**Introduction to Machine Learning**

**And its Python Implementation**

Author: Li Hong Zephan

Version: 1.0

Last Update: 2020/09/01

[1 Introduction to Machine Learning 4](#_Toc50257585)

[1.1 What is NOT Machine Learning 4](#_Toc50257586)

[1.2 What is Machine Learning 9](#_Toc50257587)

[2 Introduction to Gradient Descent 10](#_Toc50257588)

[2.1 What is Gradient Descent 10](#_Toc50257589)

[2.2 Notation 13](#_Toc50257590)

[2.3 Standard Gradient Descent 17](#_Toc50257591)

[2.4 Stochastic Gradient Descent 20](#_Toc50257592)

[2.5 Mathematical Alternative of Gradient Descent 21](#_Toc50257593)

[2.6 Python Implementation of Gradient Descent 31](#_Toc50257594)

[2.7 The influence of the Learning Rate on the Convergence Rate 34](#_Toc50257595)

[2.8 Finding the Parameters using the Normal Equation 37](#_Toc50257596)

[3 Locally Weighted Regression 39](#_Toc50257597)

[3.1 Realistic Training Examples for Human Height and Weight 39](#_Toc50257598)

[3.2 Introduction to Locally Weighted Regression 40](#_Toc50257599)

[3.3 Python Implementation of Locally Weighted Regression 43](#_Toc50257600)

[4 Introduction to Classification Problems 51](#_Toc50257601)

[4.1 Examples of Classification Problems 51](#_Toc50257602)

[4.2 Probability Interpretation in Linear Regression 52](#_Toc50257603)

[4.3 Introduction to Logistic Regression and its Probabilistic Interpretation 55](#_Toc50257604)

[4.4 Python Implementation of Logistic Regression 59](#_Toc50257605)

[5 Practical Review Exercise 64](#_Toc50257606)

[5.1 Gradient Descent Practical Exercise 64](#_Toc50257607)

[5.2 Logistic Regression Practical Exercise 71](#_Toc50257608)

[6 Newton’s Method 90](#_Toc50257609)

[6.1 Another Method for Finding Thetas 90](#_Toc50257610)

[6.2 Python Implementation of Newton’s Method 92](#_Toc50257611)

[6.3 Difference between Gradient Descent and Newton’s Method 95](#_Toc50257612)

[7 The Exponential Family and the Generalized Linear Model 98](#_Toc50257613)

[7.1 Introduction to The Exponential Family 98](#_Toc50257614)

[7.2 Gaussian Distribution in the Exponential Family 99](#_Toc50257615)

[7.3 Bernoulli Distribution in the Exponential Family 99](#_Toc50257616)

[7.4 Introduction to Generalized Linear Models 101](#_Toc50257617)

[7.5 Generalized Linear Model for Gaussian Distribution 101](#_Toc50257618)

[7.6 Generalized Linear Model for Bernoulli Distribution 102](#_Toc50257619)

[8 Softmax Regression 104](#_Toc50257620)

[8.1 Introduction to Multinomial 104](#_Toc50257621)

[8.2 Derivation of Softmax Regression 104](#_Toc50257622)

[8.3 Python Implementation of Softmax Regression 104](#_Toc50257623)

# 1 Introduction to Machine Learning

## 1.1 What is NOT Machine Learning

Instead of defining what is Machine Learning by statements, I would like to help the readers understand what is NOT machine learning by a simple example.

Suppose we built a robot which has a speaker and a camera, and the camera can read a 8 x 8 pixels monitor, each pixel can either be turned on or off (Let’s use “1” to represent ON, and “0” to represent OFF). If we represent these pixels by a Two Dimensional Array, we’ll have something like this:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 |  |  |  | 1 |  |  |
|  | 1 |  | 1 |  | 1 |  |  |
|  |  | 1 |  |  | 1 |  |  |
|  |  |  |  |  |  |  | 1 |
|  |  |  |  |  | 1 |  |  |
| 1 |  | 1 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  | 1 |  |  | 1 |  |  |

Table 1.1 – 8 x 8 pixels monitor

(For better looking, we omitted all zeros in the above table, i.e. empty space “ ” means zero “0” in the array)

In computer programs such as Python or JavaScript, we usually say that the above array has 8 rows (i from 0 to 7) and 8 columns (j from 0 to 7)

Now suppose, we program the robot so that, if the number of “1” in the upper part of the table, is more than or equal to the number of “1” in the lower part of the table, the robot will say “Up” (or say “1”). On the contrary, the robot will say “Down” (or say “0”).

For a computer programmer that does not understand Machine Learning, he can write a simple program, that counts the number of “1” in rows 0 to 3 (then assign this value to, say intUp), and then counts the number of “1” in rows 4 to 7(then assign this value to intDown), and compare the values of intUp and intDown to determine whether the robot should say “Up” or “Down”.

