Lecture 2: Regression Analysis

DATA 602 @ UMBC

Masoud Soroush

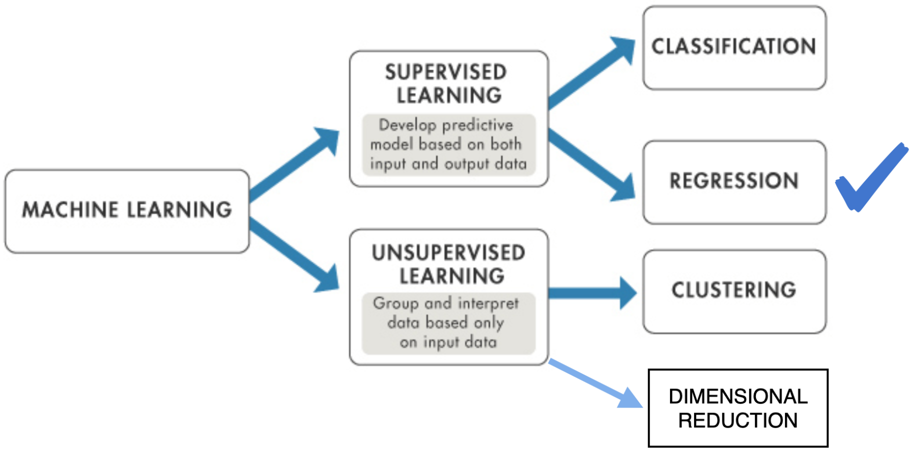
September 5, 2022

1 Regression Analysis

In today's lecture[1], we introduce one of the simplest and most common machine learning algorithms, namely the regression. Despite its simplicity, it is still widely used in statistical learning.

1.1 Background

In the context of *supervised learning*, regression is used to predict a quantitative continuous target variable. Moreover, in this lecture we will assume that all features (*i.e.* explanatory variables) are all continuous too.



Note: Be aware that regression methods have been generalized to cases where features include categorical variables as well. However, analyzing those cases require a deeper knowledge of statistics, and are beyond the scope of our current course.

[1] This product is a property of UMBC, and no distribution is allowed. These notes are solely for your own use, as a registered student in this class.

1.2 Formulating the Regression Problem

Let us first start with a simple case. Assume that we have a continuous target variable y explained by one single continuous feature x. We have recorded the values of x and y for n observations in a dataset. The dataset appears to have the following format:

Observation	x	y
1	x_1	y_1
2	x_2	y_2
3	x_3	y_3
$\vdots$	$\vdots$	$\vdots$
n	x_n	y_n

We assume a linear relation between the feature and the target variables

$$\hat{y} = \omega_1 x + \omega_0 \,, \tag{1}$$

where $\hat{y}$ denotes the predicted value for the target (Note that $\hat{y}$ is the predicted value of the target whereas y is the true value of the target). In equation (1), ω_0 and ω_1 are the intercept and the slope of the line, respectively. Note that in the literature, the constant term ω_0 is sometimes called the *bias term*. The bias term ω_0 *has nothing to do with the concept of bias* (appearing in Bias-Variance trade off for instance) in machine learning.

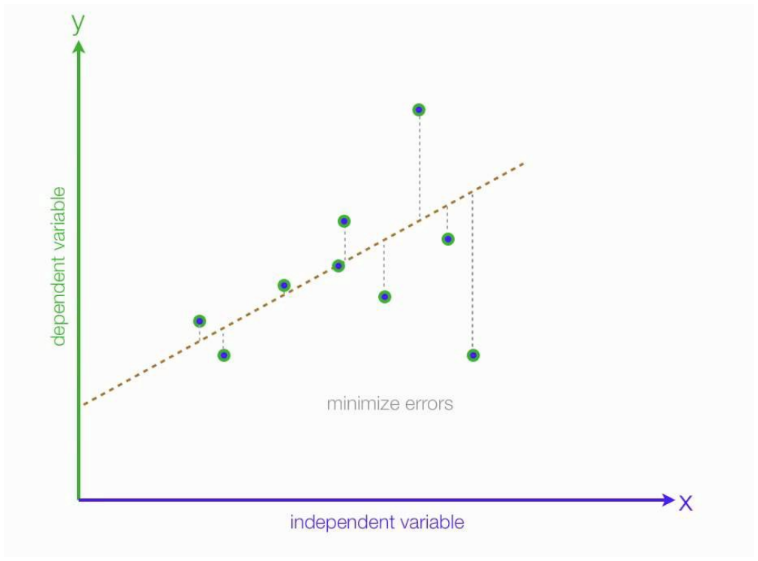
Goal: To determine the *best fit line*, we have to determine ω_1 (*i.e.* the slope of the line) and ω_0 (*i.e.* the intercept of the line) in equation (1). But before that, we need a criterion by which we can compare different lines.

