Data Analysis and Machine-Learning

Chapter 2.3:

Linear Regression Model (3): Gradient Descent



1. Defining Formula for the Gradient Descent Methodology

In chapter 2.2, I have illustrated the process of locating the X coordination point that minimizes the value of Y (i.e., the point where differential coefficient equals to 0, that is, when the instantaneous slope of a function equals to 0) in a convex-shaped error function.

Such differentiation method can be mathematically concise, nonetheless with one limitation: the information regarding the equation of an error function is essential to actually conduct differentiations. Then, what if there is no prior information given regarding the functional equation, or if the equations are simply too complicated to solve via differentiations?

For such cases, there is yet another powerful method to find out the target parameters in an error function, namely a gradient descent method, which is one type of iterative solving method. The formula can be defined as follows.

To explain the given formula in detail, recall that in chapter 2.2, I have illustrated that the derivative of a function gives us the instantaneous slope of a function. What we want to know is the point () where the value of error (Y value) is minimized. Under gradient descent method, we first designate a starting point () in a function, then “descend” gradually following the changes in the instantaneous slope of a function (by ) until we can locate the minimizing point in a given error function; this is why there is a negative sign, because the function in hand is shaped as a convex. Similarly for the case of , where we artificially designate a starting point for the descent, the value of is also designated, which we refer to as the “learning rate” in machine-learning.

1. Application of Gradient Descent on Error Functions

Now, let us apply the above formula to a sample error function that we have created in chapter 2.2.

Suppose a linear regression model defined as:

The error function (LSE) for this function would be:

Now, assume the following function as our sample error function:

# Define error function:

def error(x):

  return x \*\* 2 - 5 \* x + 10

# Define gradient function ) for error function:

def grad\_fx(x):

  return 2 \* x - 5

# Designate starting point (), learning rate (), and max iterations for gradient descent

x0 = -4

lr = 0.01

maxIter = 1000

for i in range(maxIter):

  x1 = x0 - lr \* grad\_fx(x0)

  x0 = x1

print('error: {}, grad: {}, x: {}'.format(error(x0), grad\_fx(x0), x0))

Output:  
error: 3.75, grad: -2.187857361946044e-08, x: 2.499999989060713

As we can see, the estimated x parameter (x) is analogous to the differentiation method that we have seen in chapter 2.2.

1. Visualization of the Gradient Descent

For better understanding, let us visualize the estimation process of a gradient descent.

#Append estimation process in an empty list

x0 = -4

lr = 0.01

maxIter = 1000

descent = []

for \_ in range(maxIter):

  descent.append(x0)

  x1 = x0 - lr \* grad\_fx(x0)

  x0 = x1

#Visualize

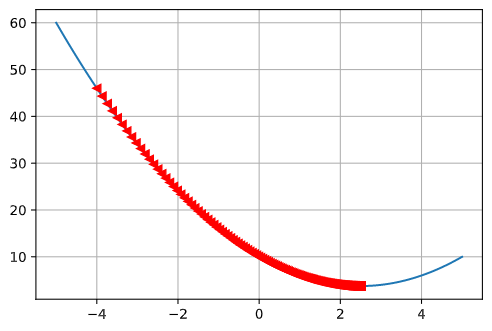
descent = np.array(descent)

plt.plot(x, error(x))

plt.plot(descent, error(descent), '<', color='r')

plt.grid()

plt.show()



1. Changes in Error Functions

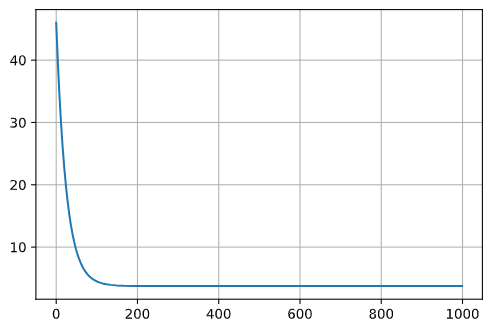
For real-world models, it is usually difficult to judge if the estimation process of a gradient descent is going smoothly, because visualizations are difficult for models with multiple parameters. However, keep in mind that regardless of the numbers of parameters, the value of function is always single. This is why visualizations of changes in error functions are useful to judge if the optimization and estimation processes are going well.

