Linear regression

Miguel-Angel Canela, IESE Business School

October 30, 2017

Contents

Introduction

In Data Science, one of the main jobs is **prediction**, that is, the description of one attribute (Y) in terms of the other attributes (X's). In this context, the term **regression** applies to the prediction of a numeric variable, and **classification** to the prediction of a categorical variable. In an example of regression, we may try to predict the price of a house from a set of attributes of this house. In one of classification, to predict whether a customer is going to quit or not, from his/her demographics plus some measures of customer activity.

Regression models are not necessarily related to a mathematical equation, as in Statistics, although this is the favourite approach for beginners. When this equation is linear, we have **linear regression**, which is the object of this lecture. Although the predictions of a linear regression model can usually be improved by more sophisticated techniques, most data scientists start there, because it helps them to understand the data.

Two alternatives to linear regression are:

- **Regression trees**. Although in this course tree algorithms are discusused in a classification context, some of them, like the CART algorithm used in the `R` package `rpart`, also apply to regression trees.
- **Neural networks**. A neural network is a programming device whose design is based on the models developed by biologists for the brain neurons. Among the many types of neural networks, the most popular is the **multilayer perceptron** (MLP), which can be regarded as a set of (nonlinear) regression equations. In `R`, the package `nnet` provides a simple approach to MLP regression models.

Evaluation of a linear regression model

In general, regression models are evaluated through the prediction errors. The basic schema is

$$\text{Prediction error} = \text{Actual value} - \text{Predicted value}.$$

Prediction errors are called **residuals** in linear regression. In that special case, the mean of the prediction errors is zero, but this is no longer true in other models. The usual approach for estimating the regression coefficients is the **least squares method**, which minimizes the sum of the squared residuals.

In Statistics, the predictive power of a linear regression model is evaluated through the **residual sum of squares**. The **R-squared statistic** is a standardized measure which operationalizes this. More specifically, we take advantage of the formula

$$\text{var}\big(\text{Actual values}\big) = \text{var}\big(\text{Predicted values}\big) + \text{var}\big(\text{Prediction error}\big)$$

to evaluate the model through the proportion of **variance explained**,

$$R^2 = \frac{\text{var}\big(\text{Predicted values}\big)}{\text{var}\big(\text{Actual values}\big)}\,.$$

It turns out that the square root of R-squared coincides with the **correlation** between actual values and predicted values. Although this stops being true for other regression methods, this correlation is still the simplest approach to the evaluation of a regression model.

Dummies and factors

Categorical variables are managed by `R` as factors. Factors enter a regression equation through 1/0 dummies. In `R`, we can manage this in two ways:

- We can create dummies explicitly and include them in the equation (as many dummies as the number of categories minus 1).
- We can include directly the factor as a single term in the formula. Then `R` creates the dummies by itself, without need of changing the data set. These dummies are not part of the data set, and cannot be used out of the model.

If we include a character variable in a regression equation, `R` coerces it to a factor while building the model. So, in practice, we perform a linear regression analysis without using factors. Many people (including myself) prefer to skip factors as much as possible, leaving the categorical variables as character vectors.

Example: House sales in King County

In this example, I develop a model for house sale prices in King County (Washington), which includes Seattle. King County is a county located in the U.S. state of Washington. It is the most populous county in Washington (population 1,931,249 in the 2010 census), and the 13th-most populous in the United States. The data include homes sold between May 2014 and May 2015.

The data set contains 15 house features plus the price and the id columns, along with 21,613 observations. The variables included in the data set are:

- `id`, an identifier.
- `date`, the date when the sale took place.
- `price`, the sale price.
- `bedrooms`, the number of bedrooms.
- `bathrooms`, the number of bathrooms
- `sqft_living`, the square footage of the home.
- `sqft_lot`, the square footage of the lot.
- `floors`, the total floors (levels) in house.
- `waterfront`, a dummy for having a view to the waterfront.
- `condition`, a 1-5 rating.
- `sqft_above`, the square footage of the basement.
- `yr_built`, the year when the house was built.
- `yr_renovated`, the year when the house was renovated.

- `zipcode`, the ZIP code of the house.
- `lat`, the latitude of the house.
- `long`, the longitude of the house.

