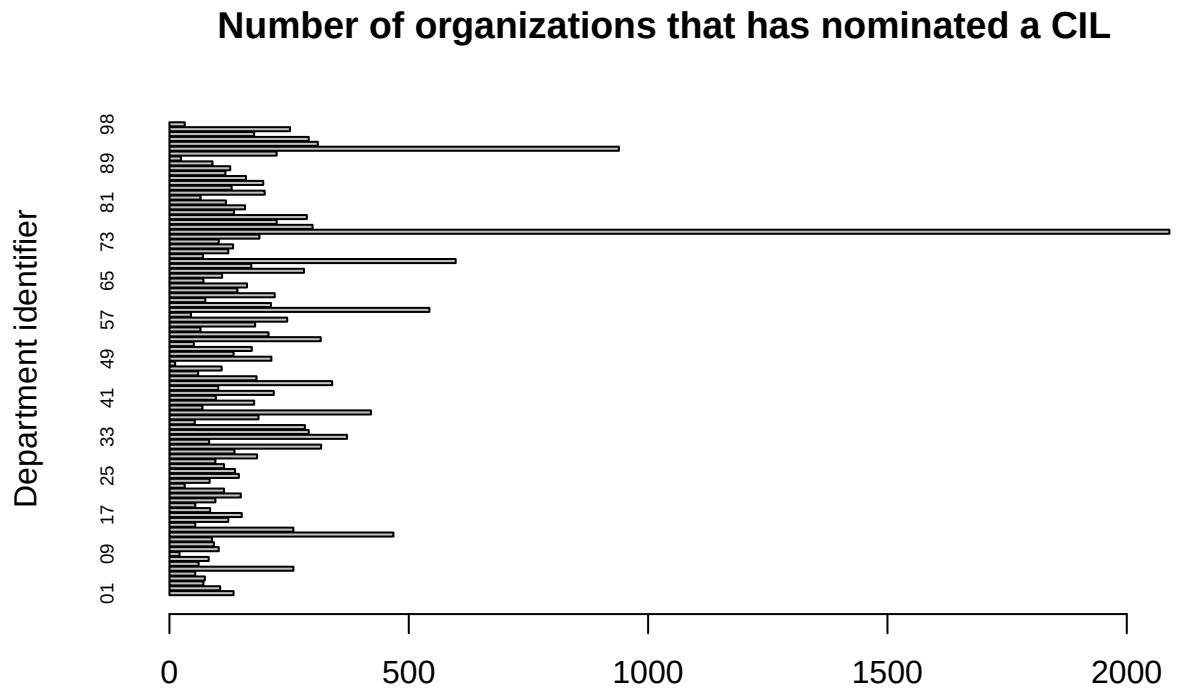
Step 1 - Import the CNIL dataset from the Open Data Portal

```
library(stringr)
CNIL <- read.csv("OpenCNIL_Organismes_avec_CIL_VD_20171115.csv", sep = ";")
```

Step 2 - Show a (nice) table with the number of organizations that has nominated a CNIL per department.

```
CP2 <- str_sub(CNIL$Code_Postal,start=1, end=2) #Get the first 2 digits
CNIL$CP2 <- CP2
```

```
CNIL$numeric <- CNIL$CP2 %in% c("01","02","03","04","05","06","07","08","09",0:100)
CNIL<-CNIL[CNIL$numeric==TRUE,]
count <- table(CNIL$CP2) # Count the number of rows that have the same CP2

#barplot(count, main="Number of in each department", horiz=T,names.arg= CNIL$Ville )
barplot(count, main="Number of organizations that has nominated a CIL", ylab = "Department identifier",
```