#plot changes in error functions

plt.plot(error(descent))

plt.grid()

plt.show()



As it can be implied from the error function graph, we can see that the values of errors are decreasing smoothly throughout iterations, which means that the optimization process for our estimation is proceeding well.

1. Error Functions with Multiple Variables

Lastly, let us assume a function with binomial parameters, as follows:

The differentiation of this error function would be:

a = sp.Symbol('a')

b = sp.Symbol('b')

f = (a + 4) \*\* 2 + (b + 3) \*\* 2

display(f)

display( sp.diff(f, a) )

display( sp.diff(f, b) )

#Define error function and gradient functions

def error(a, b):

  return (a - 2) \*\* 2 + (b - 2) \*\* 2

def grad\_fa(a, b):

  return 2 \* a + 8

def grad\_fb(a, b):

  return 2 \* b + 6

# Gradient Descent

beta0 = np.array([-80, 80])

lr = 0.001

maxIter = 1000

DC = []

for \_ in range(maxIter):

  DC.append(beta0)

  grad = np.array([ grad\_fa(\*beta0), grad\_fb(\*beta0) ])

  beta1 = beta0 - lr \* grad

  beta0 = beta1

#Visualization

fig = plt.figure(figsize = (10, 10))

ax = fig.add\_subplot(projection='3d')

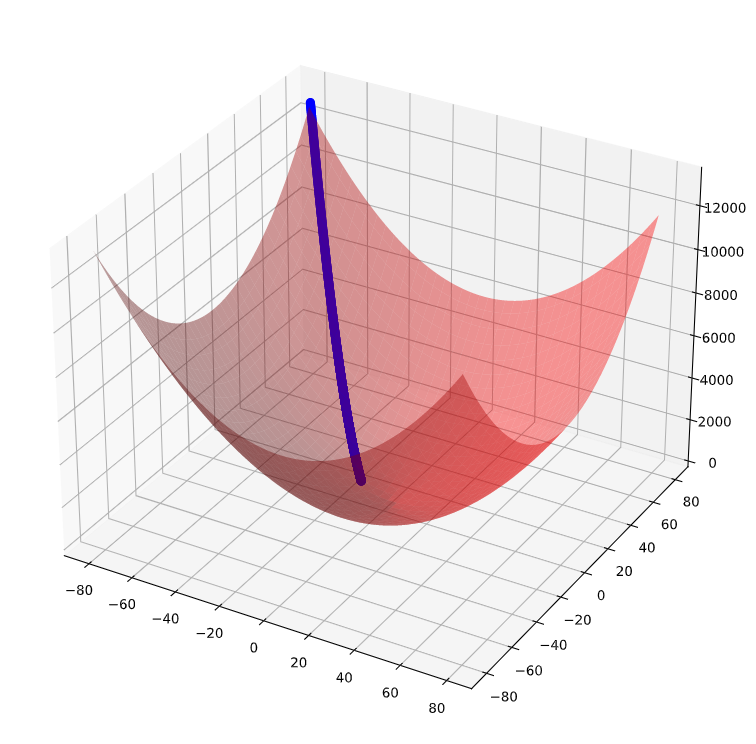
xx1, xx2 = np.meshgrid(np.arange(-80, 80), np.arange(-80, 80))

ER = error(xx1, xx2)

ax.plot\_surface(xx1, xx2, ER, alpha=0.4, color='r')

ax.plot(np.array(DC)[:,0], np.array(DC)[:,1], error(np.array(DC)[:,0], np.array(DC)[:,1]), 'o-', color='b')

plt.show()



As implied from the error function graph, we can see that the values of errors are decreasing smoothly throughout iterations (blue dots), which means that the optimization process for our estimation is proceeding well.