The Python Implementation of this program is:

1. # In this NON-Machine Learning Example, we try to use a simple program
2. # to determine whether the no. of 1 in the upper part of the array
3. # is more than the no. of 1 in the lower part of the array
5. # Define Array
6. aryT = []
7. aryT.append([1,1,0,0,0,1,0,0])
8. aryT.append([0,1,0,1,0,1,0,0])
9. aryT.append([0,0,1,0,0,1,0,0])
10. aryT.append([0,0,0,0,0,0,0,1])
11. aryT.append([0,0,0,0,0,1,0,0])
12. aryT.append([1,0,1,0,0,0,0,0])
13. aryT.append([0,0,0,0,0,0,0,0])
14. aryT.append([0,0,1,0,0,1,0,0])
16. # No. of Rows and Columns
17. intM = len(aryT)
18. intN = len(aryT[0])
20. # Init No. of Up and Down
21. intUp = 0
22. intDown = 0
24. # Count No. of Up and Down in Array
25. **for** i **in** range(intM):
26. **for** j **in** range(intN):
27. **if** (i <= 3):
28. **if** (aryT[i][j] == 1):
29. intUp += 1
30. **else**:
31. **if** (aryT[i][j] == 1):
32. intDown += 1

35. # Print result
36. **print**('intUp: ' + str(intUp))
37. **print**('intDown: ' + str(intDown))
39. **if** (intUp >= intDown):
40. **print**('The result is: 1')
41. **else**:
42. **print**('The result is: 0')

Program 1.1 - NON Machine Learning classification program ML0101.py

This is the output of the program:

1. intUp: 9
2. intDown: 5
3. The result **is**: 1

Output 1.1

As expected, the numbers of “1” in the upper part and lower part of the array are printed and the result is “1”, i.e. the array has more “1”s in the upper part of the array.

Now, if we want the robot to recognize the digits from “0” to “9”, for examples:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 1 | 1 |  |  |  |
|  |  | 1 |  | 1 |  |  |  |
|  |  |  |  | 1 |  |  |  |
|  |  |  |  | 1 |  |  |  |
|  |  |  |  | 1 |  |  |  |
|  |  |  |  | 1 |  |  |  |
|  |  |  |  | 1 |  |  |  |
|  |  | 1 | 1 | 1 | 1 | 1 |  |

Or other “similar figures” to be recognized as “1” and:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 1 | 1 | 1 |  |  |
|  | 1 |  |  |  |  | 1 |  |
|  |  |  |  |  |  | 1 |  |
|  |  |  |  |  |  | 1 |  |
|  |  | 1 | 1 | 1 | 1 |  |  |
|  | 1 |  |  |  |  |  |  |
|  | 1 |  |  |  |  |  |  |
|  | 1 | 1 | 1 | 1 | 1 | 1 |  |

Or other “similar figures” to be recognized as “2”, and so on for the “3” to “9”.

How can we modify the Python program so that the robot can “act like a human being”, and tell us what is the digit shown in 8x8 pixels monitor, if there are so many combinations to represent each digit?

In this scenario, we cannot simply explicitly tell the program how each digit is represented, we need to use some machine learning algorithm(s) for this program, and we’ll explain those algorithms in our later chapters.

## 1.2 What is Machine Learning

So what is Machine Learning? According to Wikipedia, “Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so.”

In our next chapter, we’ll learn our first Machine Learning Algorithm – Gradient Descent.

# 2 Introduction to Gradient Descent

## 2.1 What is Gradient Descent

Suppose we have a data set of weight against height of a particular group of people:

|  |  |
| --- | --- |
| Height | Weight |
| 20 | 1 |
| 40 | 4 |
| 60 | 20 |
| 120 | 30 |
| 150 | 38 |
| 162 | 50 |
| 171 | 60 |

Table 2.1.1

If we try to plot a straight line with Weight against Height using Excel, we may have the following graph:

Figure 2.1.1

The equation y = 0.35x – 6.77 is found by Excel, we are not going to investigate how Excel get this equation, but instead, we are going to use the Machine Learning Algorithm of Gradient Descent to get this equation in our later sections of this Chapter.

First of all, we try to elaborate more on this equation, so that you can understand the later sections more easily.

The above equation can be expressed as:

y = ax + b, or y = b + ax, or simply

y = ϴ0 + ϴ1x (By setting b = ϴ0 and a = ϴ1)

In realistic situations, the weight of a person, is not simply related to the height, it may also related to other “features” of that person, e.g. age and living countries......, thus, to generalize the equation, we have

y = ϴ0 + ϴ1x1 + ϴ2x2 + ϴ3x3 + ...... + ϴnxn Eq. 2.1.1 (if y is related to n features such as weight, age, sex, countries......)

But how do we know if y is directly proportional to x1, or it is actually directly proportional to the square of x1?

What about the case that y = ax2 + bx + c, can we still use the above generic equation of y?