To achieve the above goal, we define an error term $e_i = (y_i - \hat{y}_i)$ for each observation. We then define the *cost function* $J(\omega_0, \omega_1)$ associated with the regression algorithm to be

$$J(\omega_0, \omega_1) = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - \omega_1 x_i - \omega_0)^2 \,. \tag{2}$$

The best fit line is the one whose slope and intercept (*i.e.* ω_1, ω_0) minimizes the cost function in equation (2). This implies that, to minimize the cost function, one has to solve

$$\frac{\partial J}{\partial \omega_0} = 0 \qquad , \qquad \frac{\partial J}{\partial \omega_1} = 0 \,. \tag{3}$$



Before we solve equation (3), we have to think about the generalization of the above setup. In reality, we rarely encounter a situation where the feature is a single variable x. The features of a given model typically consists of several variables. We can employ the matrix notation to easily generalize the above picture to the case where we have a set of explanatory variables. Now, suppose we collect the following dataset

Observation	$\vec{x} \in \mathbb{R}^d$	y
1	$\vec{x}^{(1)}$	y_1
2	$\vec{x}^{(2)}$	y_2
3	$\vec{x}^{(3)}$	y_3
$\vdots$	$\vdots$	$\vdots$
n	$\vec{x}^{(n)}$	y_n

We now construct an $n \times (d+1)$ matrix $\mathbb{X}$ associated with the above dataset as follows:

$$\mathbb{X} = (\mathbb{1}, x_1, x_2, \cdots, x_d) = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \cdots & x_d^{(n)} \end{pmatrix} . \tag{4}$$

There are a couple of comments in order:

- Note that the subscript i in $x_i^{(j)}$ denotes the i-th feature among d features, whereas the superscript (j) in $x_i^{(j)}$ denotes the j-th observation for the i-th feature.
- A column of 1's has been inserted in matrix $\mathbb{X}$ (the first column in (4)) to capture the intercepts (*i.e.* bias term ω_0 in (1)) in a convenient way in matrix notation.

We can now represent the target variable by the $n \times 1$ matrix $\mathbb{Y}$. The linear regression model assumes a linear relation between the target $\mathbb{Y}$ and the features $\mathbb{X}$

$$\hat{\mathbb{Y}} = \mathbb{X} \cdot \omega^{\mathsf{T}} \,, \tag{5}$$

where matrix ω is a $1 \times (d+1)$ matrix, namely $\omega = (\omega_0, \omega_1, \cdots, \omega_d)$, that encapsulates the coefficients of the linear relation. As in equation (1), $\hat{\mathbb{Y}}$ indicates the *predicted values* of the target as opposed to the true value of the target, $\mathbb{Y}$. Mpreover, note that equation (5) is the analog of equation (1) in the case where we have more than one feature.

We can now define the cost function in a similar manner and proceed with its minimization. The cost function is given by

$$J(\omega) = \frac{1}{2}(\mathbb{Y} - \hat{\mathbb{Y}})^{\mathsf{T}} \cdot (\mathbb{Y} - \hat{\mathbb{Y}}) \tag{6}$$

Equation (6) should be thought as the analog of equation (2). The difference between the two is that in equation (6), the cost function J is the function of $d+1$ omega's $\omega = (\omega_0, \omega_1, \cdots, \omega_d)$. Notice that ω_0 denotes the bias term (*i.e.* the constant term) as usual. We now want to minimize the cost function $J(\omega)$. Replacing equation (5) for the predicted target $\hat{\mathbb{Y}}$ in equation (6), we obtain

$$J(\omega) = \frac{1}{2}(\mathbb{Y}^{\mathsf{T}} - \omega\, \mathbb{X}^{\mathsf{T}}) \cdot (\mathbb{Y} - \mathbb{X}\, \omega^{\mathsf{T}}) = \frac{1}{2}(\mathbb{Y}^{\mathsf{T}}\mathbb{Y} - 2\,\omega\, \mathbb{X}^{\mathsf{T}}\mathbb{Y} + \omega\, \mathbb{X}^{\mathsf{T}}\mathbb{X}\, \omega^{\mathsf{T}}) \,. \tag{7}$$

Differentiating with respect to ω and setting $\dfrac{\partial J}{\partial \omega} = 0$, we obtain

$$-\mathbb{X}^{\mathsf{T}}\mathbb{Y} + \mathbb{X}^{\mathsf{T}}\mathbb{X}\, \omega^{\mathsf{T}} = 0 \qquad \Longrightarrow \qquad \boxed{\boxed{\omega^{\mathsf{T}} = \left(\mathbb{X}^{\mathsf{T}}\mathbb{X}\right)^{-1} \mathbb{X}^{\mathsf{T}}\mathbb{Y}}} \tag{8}$$

The boxed equation in (8) is the **exact solution** of the linear regression problem! Also note that the superscript -1 in equation (8) indicates the *inverse matrix*.

Question: Does scikit-learn library use equation (8) to find the coefficients of linear regression?

Answer: No! It is computationally very costly to calculate the inverse of big matrices. The scikit-learn library finds the coefficients of regression (*i.e.* matrix ω) by approximation through an iterative process.

1.3 Assessing the Accuracy of the Model

Now we would like to see how we should assess the performance of the linear regression model. In order to assess the performance of the linear regression, we first need to define metrics by which the accuracy of the model can be measured. These measures are not exclusive to linear regression model, and can be used to assess the performance of any model whose job is the prediction of a continuous variable. There are several different metrics defined for assessing the performace of a given model, and in here we dicuss three important metrics, namely the RSE, the R^2 score, and the F-statistic.

1.3.1 Residual Standard Error (RSE)

The residual standard error is one metric to assess the performance of the linear regression. It is denoted by RSE and is defined by

$$RSE = \sqrt{\frac{1}{n-2}J(\omega)} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}\,, \tag{9}$$

where $J(\omega)$ is the cost function defined in (6). Minimization of the cost function through linear regression implies that RSE will assume its minimum value. In other words, any other linear model will have a higher value of RSE. Among linear models, the model with the smallest value of RSE is most favorable. If you construct two linear models, and they happen to have different values for RSE, the model with smaller RSE should be preferred over the other. Note that RSE *carries the same unit as the target variable.*

1.3.2 R^2 Statistic

One problem with RSE metric is that it is dimensionful, and it is not always clear what constitutes a good RSE. To address this issue, an alternative measure of the fit (*i.e.* R^2) is defined that takes the form of proportion, and hence, is dimensionless. R^2 statistic measures *what portion of variance is explained*, and $0 \leq R^2 \leq 1$. The closer R^2 is to 1, the more successful the linear model is in predicting the target variable. To define R^2, we need to define the Mean Standard Error MSE. First, let us recall the definition of variance from your statistics knowledge:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mu_y)^2\,, \tag{10}$$

where μ_y is the mean of y-variable (*i.e.* $\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i$). The mean standard error, MSE, is then defined by

$$MSE = \frac{1}{n}J(\omega) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\,. \tag{11}$$

Note that MSE is different from the cost function J only by a factor of $\frac{1}{n}$. The R^2 score/statistic is then defined by

$$R^2 = 1 - \frac{MSE}{\sigma^2}\,. \tag{12}$$

Note that for the perfect linear model ($RSE = 0$), we have $R^2 = 1$. If $R^2 = 0.6$ for instance, then this *implies that* 60% *of the variance is explained by the model.*

1.3.3 F-Statistic

The F-statistic is used to verify if there is a relationship between the response/target variable and the features. In other words, the F-statistic performs a hypothesis testing with the following null and alternative hypotheses. The **null hypothesis** states that *there is no relationship between the target and features*

$$H_0 : \forall i \in \{1, 2, \cdots, d\} \quad \omega_i = 0\,, \tag{13}$$

while the **alternative hypothesis** states that *there is a relationship between the target and features*

$$H_1 : \exists i \in \{1, 2, \cdots, d\} \quad \text{such that} \quad \omega_i \neq 0\,. \tag{14}$$

Note that the above hypotheses, H_0 and H_1, do not concern the bias term ω_0. The hypothesis testing is performed by computing the F-statistic

$$F = \frac{(n-d-1)(\sigma^2 - MSE)}{d\,MSE} = \frac{n-d-1}{d}\frac{R^2}{1-R^2}\,. \tag{15}$$

Note that just like R^2 statistic, the F-statistic is a dimensionless quantity. Moreover, the F-statistic is always greater than 1 (*i.e.* $F > 1$). If the *F-statistic is a number close to 1*, you should reject the alternative hypothesis (This means that H_0 is true and there is no relationship between the target and the features). On the other hand, if F-statistic is greater than 1, then you should reject the null hypothesis and accept the alternative hypothesis (This means that there is a relationship between the target and the features).

1.4 Pearson Correlation

Although F-statistic can beused to decide whether there is a relationship between a target variable and a set of features, the F-statistic is unable to determine the role of dominant features when there is a relationship between the target and features. In other words, we need a **feature selection criterion**. To compare the significance of different features, one needs a more effective statistical tool. It turns out that the Pearson correlation enables us to compare the significance of different variables in an effective way.

To introduce the notion of correlation, we do not take a rigorous statistical approach. Rather we introduce Pearson correlation in a practical manner. Suppose X and Y are two random variables (Roughly speaking, this implies that they are associated with probability distributions). Then, the (Pearson) correlation between X and Y is defined by

$$\text{corr}[x, y] = \frac{1}{n}\frac{\text{cov}[x, y]}{\sigma_x \sigma_y} \in [-1, 1]\,, \tag{16}$$

where σ_x and σ_y refer to the standard deviations of variables X and Y, respectively. Moreover, the covariance between variables X and Y in (16) is defined by

$$\mathrm{cov}[x,y] = \sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)\ . \tag{17}$$

There are a couple of important comments in order:

- Correlation $\mathrm{corr}[x,y]$ between two (random) varianles X and Y is always a number between -1 and 1 (*i.e.* $-1 \leq \mathrm{corr}[x,y] \leq 1$).
- A value close to 0 shows that there is little correlation between X and Y. The extreme case $\mathrm{corr}[x,y] = 0$ shows that there is absolutely no correlation between X and Y, and the two variables are independent.
- A positive value for the correlation ($\sim\ \mathrm{corr}[x,y] > 0.2$) shows a positive correlation between X and Y (*i.e.* Y increases as X increases and vice versa). The higher $\mathrm{corr}[x,y]$ is, the stronger the correlation between X and Y is.
- A negative value for the correlation ($\sim\ \mathrm{corr}[x,y] < -0.2$) shows a negative correlation between X and Y (*i.e.* Y increases as X decreases and vice versa). The smaller $\mathrm{corr}[x,y]$ is, the stronger the negative correlation between X and Y is.
- $\mathrm{corr}[x,y] = +1$ and $\mathrm{corr}[x,y] = -1$ correspond to perfect linear correlation between X and Y.