1. OLS Summaries

Now that we have understood the concepts and frameworks of a linear regression model, here I introduce some convenient tools in python to examine various statistical summaries at a glance.

#Import samples: diabetes

from sklearn import datasets

data = datasets.load\_diabetes()

#Split features and target, and make dataframes of Xs and Y

features = pd.DataFrame( data['data'], columns=data['feature\_names'] )

target = pd.DataFrame(data['target'], columns=['target'])

#Import statsmodels for statistical summaries

from statsmodels.formula.api import ols

model = ols("target ~ features", data).fit()

print(model.summary())

Output:

OLS Regression Results

==============================================================================

Dep. Variable: target R-squared: 0.518

Model: OLS Adj. R-squared: 0.507

Method: Least Squares F-statistic: 46.27

Date: Prob (F-statistic): 3.83e-62

Time: Log-Likelihood: -2386.0

No. Observations: 442 AIC: 4794.

Df Residuals: 431 BIC: 4839.

Df Model: 10

Covariance Type: nonrobust

===============================================================================

coef std err t P>|t| [0.025 0.975]

-------------------------------------------------------------------------------

Intercept 152.1335 2.576 59.061 0.000 147.071 157.196

features[0] -10.0122 59.749 -0.168 0.867 -127.448 107.424

features[1] -239.8191 61.222 -3.917 0.000 -360.151 -119.488

features[2] 519.8398 66.534 7.813 0.000 389.069 650.610

features[3] 324.3904 65.422 4.958 0.000 195.805 452.976

features[4] -792.1842 416.684 -1.901 0.058 -1611.169 26.801

features[5] 476.7458 339.035 1.406 0.160 -189.621 1143.113

features[6] 101.0446 212.533 0.475 0.635 -316.685 518.774

features[7] 177.0642 161.476 1.097 0.273 -140.313 494.442

features[8] 751.2793 171.902 4.370 0.000 413.409 1089.150

features[9] 67.6254 65.984 1.025 0.306 -62.065 197.316

==============================================================================

Omnibus: 1.506 Durbin-Watson: 2.029

Prob(Omnibus): 0.471 Jarque-Bera (JB): 1.404

Skew: 0.017 Prob(JB): 0.496

Kurtosis: 2.726 Cond. No. 227.

==============================================================================

Notes:

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

As the table depicts, the OLS summaries provide various statistical insights regarding the model including model’s general accuracy (R2, adjusted R2), standard errors, beta coefficients (weights of Xs), intercept, and insights regarding statistical significance (F-statistic, t-test, and P-values).

1. Inferential Statistics
   1. R-Squared

For intuitive understanding of the equation, consider following samples for illustrations.

#Generate Samples

x = np.array([60, 65, 69, 75, 82, 88])

y = np.array([165, 168, 174, 177, 179, 183])

x = x.reshape(-1,1)

#Scatter Plot of the samples

plt.plot(x, y, 'o')

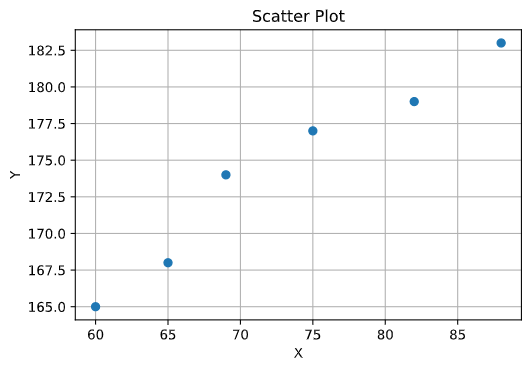
plt.title('Scatter Plot')

plt.xlabel('X')

plt.ylabel('Y')

plt.grid()

plt.show()



#Linear regression and visualization for explanation (1)

linearModel = LinearRegression().fit(x, y)

display(linearModel.coef\_)

display(linearModel.intercept\_)

fitline=linearModel.coef\_\*x+linearModel.intercept\_

meanline = np.mean(y)+0\*x

plt.figure(figsize=(10,5))

plt.subplot(1,2,1)

plt.plot(x,y,'o')

plt.plot(x, fitline, label='Linear Model')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.subplot(1,2,2)

plt.plot(x, y, 'o')