I import the data from a csv file, using here the option `stringsAsFactors=F`. The column `date` is then imported as string date, not as factor, which is the default. This is irrelevant for the actual analysis presented here.

```
king <- read.csv(file="king.csv", stringsAsFactors=F)
```

Actually, `king` is a data frame, with 21,613 rows and 17 columns.

```
str(king)
```

```
## 'data.frame':    21613 obs. of  17 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ zipcode      : int  98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ bedrooms     : int  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : int  1180 2570 770 1960 1680 5420 1715 1060 1780 1890 ...
##  $ sqft_above   : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
##  $ sqft_basement: int  0 400 0 910 0 1530 0 0 730 0 ...
##  $ sqft_lot     : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  3 3 3 5 3 3 3 3 3 3 ...
##  $ yr_built     : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
##  $ yr_renovated : int  0 1991 0 0 0 0 0 0 0 0 ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
```

Note the format of `date`. "20140502T000000" must be read as "2014-05-02 00:00:00". This is called **datetime format**. The "T" indicates the beginning of the time part. Since "20140502T000000" contains one non-numeric character, it has to be read as string data.

A summary of the data helps. Of course, the statistics of `id` and `zipcode` do not make sense. For `date`, which is a factor, the summary gives counts for the first six levels. Also, note that, for a dummy like `waterfront`, the mean is just the proportion of 1's. So, 0.75% of the houses have a view to the waterfront. The rest is easy.

```
summary(king)
```

```
##        id                 date              zipcode           lat       
##  Min.   :1.000e+06   Length:21613       Min.   :98001   Min.   :47.16  
##  1st Qu.:2.123e+09   Class :character   1st Qu.:98033   1st Qu.:47.47  
##  Median :3.905e+09   Mode  :character   Median :98065   Median :47.57  
##  Mean   :4.580e+09                      Mean   :98078   Mean   :47.56  
##  3rd Qu.:7.309e+09                      3rd Qu.:98118   3rd Qu.:47.68  
##  Max.   :9.900e+09                      Max.   :98199   Max.   :47.78  
##       long           bedrooms        bathrooms      sqft_living    
##  Min.   :-122.5   Min.   : 0.000   Min.   :0.000   Min.   :  290  
##  1st Qu.:-122.3   1st Qu.: 3.000   1st Qu.:1.750   1st Qu.: 1427  
##  Median :-122.2   Median : 3.000   Median :2.250   Median : 1910  
##  Mean   :-122.2   Mean   : 3.371   Mean   :2.115   Mean   : 2080  
##  3rd Qu.:-122.1   3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.: 2550  
##  Max.   :-121.3   Max.   :33.000   Max.   :8.000   Max.   :13540
```

```
##    sqft_above   sqft_basement       sqft_lot           floors
##  Min.   : 290   Min.   :   0.0   Min.   :    520   Min.   :1.000
##  1st Qu.:1190   1st Qu.:   0.0   1st Qu.:   5040   1st Qu.:1.000
##  Median :1560   Median :   0.0   Median :   7618   Median :1.500
##  Mean   :1788   Mean   : 291.5   Mean   :  15107   Mean   :1.494
##  3rd Qu.:2210   3rd Qu.: 560.0   3rd Qu.:  10688   3rd Qu.:2.000
##  Max.   :9410   Max.   :4820.0   Max.   :1651359   Max.   :3.500
##    waterfront         condition        yr_built     yr_renovated
##  Min.   :0.000000   Min.   :1.000   Min.   :1900   Min.   :   0.0
##  1st Qu.:0.000000   1st Qu.:3.000   1st Qu.:1951   1st Qu.:   0.0
##  Median :0.000000   Median :3.000   Median :1975   Median :   0.0
##  Mean   :0.007542   Mean   :3.409   Mean   :1971   Mean   :  84.4
##  3rd Qu.:0.000000   3rd Qu.:4.000   3rd Qu.:1997   3rd Qu.:   0.0
##  Max.   :1.000000   Max.   :5.000   Max.   :2015   Max.   :2015.0
##      price
##  Min.   :  75000
##  1st Qu.: 321950
##  Median : 450000
##  Mean   : 540088
##  3rd Qu.: 645000
##  Max.   :7700000
```

Linear regression model

A linear regression model can be obtained with the function `lm`. The key arguments are `formula` and `data`, of obvious meaning. The syntax of the formula is `y ~ x1 + x2 + ...`

```
fm <- price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors +
  waterfront + condition + sqft_above + yr_built + yr_renovated + lat + long
mod <- lm(formula=fm, data=king)
```

`mod` is a list. The same will be true for other models which appear in this course, for which I will not discuss their content, because that would be too technical. If you are interested in the structure of an `R` object, you can explore it with the function `names`, or (at your own risk) with `str`, which provides a lot of information but can give you too much output for complex objects. For an `lm` model, `names` is enough for learning what the model contains.

```
names(mod)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

In most models, the function `summary` provides useful information. For a linear regression model, it includes the regression coefficients, with the corresponding **p-values**, and the R-squared statistic, whose square root, R = 0.804, is the correlation between actual and predicted prices.

The function `summary` also provides a summary of the residuals. The maximum and minimum have less interest, but the median (50% percentile), and the first (25% percentile) and third quartiles (75% percentile) can be useful. Note that most terms are significant, due to the sample size. This is typical of big data sets, and not much attention is paid to significance in this context.

```
summary(mod)
```

```
##
## Call:
## lm(formula = fm, data = king)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1561603  -110101   -12300    84087  3922773
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -5.038e+07  1.656e+06 -30.422  < 2e-16 ***
## bedrooms     -5.244e+04  2.020e+03 -25.964  < 2e-16 ***
## bathrooms     5.648e+04  3.518e+03  16.057  < 2e-16 ***
## sqft_living   2.373e+02  4.388e+00  54.082  < 2e-16 ***
## sqft_lot     -3.205e-02  3.764e-02  -0.851   0.3945
## floors        9.801e+03  3.848e+03   2.547   0.0109 *
## waterfront    7.585e+05  1.746e+04  43.453  < 2e-16 ***
## condition     3.332e+04  2.535e+03  13.145  < 2e-16 ***
## sqft_above    6.303e+01  4.551e+00  13.849  < 2e-16 ***
## yr_built     -1.788e+03  7.587e+01 -23.567  < 2e-16 ***
## yr_renovated  2.686e+01  3.959e+00   6.783 1.21e-11 ***
## lat           6.420e+05  1.117e+04  57.496  < 2e-16 ***
## long         -1.900e+05  1.247e+04 -15.239  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 218500 on 21600 degrees of freedom
## Multiple R-squared:  0.6461, Adjusted R-squared:  0.6459
## F-statistic:  3286 on 12 and 21600 DF,  p-value: < 2.2e-16
```

The predicted values of a linear regression model are actually contained in the model (the element `fitted.values`), but I use here the function `predict` in order to apply the same steps as with other models. `predict` can be applied to a new data set, as far as it contains all the variables included on the right side of the formula of the model.

```
pred_price <- predict(object=mod, newdata=king)
```

Residual analysis

We can also examine directly the prediction errors, that is, the regression residuals,

```
pred_error <- king$price - pred_price
```

The standard deviation of the residuals is 218,407.1. Note that, although the correlation achieved with this regression equation may look quite strong, the residual standard deviation is still the 59.5% of that of the price (367,127.2), as shown by the following calculations.

```
sd(pred_error)
```

```
## [1] 218407.1
```

```
sd(king$price)
```

```
## [1] 367127.2
```

```
sd(pred_error)/sd(king$price)
```

```
## [1] 0.5949084
```

Another way of assessing the performance of the model is through the percentage of cases in which the error

exceeds a given threshold. For instance, in this case, only 20.2% of the prediction errors exceed 200,000 dollars (in absolute value), while 31.6% exceed 150,000. This allows for an assessment in dollar terms which is always helpful and may bridge the gap between the data scientist and a non-trained audience. To get the first of these proportions, I write the expression that I want to evaluate, `abs(pred_error) > 200000`. This creates a logical vector (`TRUE/FALSE`). When I apply `mean`, `R` transforms this vector into a dummy, so the mean gives the proportion of `TRUE` values.

```
mean(abs(pred_error) > 200000)
```

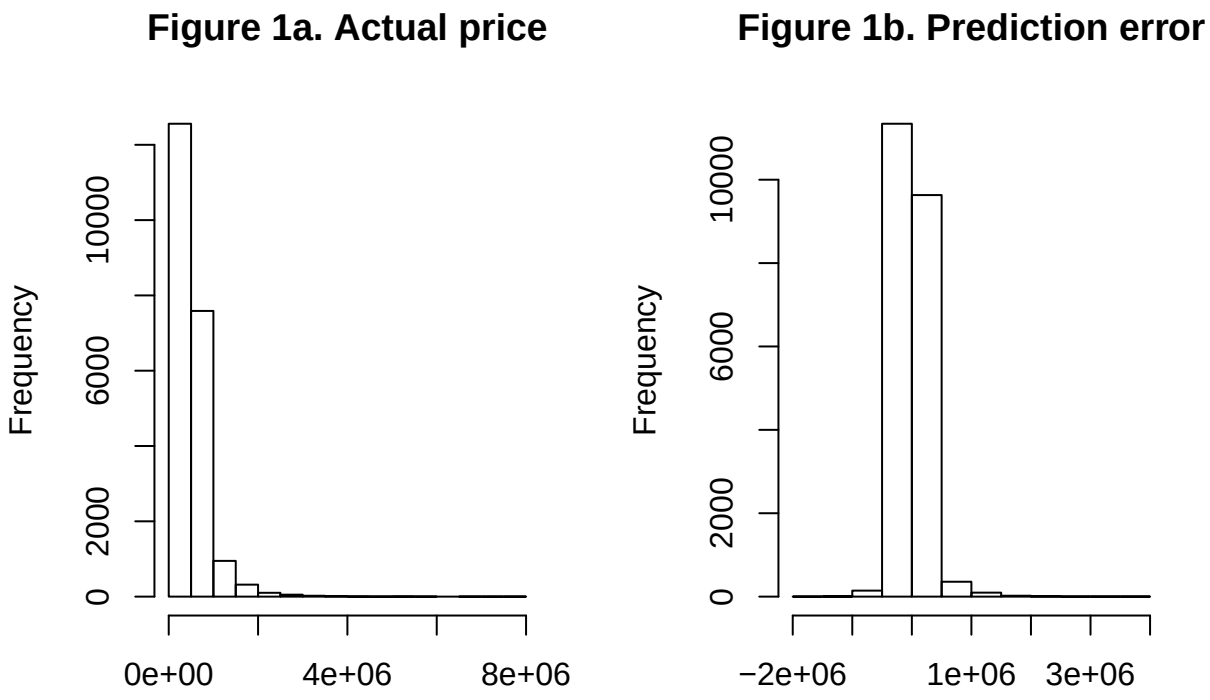
```
## [1] 0.2015916
```

```
mean(abs(pred_error) > 150000)
```

```
## [1] 0.3156434
```

Although the price has a (very) **skewed distribution**, that of the prediction error is reasonably symmetric, as shown in the exhibit below.