We can employ the (Pearson) correlation to find the dominant features that affect the target variable of the model. Moreover, (Pearson) correlation can be used to detect strong correlations among features themselves.

1.5 Spearman Rank Correlation Coefficient

The Pearson correlation coefficient determines to what extent a linear model of the form $y = m\,x + b$ between the target and features can fit the observed data. This *generally* does a good job in measuring the similarity between the variables, but it is possible to construct pathological examples where the correlation coefficient between X and Y is zero, yet Y is completely dependent on (and hence perfectly predictable from) X. Therefore, it is essential to measure correlation between X and Y through different benchmarks.

Another important quantity that measures correlation between two variables is the **Spearman correlation coefficient** (Note that there exists a very similar notation, called the **Kendall correlation coefficient**, to Spearman correlation. The two notions are very similar, and in hence, we only talk about one of them). The Spearman correlation coefficient essentially counts the number of pairs of data points which are *out of order*. Assume that our data set contains points (x_1, y_1) and (x_2, y_2) where $x_1 < x_2$ and $y_1 < y_2$. This is a vote that the values are positively correlated, whereas the vote would be for a negative correlation if $y_2 < y_1$. More specifically, the Spearman rank correlation coefficient ρ is defined by

$$\rho = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}\ , \tag{18}$$

where $d_i = \mathrm{rank}(x_i) - \mathrm{rank}(y_i)$. To see how the above formula works, let us consider the following baby example. Consider the following dataset.

x_i	y_i	rank(x_i)	rank(y_i)	d_i	d_i^2
5	6	3	2	1	1
2	14	1	3	-2	4
4	-2	2	1	1	1
8	17	4	4	0	0

We have basically ranked x and y columns, and calculated d_i for each row. Now we can use equation (18) to calculate the correlation coefficient.

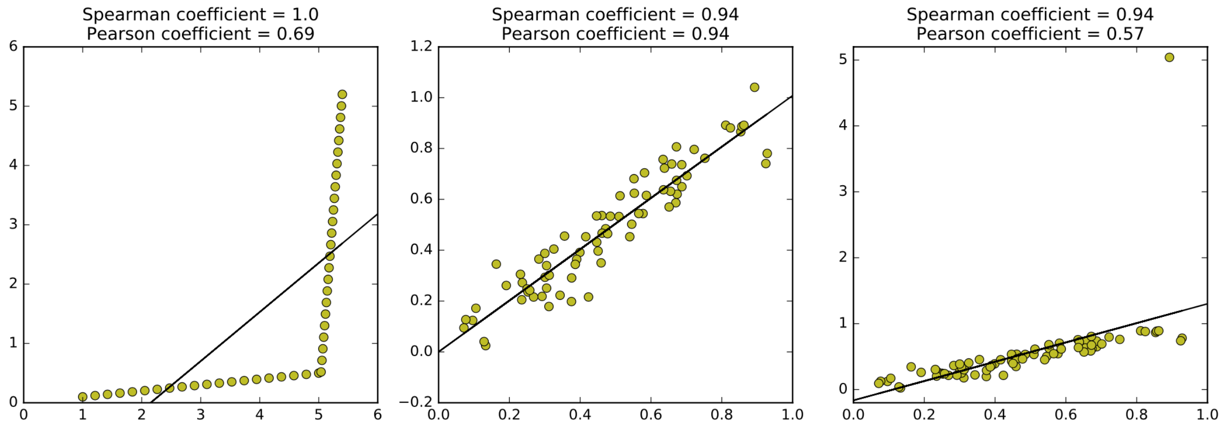
$$\rho = 1 - \frac{6(1+4+1+0)}{4(16-1)} = 1 - \frac{36}{60} = \frac{2}{5} = 0.4\,. \tag{19}$$

We can easily calculate the Spearman (and the Kendall) correlation coefficients through the scipy library. Let us quickly see how this is done.

[1]:
```python
# Import the necessary libraries

from numpy.random import rand
from scipy.stats import spearmanr    # For Spearman correlation
from scipy.stats import kendalltau   # For Kendall correlation

# Create a toy dataset

x = rand(1000) * 20          # create a feature x as numpy array

y = x + (rand(1000) * 5)     # creat a target y by adding some noise to x

# Calculate the coefficient rho and its associated p-value

rho, p = spearmanr(x, y)

print('Spearmans correlation coefficient rho = %.4f' % rho, '\n')
print('Spearmans correlation p-value = %.5f' % p, '\n')

# If p-value < alpha (level of significance) H_0 is rejected --> There is␣
 ↪correlation between x and y.

# Note H_0: There is no correlation between x and y.
# Note H_1: x and y are correlated.

# Similarly, you can calculate the Kendall correlation

tau, p = kendalltau(x, y)

print('Kendall correlation coefficient tau = %.4f' % tau, '\n')
print('Kendall correlation p-value = %.5f' % p)
```

```
Spearmans correlation coefficient rho = 0.9708

Spearmans correlation p-value = 0.00000

Kendall correlation coefficient tau = 0.8418

Kendall correlation p-value = 0.00000
```

In practice, one should consider both the Pearson and the Spearman (or Kendall) correlations. The combination of the two will provide more insight about your dataset. Consider the following plots.



Question: As observed in above plots, the middle plot receives high Pearson correlation and Spearman correlation scores. Why are the Pearson and Spearman correlation coefficients for the left and right plots very different?

We finish the discussion on correlation by reminding you an important fact from statistics:

Warning: Correlation does not imply causation!

You should keep the above fact in mind whenever you work with the concept of correlation.

1.6 Bias-Variance Trade-Off

In statistics, the contrast between complexity and performance is known as the **bias-variance trade-off**. Let us review the two concepts briefly:

- **Bias** is an error that is caused by *assuming incorrect assumptions* that built into the model. For instance, assuming that a nonlinear relation between a target and features is linear. Bias can be associated with *prejudice*. In this case, models perform on both training and test datasets lousy, and they are *underfit*.

- **Variance** is a sensitivity that is caused by fluctuations in the training dataset. If the training set contains sampling or measurement errors, this noise induces variance in the resulting model. Errors of variance lead to *overfit* models. The search of overfit models for accuracy causes them to mistake noise for actual signal. They adjust so well to the training data that noise leads them astray. Models that *perform much better on the training dataset than the testing dataset are overfit*.

2 Coding for Regression Analysis

[2]:
```
# Loading Boston housing dataset

from sklearn.datasets import load_boston
```

[3]:
```
# Finding the description of the involved variables

boston = load_boston()
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people
by town
```

```
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

[4]:
```python
# Defining a dataframe for the Boston housing dataset

import numpy as np
import pandas as pd

boston_df = pd.DataFrame(boston.data)         # Defining dataframe boston_df
boston_df.columns = boston.feature_names      # Defining the headers of the␣
 ↪dataframe
boston_df['PRICE'] = boston.target            # Adding Price column

boston_df.head()                              # Viewing the first few rows of␣
 ↪dataframe
```

[4]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  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
```

Calculating Pearson Correlation between Variables

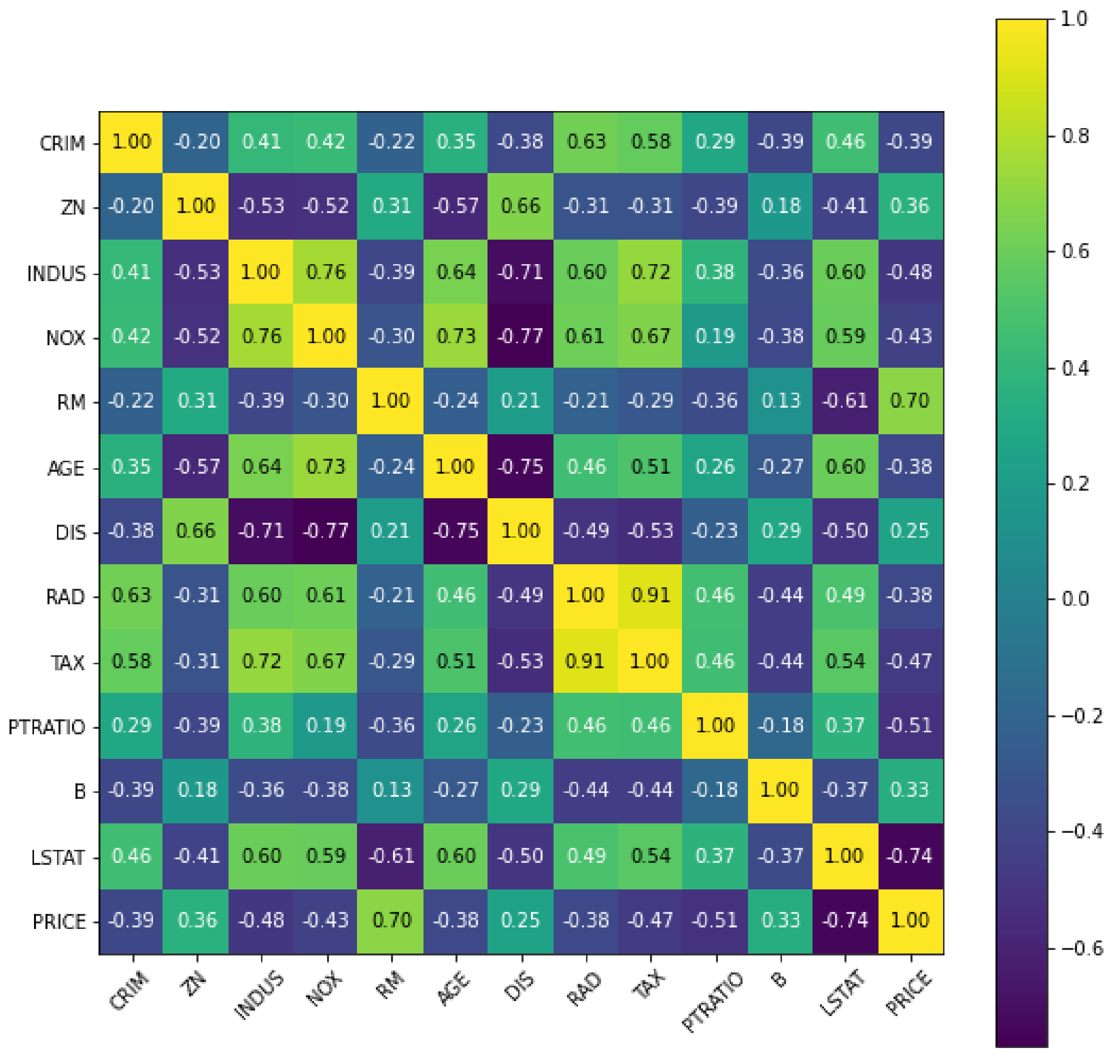
Note that `CHAS` is a categorical variable, and we do not include it in the regression analysis.