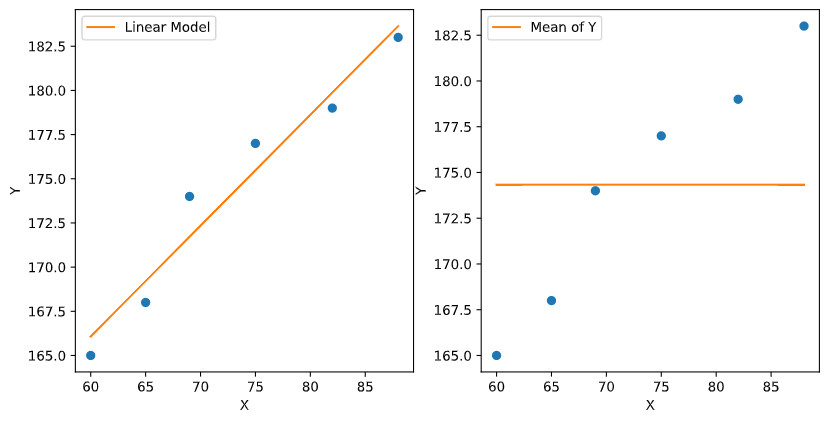
plt.plot(x, meanline, label='Mean of Y')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.show()



Now, going back to the equation of R-Squared, can be depicted as the sum of distances between the line in the second graph (Mean of Y) and the actual values of Y data. Dividing the result by N (number of samples), we can get the variance of the result.

Similarly, can be depicted as the sum of distances between the line in the first graph (Linear Model) and the actual values of Y data. Dividing the result by N, we get the variance of the result.

As the denominator is set as (or the variance of it), we can assume that if the model explains the data well (i.e., if the variation of Y is explained by taking X into account), the value of R-squared will become closer to 1, because the subtracted value on the numerator () will get smaller. Thus as implied from the equation, the R-squared value depicts the extent of variation in Y explained by taking X into account.

For a clear understanding, now consider the following illustrations:

y2 = linearModel.predict(x)

meanline2 = np.mean(y2) + 0\*x

plt.figure(figsize=(10,5))

plt.subplot(1,2,1)

plt.plot(x,y2,'o')

plt.plot(x, fitline, label='Linear Model')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.subplot(1,2,2)

plt.plot(x, y2, 'o')

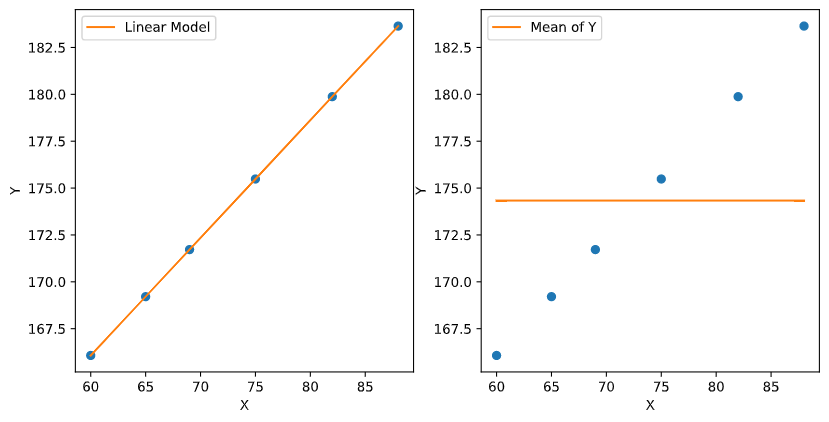
plt.plot(x, meanline2, label='Mean of Y')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.show()



What would be the R-squared value for the linear model generated this time (graph 1)? By now, you will know that the R-squared value for this model would be 1, because the subtraction value () is 0 (because there is no distance between each sample and the fitted linear line). Then, this would mean that the variation of Y is 100% explained by taking X into account.