```
par(mfrow=c(1,2))
hist(king$price, main="Figure 1a. Actual price", xlab="")
hist(pred_error, main="Figure 1b. Prediction error", xlab="")
```

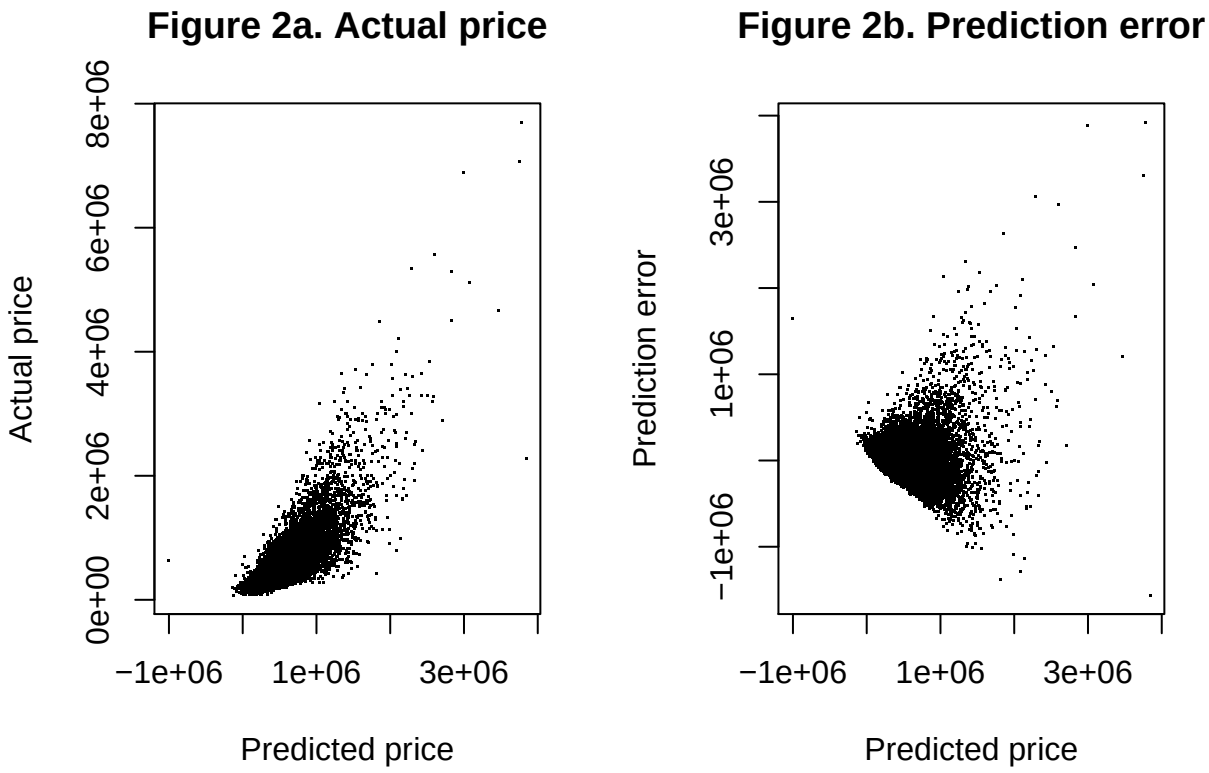


The command `par(mfrow=c(1,2))` splits the graphic device in two parts, so the next two plots will fill the resulting two slots. With the argument `mfrow` we control the partition. The `mfrow=c(1,2)` specification means "one row, two columns". You may resize the window in your screen to take advantage of this. Note that this split looks great for the `R` console, but can produce a poor result in RStudio, unless we resize the windows.

Plotting either actual values or prediction errors vs predicted values is useful to detect problems in the models

fitted. I leave the discussion of Figure 2 for the homework.

```
par(mfrow=c(1,2))
plot(king$price ~ pred_price, pch=".", main="Figure 2a. Actual price",
 xlab="Predicted price", ylab="Actual price")
plot(pred_error ~ pred_price, pch=".", main="Figure 2b. Prediction error",
  xlab="Predicted price", ylab="Prediction error")
```



Homework

Figure 1.a shows that prices have a very skewed distribution. There is nothing wrong in that, typically housing price data sets cover a wide range of sizes and prices (e.g. a house with 33 bedrooms in this dats set), and the big expensive houses are much less frequent than the small unexpensive ones. But, as Figure 2 shows, a linear regression equation fitted to such a data set comes with some problems attached. The most obviuous of these problems is that predicted prices can be negative, as we see in Figure 2.

In this homework, you can explore various approaches:

- Transformations, such the square root or the logarithm, are recommended in Statistics textbooks in many situations. In particular, the **log transformation**, is recommended for variables with skewed distributions, to limit the influence of extreme values. Develop a model for predicting the price which is based on a linear regression equation that has the logarithm of the price on the left side. Do you get better predictions with this model?
- It can be argued that having a model based on a linear equation does not make sense on such a wide range of prices. Indeed, in order to cope with the expensive houses, we are spoiling the prediction for

some of the nonexpensive houses. So, we could trim the data set, dropping the houses that exceed a certain theshold of price and/or size. What do you suggest?

- The ZIP code has not been used in the analysis. How would you introduce it? Would it be more useful than the longitude and the latitude?
- Time effects (trend/seasonality) have not been considered in the analysis. How would you explore that in the data?