[5]:
```python
# Dropping 'CHAS' column

boston_df = boston_df.drop(['CHAS'], axis=1)

boston_df.head()
```

[5]:
```
      CRIM    ZN  INDUS    NOX     RM   AGE     DIS  RAD    TAX  PTRATIO  \
0  0.00632  18.0   2.31  0.538  6.575  65.2  4.0900  1.0  296.0     15.3
1  0.02731   0.0   7.07  0.469  6.421  78.9  4.9671  2.0  242.0     17.8
2  0.02729   0.0   7.07  0.469  7.185  61.1  4.9671  2.0  242.0     17.8
3  0.03237   0.0   2.18  0.458  6.998  45.8  6.0622  3.0  222.0     18.7
4  0.06905   0.0   2.18  0.458  7.147  54.2  6.0622  3.0  222.0     18.7

        B  LSTAT  PRICE
0  396.90   4.98   24.0
1  396.90   9.14   21.6
2  392.83   4.03   34.7
3  394.63   2.94   33.4
4  396.90   5.33   36.2
```

[6]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import mlxtend
from mlxtend.plotting import heatmap

cols = boston_df.columns      # List of colmuns of dataframe boston_df

cm = np.corrcoef(boston_df[cols].values.T)      # Calculate Pearson correlation
hm = heatmap(cm, figsize=(10,10), row_names=cols, column_names=cols)   #␣
 ↪Represent correlation by a heat map
```

```
plt.show()
```



	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
CRIM	1.00	-0.20	0.41	0.42	-0.22	0.35	-0.38	0.63	0.58	0.29	-0.39	0.46	-0.39
ZN	-0.20	1.00	-0.53	-0.52	0.31	-0.57	0.66	-0.31	-0.31	-0.39	0.18	-0.41	0.36
INDUS	0.41	-0.53	1.00	0.76	-0.39	0.64	-0.71	0.60	0.72	0.38	-0.36	0.60	-0.48
NOX	0.42	-0.52	0.76	1.00	-0.30	0.73	-0.77	0.61	0.67	0.19	-0.38	0.59	-0.43
RM	-0.22	0.31	-0.39	-0.30	1.00	-0.24	0.21	-0.21	-0.29	-0.36	0.13	-0.61	0.70
AGE	0.35	-0.57	0.64	0.73	-0.24	1.00	-0.75	0.46	0.51	0.26	-0.27	0.60	-0.38
DIS	-0.38	0.66	-0.71	-0.77	0.21	-0.75	1.00	-0.49	-0.53	-0.23	0.29	-0.50	0.25
RAD	0.63	-0.31	0.60	0.61	-0.21	0.46	-0.49	1.00	0.91	0.46	-0.44	0.49	-0.38
TAX	0.58	-0.31	0.72	0.67	-0.29	0.51	-0.53	0.91	1.00	0.46	-0.44	0.54	-0.47
PTRATIO	0.29	-0.39	0.38	0.19	-0.36	0.26	-0.23	0.46	0.46	1.00	-0.18	0.37	-0.51
B	-0.39	0.18	-0.36	-0.38	0.13	-0.27	0.29	-0.44	-0.44	-0.18	1.00	-0.37	0.33
LSTAT	0.46	-0.41	0.60	0.59	-0.61	0.60	-0.50	0.49	0.54	0.37	-0.37	1.00	-0.74
PRICE	-0.39	0.36	-0.48	-0.43	0.70	-0.38	0.25	-0.38	-0.47	-0.51	0.33	-0.74	1.00

The above matrix shows that LSTAT and RM have the highest correlations with the target variable PRICE. Therefore, as our first step in our regression, we identify LSTAT and RM as the two features to explain the target PRICE. Now, let's check the Spearman correlation. This is a cross check to make sure that we have not missed any inluential variable (on PRICE) that Pearson correlation hasn't been able to detect.

[7]:
```
# Calculate Spearman correlation coefficient between 'PRICE' and each feature

for col in cols[:-1]:
    rho, p = spearmanr(boston_df[col].values, boston_df['PRICE'].values)
    print('Spearman correlation between PRICE and %s is %s' %(col, round(rho,␣
 ↪4)))
```