* 1. P-Value

R-squared values are especially beloved in machine-learning as the evaluation method of the model, as “predictions” become the chief objective for machine learning. If you are an economist or a social scientist, however, statistical significance (and the statistical significance of the calculated R-squared value) is another major evaluation criterion, because R-squared (or even adjusted R-squared) value has some blind spots as well. For example, changes in the hours of sleep you have per day may somehow be highly correlated to the changes in the number of cars produced in California, which would result in high R-squared value. However, this result is most likely coming from random contingency, with no actual statistical meaning. This is why, representatively, statisticians and social scientists use P-value as another criterion for the evaluation of a model.

* + 1. Differences between P-value and Probability

A typical mistaken notion regarding P-value comes from confusing between the concept of Probability and P-value. Let us first resolve this misunderstanding by directly calculating both probability and P-value using a coin-toss example.

If you flip a coin two times, the probability of getting two tails would be: outcome of having two tails / the total number of outcomes = 1/4. Thus the probability of getting two tails out of two flips would be 25%.

P-value (or, two-tailed P-value), on the other hand, is calculated by adding up the probability of obtaining both equally probable and more extreme outcomes to the original probability. In this case, therefore, the P-value would be 1/4 + 1/4 = 1/2 (0.5).

* + 1. Null Hypothesis

It is worth mentioning that P-value always comes with hypothesis. As regards to the example above, the hypothesis would be: “the coin used in this coin-toss, which generated the outcome of two tails in a row, is not as ordinary as the other ordinary coins.” If we want our hypothesis to be right, our goal would be to reject the null hypothesis (the opposite of our hypothesis), which would be that the coin used in this coin-toss is just as ordinary as the other coins. The typical threshold in the area of social science for rejecting the null hypothesis is when p-value is smaller than 0.05 (or 0.1, depending on the cases), and is often denoted with \* sign as: [\*p<0.01, \*\*p<0.05, \*\*\*p<0.01] depending on each threshold. In case of the coin-toss example earlier, null hypothesis cannot be rejected since the calculated value is 0.5, which is much larger than the accepted threshold.

* + 1. Probability Density and P-value

Let us generate a probability density function of a normal distribution for better understanding.

stdRv = norm(loc=0, scale=1)

plt.plot(x, stdRv.pdf(x))

plt.axvline(x=4.7, ymax=0.004, color='r', linewidth=20)

plt.axvline(x=4.35, ymax=0.2, color='g', linewidth=2, linestyle=':')

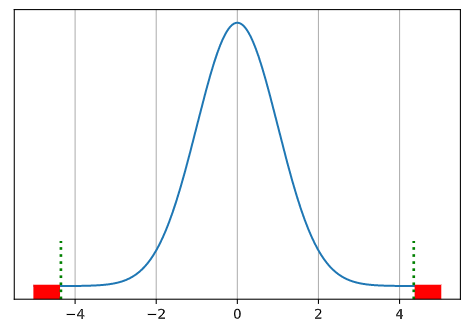
plt.axvline(x=-4.7, ymax=0.004, color='r', linewidth=20)

plt.axvline(x=-4.35, ymax=0.2, color='g', linewidth=2, linestyle=':')

plt.yticks([])

plt.grid()

plt.show()



We have learned previously that the area under the distribution curve as above depicts the probability of getting the corresponding range of X values. Let us suppose that the 99.8% of the area under the curve is between the X values of -4.35 and 4.35. This means that whenever we pick a random float number within the range of total X values, there is a 99.8% chance that the picked number is going to be between the X values of -4.35 and 4.35.