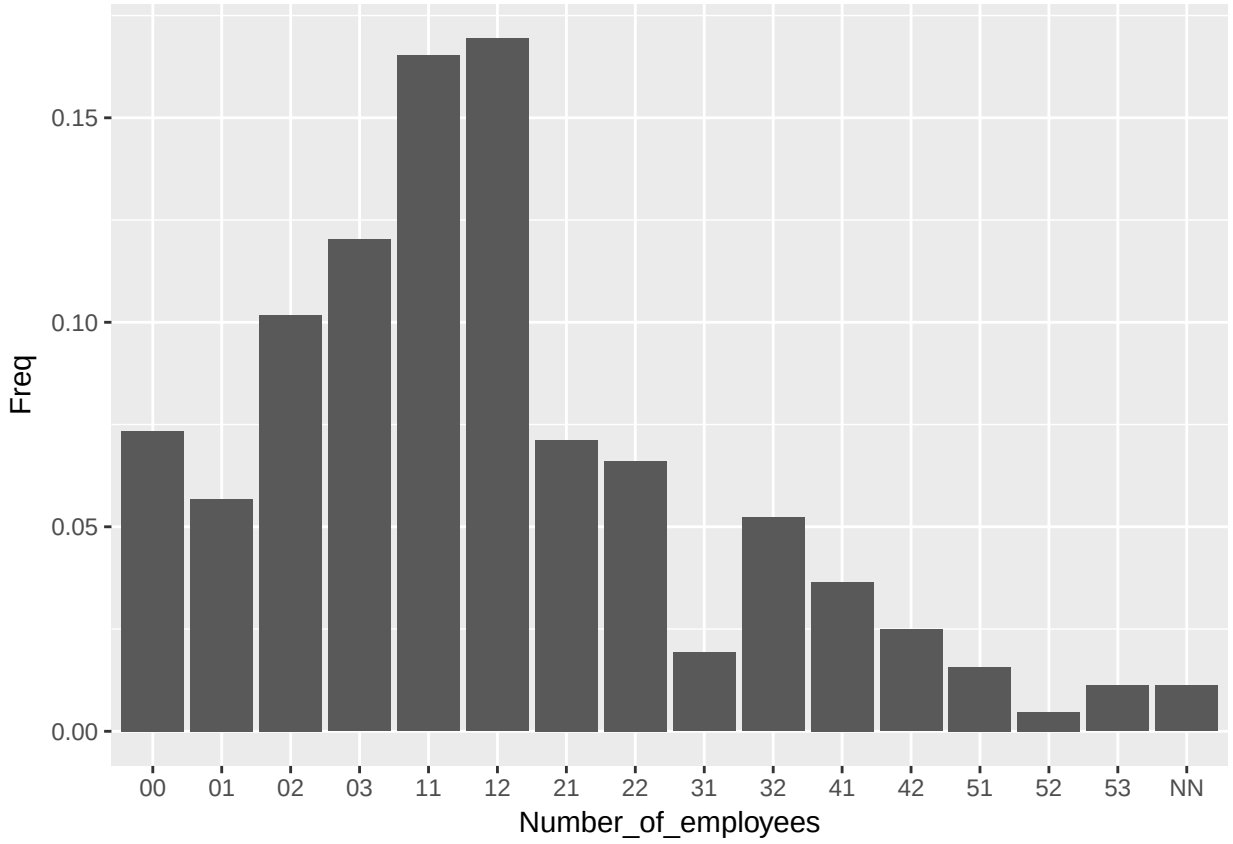
Step 3 - Merge the information from the SIREN dataset into the CNIL data. Explain the method you use.

```
#Charge the SIREN dataset:
require(data.table)

#SIREN <- fread("siren.csv")

#dup <- duplicated(SIREN[,1])

#SIREN <- SIREN[!dup, ]
#head(SIREN)

#merge <- merge(SIREN, CNIL, by='SIREN')

library(readr)
merge <- read_csv("merge.csv")
```

In order to merge the two datasets we need to:
- download and unzip SIREN (external)
- import it with fread
- identify the duplicates

- eliminate them and see that everithing is ok
- finally merge the two datasets with the common denominator.
(we include directly the results of this procedeure with the dataset “merge”)

Step 4 - Plot the histogram of the size of the companies that nominated a CIL. Comment.



Interpretation of the values of the x axis:

Value on Table	Interpretation
0	0 employee (but having employed during the reference year)
1	1 or 2 employees
2	3 to 5 employees
3	6 to 9 employees
11	10 to 19 employees
12	20 to 49 employees
21	50 to 99 employees
22	100 to 199 employees
31	200 to 249 employees
32	250 to 499 employees
41	500 to 999 employees
42	1 000 to 1 999 employees
51	2 000 to 4 999 employees
52	5 000 to 9 999 employees

Value on Table	Interpretation
53	10000 employees and more
NN	No employee during the reference year

We can see that most of the companies are neither very big nor small and have from 6 to 50 employees.