```
Spearman correlation between PRICE and CRIM is -0.5589
Spearman correlation between PRICE and ZN is 0.4382
Spearman correlation between PRICE and INDUS is -0.5783
Spearman correlation between PRICE and NOX is -0.5626
Spearman correlation between PRICE and RM is 0.6336
Spearman correlation between PRICE and AGE is -0.5476
Spearman correlation between PRICE and DIS is 0.4459
Spearman correlation between PRICE and RAD is -0.3468
Spearman correlation between PRICE and TAX is -0.5624
Spearman correlation between PRICE and PTRATIO is -0.5559
Spearman correlation between PRICE and B is 0.1857
Spearman correlation between PRICE and LSTAT is -0.8529
```

It is again observed that the dominant effect comes from the two variables `LSTAT` (with $\rho = -0.8529$) and `RM` (with $\rho = 0.6336$). The Pearson and Spearman correlations agree on the role of the dominant variables.

Defining the Variables of the Model

[8]:
```
# Defining the features and the target of the model

X = boston_df[['LSTAT', 'RM']]      # Features
y = boston_df.PRICE                 # Target
```

The linear model is now $y = \omega_1 X_1 + \omega_2 X_2 + \omega_0$, where X_1 and X_2 refer to `LSTAT` and `RM`, respectively. ω_1 and ω_2 are the coefficients of regression, and the constant ω_0 is the intercept (*i.e.* bias term).

[9]:
```
# Breaking the data into train and test subsets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=3)
```

2.1 Performing Regression through scikit-learn

[10]:
```
# Importing 'LinearRegression' through linear_model module

from sklearn.linear_model import LinearRegression

reg = LinearRegression()            # Instantiate
reg.fit(X_train, y_train)           # Fit the train data

r2_train_score = reg.score(X_train, y_train)    # Calculating R^2 score for train

print('R^2 score for train dataset = ', round(r2_train_score, 4), '\n')
print('Coefficients of Linear Model:', reg.coef_, '\n')
print('Intercept:', reg.intercept_)
```

```
R^2 score for train dataset =  0.6469

Coefficients of Linear Model: [-0.6965145  4.8839348]

Intercept: 0.79005520961854
```

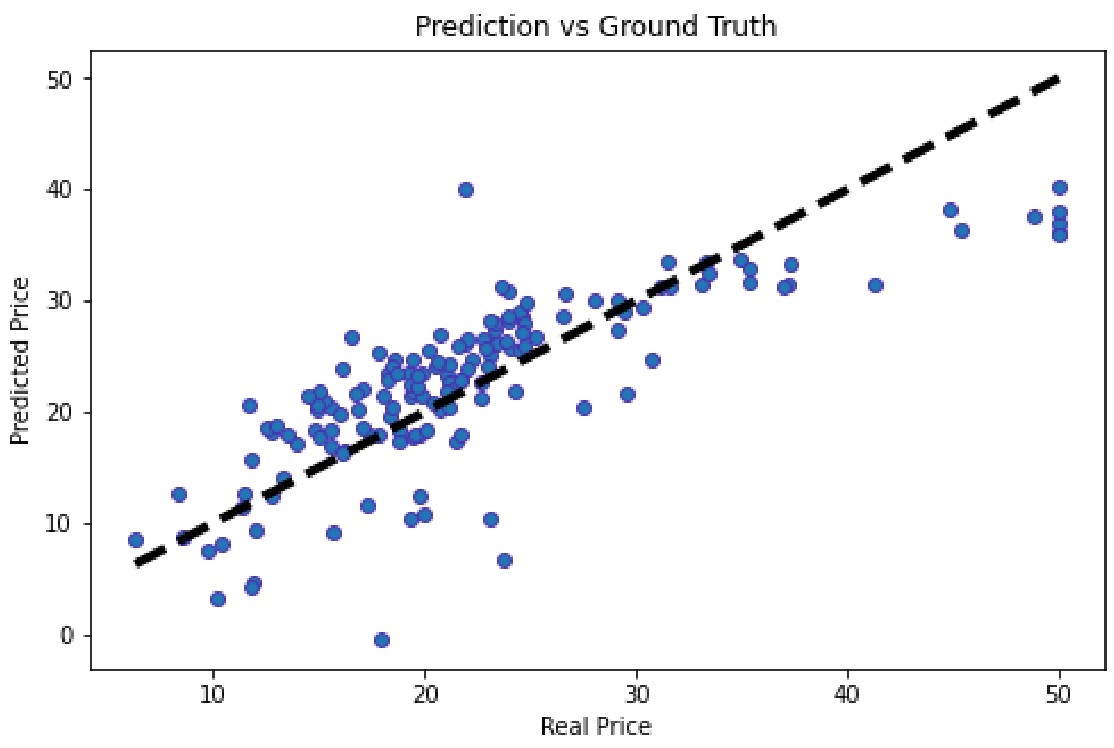
[11]:
```
# Finding the predictions of the model for test dataset

y_pred = reg.predict(X_test)

y_pred[:10]  # Representing the price prediction for the first 10 data points in␣
 ↪test dataset
```

[11]:
```
array([38.27709025, 22.14882389, 25.23733249, 31.50462753, 26.04502872,
       23.25264165, 19.60807021,  8.13929916, 25.19314285, 10.80728302])
```

[12]:
```
# Evaluating the performance of the model on the test dataset

r2_test_score = reg.score(X_test, y_test)   # Calculating R^2 score for train

print('R^2 score for test dataset = ', round(r2_test_score, 4), '\n')
```

```
R^2 score for test dataset =  0.6117
```

[13]:
```
# Let's plot predictions vs ground truth for 'PRICE'