Let us assume that we observed the X value of -4.35. What we want to know is if this X value is coming from the distribution above with average X value of 0, or:

stdRv = norm(loc=0, scale=1)

x = np.linspace(-5, 5, 1000)

x2 = np.linspace(-9.35, 0.65, 1000)

plt.plot(x, stdRv.pdf(x))

plt.axvline(x=4.7, ymax=0.004, color='r', linewidth=20)

plt.axvline(x=4.35, ymax=0.2, color='g', linewidth=2, linestyle=':')

plt.axvline(x=-4.7, ymax=0.004, color='r', linewidth=20)

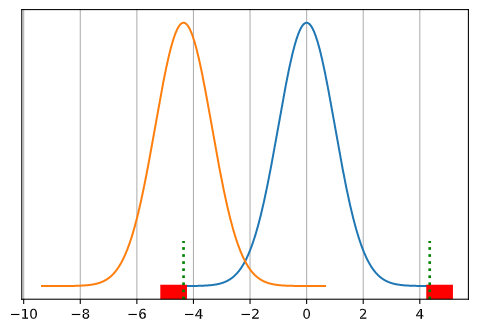
plt.axvline(x=-4.35, ymax=0.2, color='g', linewidth=2, linestyle=':')

plt.plot(x2, stdRv.pdf(x))

plt.yticks([])

plt.grid()

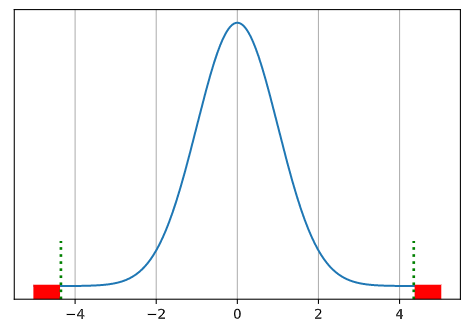
plt.show()



an alternative distribution such as the one above (orange). The null hypothesis would be that the observed X value -4.35 is indeed coming from the original distribution (blue). We know that the area marked as red (less than -4.35 and larger than 4.35) is 0.2% of the total area under the curve. As calculated earlier, P-value is obtained by adding up the possibilities of having equally probable and more extreme outcomes (and thus, the area of both ends colored as red, i.e., X <= -4.35 and X >= 4.35). Thus we get the P-value of 0.002 (0.001 + 0.001), which is considered as statistically significant P-value (\*\*\*p<0.01). This implies that we can reject the null hypothesis and conclude that the observed X value is more likely coming from other forms of distribution such as the orange one.

* + 1. Applications of P-Value

Now that we understand the whole concepts and mechanism of P-value, what would be the example usage of it? Consider the distribution graph we drew earlier:



Now, suppose that the X values indicate the average GDP growth rates of countries at year 2020. Using the same X example we used earlier, let us assume that country A exhibited the growth rate of -4.35 (%). The P-value for this country’s growth rate is less than 0.01. Thus given the distribution of GDP growth rates of all countries, we can statistically state that the country A’s GDP growth rate of -4.35 (%) was highly unusual, compared to other countries, at year 2020 (which then also calls for various other analyses to investigate why country A’s GDP growth rate was unusually low compared to other countries at year 2020).

* + 1. Theoretical Backbone of Inferential Statistics, P-Value, and Significance Levels:

Central Limit Theorem

Regardless of the shapes and types of the population distribution, if the size of sample is sufficiently large and the random variables satisfy the i.i.d. supposition, the distribution of sample ***means*** will be approximately normally distributed. This becomes the theoretical basis for significance tests in inferential statistics. P-value and the level of significance (e.g., 5%, 1%), for instance, are validated under the premise that the distribution of sample means approximately forms normal distribution. Throughout creating a sample simulation and visualization, as follows, it is possible to check if the distribution of sample means approximates normal distribution. Let us suppose a uniform distribution for our sample:

#Demonstration of the Central Limit Theorem

from scipy.stats import uniform

rv = uniform()

rv.rvs(3)

n = 20000

m = 10000

mean = []

for \_ in range(m):

    x = rv.rvs(n)

    mean.append(x.mean())

import seaborn as sns

import matplotlib.pyplot as plt

sns.distplot(mean)

plt.grid()

plt.show()

output:

