# Ridge regression on housing prices

## Data-set

This contribution develops an analysis on the cal-housing.csv documents using the program RStudio. The dataset contains information of houses identifying them through: 1. Longitude, 2. Latitude, 3. Housing Median Age, 4. Total Rooms, 5. Total Bedrooms, 6. Population, 7. Households, 8. Median Income, 9. Median House Value, 10. Ocean Proximity.

I run a Ridge regression in order to calculate the prediction of the Median House Value. Ridge regression helps us to analyse multiple regression data that are affected by multicollinearity: in case of least squares regressions, this would cause unbiased estimates, however the variances are very large and far away from the true value. Multicollinearity in fact refers to those non-linear relationships between independent variables. Ridge Regression is very useful in these cases. We can write it as: Y = XB + e X is a high-dimensional matrix, which in my case considers all the information about houses; B, which is the regression coefficient to be estimated; Y is the dependent variable, which is Median House Value ( this is the value I want to predict), whereas e represents the residual errors. Dataset cal-housing.csv Since the dataset we are considering contains NA (not available) values and non-numeric ones, I decided to omit the NA variables in the dataset:

calhousing <- data.frame(na.omit(calhousing))

and, since the only column with non-numeric values is the ocean\_proximity column, I turned them into numeric ones using the following R command:

calhousing$ocean\_proximity <- as.numeric(c("INLAND"=0,"NEAR BAY"= 1, "<1 OCEAN" = 0))

After these two passages, we will get our data from the following table with the changed elements (I report the first fifteen rows as example):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| longitude | latitude | housing\_median\_age | total\_rooms | total\_bedrooms | population | households | median\_income | median\_house\_value | ocean\_proximity |
| -122.23 | 37.88 | 41 | 880 | 129 | 322 | 126 | 8.3252 | 452600 | 0 |
| -122.22 | 37.86 | 21 | 7099 | 1106 | 2401 | 1138 | 8.3014 | 358500 | 1 |
| -122.25 | 37.85 | 52 | 1627 | 280 | 565 | 259 | 3.8462 | 342200 | 1 |
| -122.25 | 37.85 | 52 | 919 | 213 | 413 | 193 | 4.0368 | 269700 | 0 |
| -122.25 | 37.84 | 52 | 2535 | 489 | 1094 | 514 | 3.6591 | 299200 | 0 |
| -122.25 | 37.84 | 52 | 3104 | 687 | 1157 | 647 | 3.1200 | 241400 | 1 |
| -122.26 | 37.84 | 42 | 2555 | 665 | 1206 | 595 | 2.0804 | 226700 | 0 |
| -122.25 | 37.84 | 52 | 3549 | 707 | 1551 | 714 | 3.6912 | 261100 | 0 |
| -122.26 | 37.85 | 52 | 2202 | 434 | 910 | 402 | 3.2031 | 281500 | 1 |
| -122.26 | 37.85 | 52 | 3503 | 752 | 1504 | 734 | 3.2705 | 241800 | 0 |
| -122.26 | 37.85 | 52 | 2491 | 474 | 1098 | 468 | 3.0750 | 213500 | 0 |
| -122.26 | 37.84 | 52 | 696 | 191 | 345 | 174 | 2.6736 | 191300 | 1 |
| -122.26 | 37.85 | 52 | 2643 | 626 | 1212 | 620 | 1.9167 | 159200 | 0 |

# Data partitioning

After that, it’s possible to start our analysis. First of all it is important to data-partition my dataset, dividing it in training set and test set. I chose to use as a training set the 90% of the dataset and, consequently, the test set as the rest 10% of it. Then it is important to define the x and y of, respectively, the training set and the test set. Firstly I defined the training index 90% of the dataset:

trainingIndex <- sample(1:nrow(calhousing), 0.90\*nrow(calhousing)),

consequently, the training data are the following:

trainingData <- calhousing[trainingIndex, ] # training data

and the test data are:

testData <- calhousing[-trainingIndex, ] # test data.

Now it’s time to define the variables x and y of respectively the training set and the test set:

x\_train <- trainingData %>% select(longitude, latitude, housing\_median\_age,total\_rooms,households, median\_income, median\_house\_value, ocean\_proximity) %>% data.matrix()  
y\_train <- matrix(c(trainingData$median\_house\_value))  
x\_test <- testData %>% select(longitude, latitude, housing\_median\_age, total\_rooms,households, median\_income, median\_house\_value, ocean\_proximity) %>% data.matrix()  
y\_test <- matrix(c(testData$median\_house\_value))

# Linear regression model and Ridge regression model

My analysis started with a comparison between the linear regression and the ridge one. I used the algebra principles in order to define the predictions in both cases. For linear regression I built a hat matrix that helps me in identifying the predictions of y: H = X (X⊤X)−1 X⊤ , where X⊤ is the transpose matrix of X. In RStudio I run the following command, always considering the partition done at the beginning:

H1 <- x\_test%\*%(1/(t(x\_test)%\*%x\_test))%\*%t(x\_test)

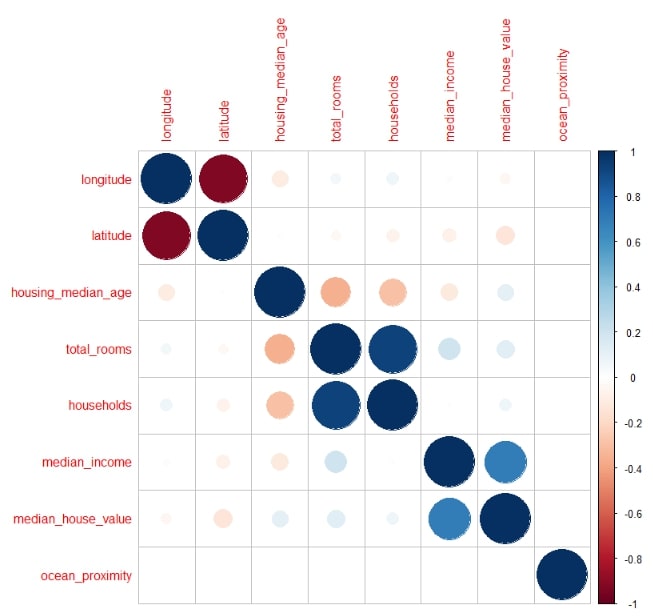
Consequently the predictions are given by: Y’ = X β’ = X (X⊤X)−1 X⊤Y = HY. At this moment it’s important to compare the real values and the predicted ones: in fact, it’s useful to create a “compare” table with two columns, the one on the left for the real values, the one on the right for the predicted ones: the RStudio command is the following

compare <- matrix(c(testData$median\_house\_value, predictions\_test), ncol = 2)  
```.  
I also run a correlation analysis to define which are the relations between variables, particularly focusing on   
those relations between all variables in x and the variable “median house value” in y:  
```R  
round(cor(x), 2)

whose output is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | longitude | latitude | housing\_median\_age | total\_rooms | households | median\_income | median\_house\_value | ocean\_proximity |
| longitude | 1.00 | -0.92 | -0.11 | 0.05 | 0.06 | -0.02 | -0.05 | 0 |
| latitude | -0.92 | 1.00 | 0.01 | -0.04 | -0.07 | -0.08 | -0.14 | 0 |
| housing\_median\_age | -0.11 | 0.01 | 1.00 | -0.36 | -0.30 | -0.12 | 0.11 | 0 |
| total\_rooms | 0.05 | -0.04 | -0.36 | 1.00 | 0.92 | 0.20 | 0.13 | 0 |
| households | 0.06 | -0.07 | -0.30 | 0.92 | 1.00 | 0.01 | 0.06 | 0 |
| median\_income | -0.02 | -0.08 | -0.12 | 0.20 | 0.01 | 1.00 | 0.69 | 0 |
| median\_house\_value | -0.05 | -0.14 | 0.11 | 0.13 | 0.06 | 0.69 | 1.00 | 0 |
| ocean\_proximity | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1 |

In a correlation test the absolute values are between 0 and 1. If the correlation is equal to 1 then the relation is linear, otherwise, in case the value is close to 0, it is a non-linear relationship. Furthermore, if both values tend to increase or decrease together the coefficient is positive, and the line that represents the correlation slopes upward, otherwise the coefficient is negative.



images

##Linear regression and ridge regression The main difference between the linear regression model and the ridge one is the presence in the equation of the latter of an additional element alpha, which is the regularization parameter that allows us to control the bias of the algorithm, and I defined it as: alphas <- matrix(seq(0.0, 1.0, by = .1), ncol = 8, nrow = 8). In fact, the equation of the ridge regression estimator is β = (X⊤X + alphas\*I) −1X⊤Y, Where “I” is the identity matrix and multiplied by the different values of alpha. It is useful to assign different values to alpha in order to find the optimal value that minimises the cross-validation risk estimate, identified as CV(n) =1/n ∑(y-y’)2.

Using RStudio it’s possible to build a ridge regression function, starting from the equation of ridge estimator:

ridge\_regr <- function () {  
 ((t(x)%\*%x+alphas%\*%I)^(-1))%\*%t(x)%\*%y  
 }

I build two other ridge regressions function respectively for the test data and the training data, obtained by substituting in the command above x\_test, y\_test and x\_train, y\_train:

ridge\_regr\_test(),  
  
ridge\_regr\_train().

After the identification of the ridge parameter is, now, possible to calculate the predictions on test set and training set as Y’=(X(t(X)X + alpha\*I)^(-1))t(X)Y, running the following functions in R:

pred\_test\_ridge <- x\_test%\*%(((t(x\_test)%\*%x\_test+alphas%\*%I1))^(-1))%\*%t(x\_test)%\*%y\_test  
  
pred\_train\_ridge <- x\_train%\*%(((t(x\_train)%\*%x\_train+alphas%\*%I2))^(-1))%\*%t(x\_train)%\*%y\_train

Now it is possible to evaluate the RMSE (Root Mean Squared Error) and the RSQ (R Squared), which help us to evaluate the variances between the real values and the estimated ones and the proportion of the variance in the dependent variable that is predictable from the independent ones. I used the following function:

evaluations <- function (true, predicted, df) {   
ssr<- sum((pred\_ridge - mean(y))^(2))   
sse<- sum((pred\_ridge - y)^2)  
sst <- ssr + ssr   
rsq <- 1 - sse / sst   
RMSE = sqrt(sse/nrow(df))  
data.frame(   
RMSE = RMSE,   
rsq = rsq  
 )  
}

The output of this function given by:

evaluations(y\_train, pred\_train\_ridge, trainingData)  
  
evaluations(y\_test, pred\_test\_ridge, testData)

give us the information we were looking for, that is the value of the R Squared, which, in both cases, is around 50%, whereas the RMSE is very high, but we want to enhance this value true the principal component analysis (PCA). Principal component analysis The principal component analysis is a process through which principal components are computed and these components are used to understand the data. In particular, PCA helps us to find a low-dimensional representation of the dataset that contains as much of the information as possible. I run the PCA through the following RStudio function:

cla.pca <- prcomp(x , center = TRUE,scale. = TRUE)

and the summary of it is:

Thanks to this it is now possible to consider just three of the principal components, always keeping in mind that our goal is to lower the cross-validation risk estimate. In fact, what can be done here is building a new matrix with the first three components and the response variable that we have considered in our study. I take the first three components:

pc1 <- cla.pca$rotation[,1]  
  
pc2 <- cla.pca$rotation[,2]  
  
pc3 <- cla.pca$rotation[,3]

and I build the final matrix throush the following commands:

housepc <- rbind(pc1,pc2,pc3)  
  
median\_house\_value <- rbind(pc1,pc2,pc3) [,7]  
  
house\_pc <- rbind(pc1,pc2,pc3,median\_house\_value)

The output is:

This is the x matrix of variables to consider in the analysis. I divide my new dataset in training set and test set, applying the ridge regression to it. I define the training index as 90% of my new data set:

trainIndex\_housepc <- sample(nrow(house\_pc), 0.90\*nrow(house\_pc))

Consequently, the training set and the test set will be:

train <- house\_pc[trainIndex\_housepc, ]  
  
test <- house\_pc[,trainIndex\_housepc ] %>% data.matrix()

As we did at the beginning, we have to define the variables x and y of both training set and test set:

x\_train\_housepc <- train %>% data.matrix()  
  
y\_train\_housepc <- matrix(c(median\_house\_value))  
  
x\_test\_housepc <- test %>% data.matrix()  
  
y\_test\_housepc <- matrix(c(median\_house\_value), nrow = 4, ncol = 1)

Then, I applied the ridge regression in both cases and I found the new values of the cross-validated risk estimate, for the training set the evaluations are:

evaluations <- function (true, predicted, df) {   
sse <- sum((pred\_housepc - y\_train\_housepc)^2)   
RMSE = sqrt(sse/nrow(df))   
data.frame(   
RMSE = RMSE  
rsq = rsq  
 )  
 }

Whose output is, running evaluations(y\_train\_housepc, pred\_housepc, house\_pc), is:

|  |  |
| --- | --- |
| RMSE | rsq |
| 4.004866 | 1.00000000 |

Whereas in the test set, the function is:

evaluations <- function (true, predicted, df) {  
 sse <- sum((pred\_housepc\_ - y\_test\_housepc)^2)  
 ssr <- sum((pred\_housepc\_ - mean(y\_test))^(2))  
 sst <- ssr + sse  
 rsq <- 1 - sse / sst  
 RMSE = sqrt(sse/nrow(df))  
   
 data.frame(  
 RMSE = RMSE,  
 rsq = rsq  
   
 )  
}

And the output is :

|  |  |
| --- | --- |
| RMSE | rsq |
| 133.4052 | 0.9999996 |

We can say that the RMSE is a lot better since it is lower than the initial result without considering the principal components.