Data Analysis and Machine-Learning

Chapter 2.2:

Linear Regression Model (2): Error Functions and Differentiation



1. Methodologies for Estimating Regression Coefficients

There are several verified methodologies to estimate beta coefficients for a given regression model.

To illustrate, consider the sample that we generated in chapter 2.1:

x = np.array([30, 31, 32, 33, 34])

y = np.array([19, 22, 32, 47, 52])

When the sample regression function for this sample is estimated as:

# Gauss elimination method (simultaneous equations)

# Linear algebraic method

It is also possible to solve directly by calculating the inner product of y on the inverse matrix of set X.

Finally, there is yet another powerful method for accurately estimating the coefficients, namely, an iterative solving method (an application of an iteration algorithm). Understanding this methodology becomes the key for understanding machine-learning mechanisms.

There are several valid approaches in applying iteration method, including Jacobi method, Gauss-Seidal method, steepest descent, gradient descent, newton method, etc. In the following chapters, we will focus particularly on the gradient descent method, a framework algorithm in machine-learning. Beforehand, it is essential to understand the fundamental concepts of (1) error functions and (1) estimation of linear coefficients using differentiation, which is what this chapter is about.

1. Error Functions of a Linear Model

For a sample provided earlier, let us assume that the model is estimated as follows:

x = np.array([30, 31, 32, 33, 34])

y = np.array([19, 22, 32, 47, 52])

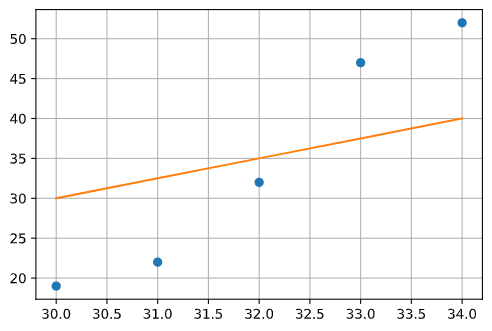
Visualize:

plt.plot(x, y, 'o')

plt.plot(x, 2.5 \* x -45)

plt.grid()

plt.show()



The sum of differences (distance) between the estimated line and the actual dots (the actual value of Y) become the errors of the estimated model.

More specifically:

There are two general methodologies to measure the values of errors, namely, the LAD (Least Absolute Deviation) method and LSE (Least Squared Error) method. Naturally, an analyst’s goal using these methods would be to find the parameters “a” and “b” that minimize the functional value of error functions. In actual applications, LSE is usually the most frequently used methodology as the square values allow for differentials.

\*Various estimation approaches on errors and residuals:

OLS(Ordinary Least Squared)/LSE(Least Squared Error/Estimation)

LAD (Least Absolute Deviation)

MSE(Mean Squared Error): takes average of LSE

#e.g.

yhat = beta[0] \* x + beta[1]

np.mean((y - yhat) \*\* 2)

MAD(Mean Absolute Deviation): divides the sum of absolute values of residuals by the total number of samples

#e.g.

yhat = beta[0] \* x + beta[1]

np.mean(np.abs((y - yhat)))

LAD (\*L1 Loss):

LSE (\*L2 Loss):

Now, let us dive deeper into the concept of error functions. Assume the following error function as our sample:

Our goal would be to find the value of X that minimizes the functional value E(x).

If we visualize this:

# Define the sample error function

def error(x):

  return x \*\* 2 - 5 \* x + 10

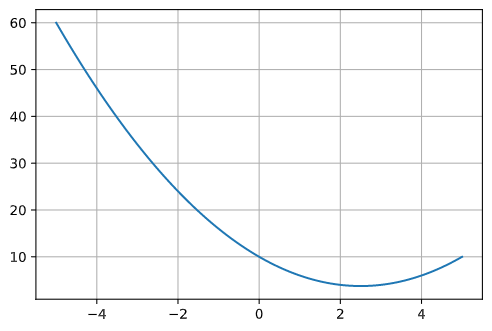
# Visualize:

x = np.linspace(-5, 5, 100)

plt.plot(x, error(x))

plt.grid()

plt.show()



Our goal would be to locate the smallest-value point in such convex function:

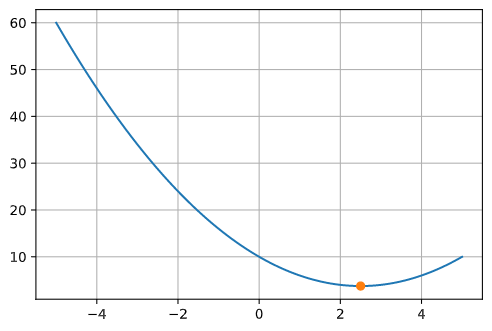
x = np.linspace(-5, 5, 100)

plt.plot(x, error(x))

plt.plot(2.5, error(2.5), 'o')

plt.grid()

plt.show()



For a convex function, the minimization point would be the coordinate where the slope of the tangent becomes 0, as follows:

x = np.linspace(-5, 5, 100)

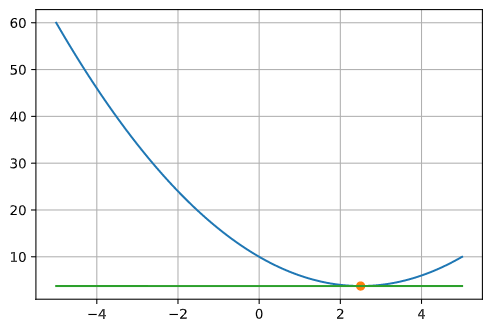
plt.plot(x, error(x))

plt.plot(2.5, error(2.5), 'o')

plt.plot(x, 0\*x+error(2.5))

plt.grid()

plt.show()



Now, in order to locate the exact point that minimizes the value of the error function, it is first necessary to understand the concept of differentiation.

1. Differentiation and Derivatives

Differentiation allows us to find out the instantaneous rate of changes in slopes of a quadratic function. To illustrate, let us start by considering the following equation to explain the process of finding the rate of changes in Y for changes in X:

If we minimize the value of changes in X to the point of an “instant” (extremity, i.e., *lim*), the above function can be restated as:

For better understanding, consider the differentiation process of the following sample function:

When:

Differentiation function:

When applying the sample f(x) function:

Thus, it is now possible to locate the X coordination point in a convex function that minimizes the value of Y, that is, the point where differential coefficient equals to 0 (i.e., when the instantaneous slope of the function equals to 0).

Returning back to the sample error function defined previously,

def error(x):

    return x\*\*2 - 5\*x + 10

# Differentiate:

import sympy as sp

x = sp.Symbol('x')

f = x \*\* 2 - 5 \* x + 10

sp.diff(f, x)

Output:

2x-5

The coordination of X that minimizes the value of Y (i.e., error) would be **2.5**, because 2 \* 2.5 – 5 = 0.

#Visualize:

x = np.linspace(-5, 5, 100)

plt.plot(x, error(x))

plt.plot(2.5, error(2.5), 'o', label='Least error: "differential coefficient=0"')

plt.plot(x, 0\*x+error(2.5), label='instantaneous slope=0')

plt.legend()

plt.grid()

plt.show()