fig, ax = plt.subplots(figsize=(8,5))

ax.scatter(y_test, y_pred, edgecolors=(0.3, 0.2, 0.7))  # Scatter plot for␣
 ↪predictions vs truth
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)␣
 ↪ # Draw line y=x

ax.set_xlabel('Real Price')
ax.set_ylabel('Predicted Price')
plt.title('Prediction vs Ground Truth', fontdict=None, loc='center')
plt.show()
```



The closer a point in above plot is to the $y = x$ line, the more realistic the prediction for the `PRICE` is.

2.2 Performing Regression through StatsModels

An alternative to scikit-learn library is StatsModels. We now solve the same problem using StatsModels.

[14]:
```
# Importing the statsmodels api

import statsmodels.api as sm
```

[15]:
```
XC_train = sm.add_constant(X_train)       # Creating the bias term omega_0
model = sm.OLS(y_train, XC_train)         # Defining model and the data into the␣
 ↪model
results = model.fit()                     # Fitting the data into the model

print('Summary:\n', results.summary())   # Printing the summary of results
```

```
Summary:
                             OLS Regression Results
==============================================================================
Dep. Variable:                  PRICE   R-squared:                       0.647
```

```
Model:                            OLS   Adj. R-squared:                  0.645
Method:                 Least Squares   F-statistic:                     321.5
Date:                Sat, 25 Jun 2022   Prob (F-statistic):           4.56e-80
Time:                        12:45:34   Log-Likelihood:                -1108.2
No. Observations:                 354   AIC:                             2222.
Df Residuals:                     351   BIC:                             2234.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7901      3.855      0.205      0.838      -6.793       8.373
LSTAT         -0.6965      0.052    -13.412      0.000      -0.799      -0.594
RM             4.8839      0.543      8.994      0.000       3.816       5.952
==============================================================================
Omnibus:                      122.682   Durbin-Watson:                   1.746
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              425.653
Skew:                           1.530   Prob(JB):                     3.72e-93
Kurtosis:                       7.416   Cond. No.                         205.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[16]:
```python
XC_test = sm.add_constant(X_test)      # Adding bias for the test set
y_pred_sm = results.predict(XC_test)   # Finding the predictions for the test set

print(y_pred_sm.values[:10], '\n')     # Printing the predictions for the test␣
 ↪set

# Let us compare the predictions of StatsModels with Scikit-learn

rd_array = np.vectorize(round)                           # Rounding numpy arrays
print(rd_array(y_pred[:20] - y_pred_sm.values[:20], 6))   # Subtract predictions␣
 ↪of StatModels from Scikit-learn
```

```
[38.27709025 22.14882389 25.23733249 31.50462753 26.04502872 23.25264165
 19.60807021  8.13929916 25.19314285 10.80728302]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

The above results show that the predictions of the StatsModels are in perfect agreement with those of scikit-learn, as they should!

2.3 Performing Regression using the Exact Formula

As you noticed, this dataset is not that big in size. Hence, we may find the *exact result* of regression through the exact formula we derived in the lecture: $\omega^{\mathsf{T}} = \left(\mathbb{X}^{\mathsf{T}}\mathbb{X}\right)^{-1}\mathbb{X}^{\mathsf{T}}\mathbb{Y}$. We can easily implement this formula into code using numpy.

[17]:
```
# Defining a function to calculate omega from X and Y

from numpy.linalg import inv     # Import 'inv' to calculate inverse matrix

def exact_reg(X, Y):
    X = np.insert(X, 0, np.ones(X.shape[0]), axis=1)              # Adding a␣
 ↪column of 1's
    OmegaT = np.matmul(inv(np.matmul(X.T, X)), np.matmul(X.T, Y))  # Using the␣
 ↪exact formula
    Y_hat = np.matmul(X, OmegaT)                                   # Predicted␣
 ↪values of y
    return OmegaT, Y_hat
```

[18]:
```
# Let's compare the exact results for coefficients and intercept with the␣
 ↪results of scikit-learn

print('The intercept and coefficients from exact formula:', exact_reg(X_train.
 ↪values, y_train.values)[0],'\n')

print('The intercept and coefficients from scikit-learn:', np.insert(reg.coef_,␣
 ↪0, reg.intercept_))
```

```
The intercept and coefficients from exact formula: [ 0.79005521 -0.6965145
4.8839348 ]

The intercept and coefficients from scikit-learn: [ 0.79005521 -0.6965145
4.8839348 ]
```

[19]:
```
# Calculate the exact R^2 scores

y_hat_train = exact_reg(X_train.values, y_train.values)[1]     # Calculating␣
 ↪predictions for train data
y_hat_test = exact_reg(X_test.values, y_test.values)[1]        # Calculating␣
 ↪predictions for test data

from sklearn.metrics import r2_score                           # Importing␣
 ↪r2_score from metrics module

print('Exact train R^2 score = ', round(r2_score(y_train.values, y_hat_train),␣
 ↪6), '\n')
print('Exact test R^2 score = ', round(r2_score(y_test.values, y_hat_test), 6))
```

```
Exact train R^2 score =  0.646888
```

```
Exact test R^2 score =  0.629446
```

The above R^2 scores are in perfect agreement with those obtained by scikit-learn!

```
[ ]:
```