Lab 1

September 22, 2021

1 Lab: Simple linear regression

In this lab, you will load data, plot data, perform simple mathematical manipulations, and fit a simple linear regression model. Before doing this lab, you can go through the demo to see an example of these operations on an automobile dataset. The lab use the Boston housing data set, a widely-used machine learning data set for illustrating basic concepts.

1.1 Loading the data

The Boston housing data set was collected in the 1970s to study the relationship between house price and various factors such as the house size, crime rate, socio-economic status, etc. Since the variables are easy to understand, the data set is ideal for learning basic concepts in machine learning. The raw data and a complete description of the dataset can be found on the UCI website:

https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.names

In the lab, you will complete all the code marked `TODO`.

First, complete the following code that uses the `pd.read_csv` command to read the data from the file located at

https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data

I have supplied a list `names` of the column headers. You will have to set the options in the `read_csv` command to correctly delimit the data in the file and name the columns correctly.

[1]:
```
import pandas as pd
import numpy as np
names =[
    'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
    'AGE',  'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'PRICE'
]

# TODO:  Complete the code
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪housing/housing.
 ↪data',header=None,delim_whitespace=True,names=names,na_values='?')
```

Display the first six rows of the data frame

[2]:
```
# TODO
df.head(6)
```

[2]:
```
       CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0
5  0.02985   0.0   2.18     0  0.458  6.430  58.7  6.0622    3  222.0

   PTRATIO       B  LSTAT  PRICE
0     15.3  396.90   4.98   24.0
1     17.8  396.90   9.14   21.6
2     17.8  392.83   4.03   34.7
3     18.7  394.63   2.94   33.4
4     18.7  396.90   5.33   36.2
5     18.7  394.12   5.21   28.7
```

1.2 Basic Manipulations on the Data

What is the shape of the data? How many attributes are there? How many samples? Print a statement of the form:

`num samples=xxx, num attributes=yy`

[3]:
```
# TODO
shape = df.shape
print("num samples =", shape[0] ,", num attributes =", shape[1])
```

```
num samples = 506 , num attributes = 14
```

Create a response vector `y` with the values in the column `PRICE`. The vector `y` should be a 1D `numpy.array` structure.

[4]:
```
# TODO
y = np.array(df["PRICE"])
```

Use the response vector `y` to find the mean house price in thousands and the fraction of homes that are above $40k. (You may realize this is very cheap. Prices have gone up a lot since the 1970s!). Create print statements of the form:

```
The mean house price is xx.yy thousands of dollars.
Only x.y percent are above $40k.
```

[5]:
```
# TODO
above_40_subset = df.apply(lambda x: True if x["PRICE"]>40 else False,  axis=1)
above_40 = len(above_40_subset[above_40_subset==True].index)
```

```
print("The mean house price is {0:1.2f} thousands of dollars".format(np.
↪mean(y)))
print("Only {0:1.1f} percent are above $40k".format(above_40*100/len(y)))
```

```
The mean house price is 22.53 thousands of dollars
Only 6.1 percent are above $40k
```

1.3 Visualizing the Data

Python's `matplotlib` has very good routines for plotting and visualizing data that closely follows the format of MATLAB programs. You can load the `matplotlib` package with the following commands.

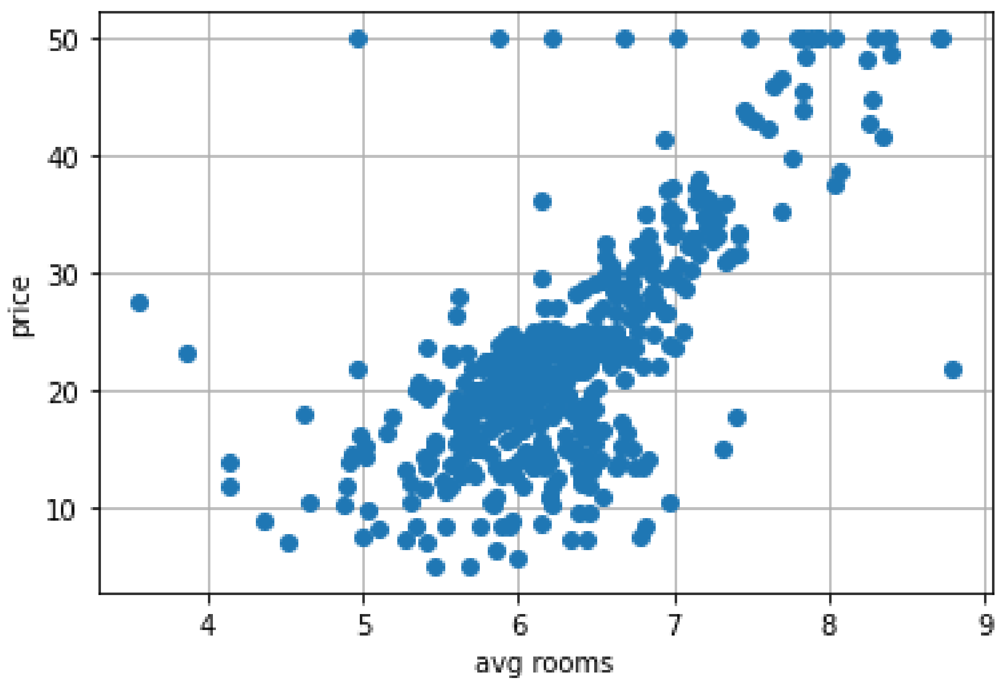
[6]:
```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

Similar to the `y` vector, create a predictor vector `x` containing the values in the `RM` column, which represents the average number of rooms in each region.

[7]:
```
# TODO
x =  np.array(df["RM"])
```

Create a scatter plot of the price vs. the `RM` attribute. Make sure your plot has grid lines and label the axes with reasonable labels so that someone else can understand the plot.

[8]:
```
# TODO
plt.plot(x,y,'o')
plt.xlabel("avg rooms")
plt.ylabel('price')
plt.grid(True)
```



1.4 Fitting a Simple Linear Model

We will write a simple function to perform a linear fit. Use the formulae given in the class, to compute the parameters β_0, β_1 in the linear model

$$y = \beta_0 + \beta_1 x + \epsilon$$

as well as the coefficient of determination R^2.

[12]:
```
def fit_linear(x,y):
    """
    Given vectors of data points (x,y), performs a fit for the linear model:
       yhat = beta0 + beta1*x,
    The function returns beta0, beta1 and rsq, where rsq is the coefficient of␣
 ↪determination.
    """
    # TODO complete the following code
    x_mean = np.mean(x)
    y_mean = np.mean(y)


    y_variance = np.mean((y-y_mean)**2)
    x_variance = np.mean((x-x_mean)**2)

    cov = np.mean((y-y_mean)*(x-x_mean))
    beta1 = cov/x_variance
```

```
    beta0 = y_mean - beta1*x_mean
    rsq = (cov**2)/((x_variance)*(y_variance))
    return beta0, beta1, rsq
```

Using the function `fit_linear` above, print the values `beta0`, `beta1` and `rsq` for the linear model of price vs. number of rooms.

[13]:
```
# TODO
beta0, beta1, rsq = fit_linear(x,y)
print("For the linear model of price vs. number of rooms\n")

print("beta0 = {0:7.2f}".format(beta0))
print("beta1 = {0:7.2f}".format(beta1))
print("rsq   = {0:7.2f}".format(rsq))
```

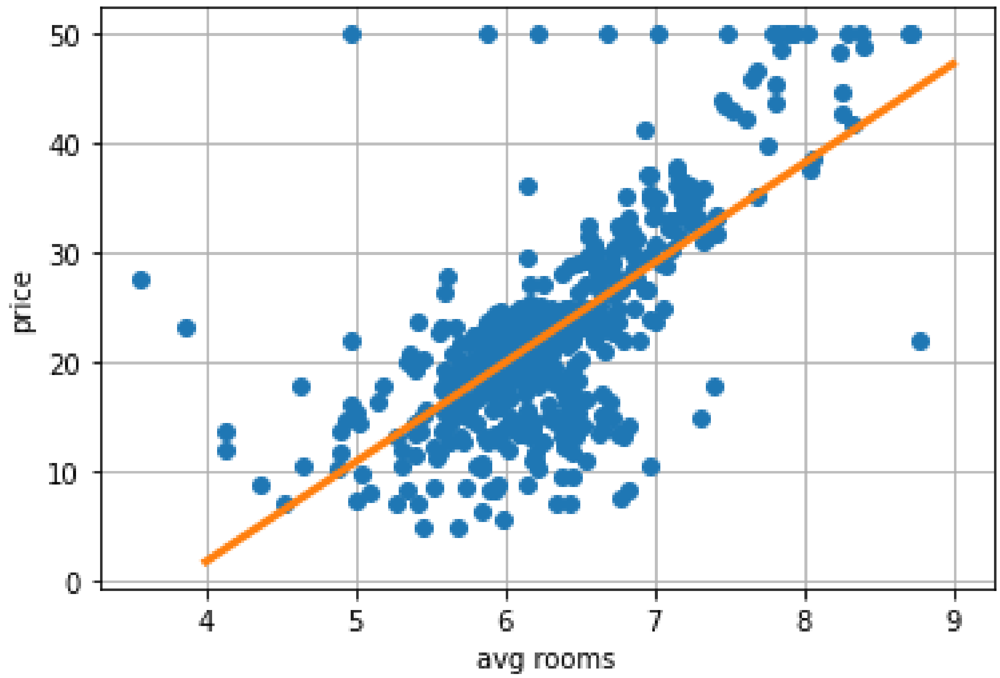
```
For the linear model of price vs. number of rooms

beta0 =  -34.67
beta1 =    9.10
rsq   =    0.48
```

Replot the scatter plot above, but now with the regression line. You can create the regression line by creating points `xp` from say 4 to 9, computing the linear predicted values `yp` on those points and plotting `yp` vs. `xp` on top of the above plot.

[15]:
```
# TODO
xp = np.array([4,9])
yp = beta1*xp + beta0

plt.plot(x,y,'o')                    # Plot the data points
plt.plot(xp,yp,'-',linewidth=3)  # Plot the regression line
plt.xlabel("avg rooms")
plt.ylabel('price')
plt.grid(True)
```



2 Compute coefficients of determination

We next compute the R^2 values for all the predictors and output the values in a table. Your table should look like the following, where each the first column is the attribute name and the second column is the R^2 value.

```
CRIM        0.151
ZN          0.130
INDUS       0.234
...         ...
```

To index over the set of columns in the dataframe `df`, you can either loop over the items in the `names` lists (skipping over the final name `PRICE`) or loop over integer indices and use the method, `df.iloc`.

[16]:
```
# TODO
attributes = dict()
df.drop("PRICE",inplace =True, axis =1)
for j in df:
    x_predictor = np.array(df[j])
    reg = fit_linear(x_predictor, y)
    # Storing the Rsq in the attribute dictionary
    attributes[j] = reg[2]

print("{:<3} {:<20}".format("Attribute    ", "R square value"))
```

```
print()

for i in attributes:
    print("{:<15} {:<20.3f}".format(i, attributes[i]))
```

```
Attribute      R square value

CRIM             0.151
ZN               0.130
INDUS            0.234
CHAS             0.031
NOX              0.183
RM               0.484
AGE              0.142
DIS              0.062
RAD              0.146
TAX              0.220
PTRATIO          0.258
B                0.111
LSTAT            0.544
```