The answer is YES, we can simply consider x2 as another feature of y, by setting a = ϴ2, b = ϴ1, c = ϴ0, x = x1(i.e. 1st feature of y), x2 = x2(i.e. 2nd feature of y), we still have:

y = ϴ0 + ϴ1x1 + ϴ2x2

which is the same form as Eq. 2.1.1

so, by setting x0 = 1, we can define:

i.e., we need to find all ϴs! To find these ϴs by Gradient Descent, we’ll first assume those ϴs to some initial values (say, all zeros or all ones), and use the method of Gradient Descent to alter each ϴ a little bit until we get the final values with least errors for all data provided by human being.

Since we are “guessing” the value of ϴs, we call y the “hypothesis” of the learning algorithm, and from now on, we should rewrite the above y as hϴ(x), and y will become another meaning in the following sections.

## 2.2 Notation

m = # training examples (e.g. in Table 2.1.1, there are 7 training examples)

x = “input” variables / features (e.g. the height in Table 2.1.1)

y = “output” variables / “target” variables (Figures inputted by human, e.g. the weight in Table 2.1.1)

(x, y) = a training example. (e,g, a row in Table 2.1.1)

The jth training example = (x(j), y(j)) (i.e. the jth row in Table 2.1.1)

Hypothesis hϴ(x), or simply h(x) = ϴ0 + ϴ1 x1 + ϴ2 x2 + …. + ϴnxn (for an x with n features)

We shall explain the meaning of Hypothesis very soon.

And Defining x0 = 1, =>

Now, let ϴ and x be matrices with 1 column and n row (i.e. consider ϴ and x be vectors represented by matrices), we have:

ϴ = and x =

ϴT =

Recall some matrix operation techniques, suppose a matrix A with i rows and j columns, another matrix B with j rows and k columns, and the multiplication of the two matrices C = AB, will be a matrix of i rows and k columns.

Now ϴT is just matrix A, with i = 1 and j = n, x is just matrix B with j = n and k = 1, so the multiplication of ϴTx is a matrix with 1 row and 1 column, and the value of the single element of this matrix product is:

ϴ0 x0 + ϴ1 x1 + ϴ2 x2 + …. + ϴnxn

Thus we have:

We call the ϴs “parameters”.

So what is a Hypothesis? (i.e. What is hϴ(x)?) A hypothesis is an equation that we “guess”, how y is related to x, in the above example, we “guess” that y is directly proportional to the sum of the x features, with a Real Number Parameter ϴi before each xi . In later chapters, we’ll also handle hypothesis with higher order of xi .

Then, for m training examples, we try to find all ϴs, in order to minimize:

The is just for the simplification of mathematical operations later on.

Now, let:

By putting in different set of ϴs, we try to minimize J(ϴ), in other words, we want to find some optimistic ϴs, such that the value hϴ(x) that we guess, can be as close as possible to the actual value y, for all m training examples. Here we call J(ϴ) the Loss Function.

So how can we find those ϴs? We’ll introduce the Gradient Descent!

## 2.3 Standard Gradient Descent

To start, we randomly pick a value for ϴi, and iterates this ϴi with the following equation:

The := denote a computer operation that a new value of ϴi will be replaced by some calculations based on the old value of the previous ϴi. And the alpha “α” is called the learning rate, which is often set by human beings based on experiences on different machine learning algorithm, we’ll talk about how to set this learning rate later.

So, this computer operation means that, we try to find an optimal ϴi such that the J(ϴ) is minimum with respect to ϴi, in other words, we try to minimize the differences between the “calculated value by the hypothesis h(x)” and the “actual value y” provided by the trainer (i.e. figures provided by the human being), so we have, for 1 training example (x,y):

=

= (By Chain Rule)

since the only term inside hϴ(x) related to ϴi is ϴi xi, the derivative of other terms are zero, we have:

=

Thus,

ϴi := ϴi – … (Eq. 2.5) for 1 set of training example (x,y) only

(Note: in the above equation, the “i” stands for the ith feature of x)

for m sets of training examples, we have

(Note: Again, the “i” stands for the ith feature of x, and “j” stands for a particular training example)

The above algorithm to find each ϴi by iteration is called “**Batch Gradient Descent**” or just “**Standard Gradient Descent**”, it is obvious that, for EACH iteration, you need to sum all the (h(x) – y). xi in order to get a “better” ϴi to minimize J(ϴ). Consider if there are so many training examples, say, 10000 training examples, it’s very time consuming for the “Batch Gradient Descent” to find all ϴs such that J(ϴ) is minimized.

To handle a large number of training examples, we would like to introduce the “**Stochastic Gradient Descent**”.

## 2.4 Stochastic Gradient Descent

Unlike “Batch Gradient Descent”, “Stochastic Gradient Descent” randomly pick a few subset of the training examples, to approximately guesses the next ϴi, and repeat the process until the ϴi converges (“converges” means the ϴi does not change much, or the J(ϴ) does not change much, while the iterations continue).

Say, for example, if we only pick a random of 100 examples from a total of 100,000 training examples for each iteration, the total time consumed to optimize ϴs may be a few hundred times faster than the “Batch Gradient Descent”.

It is important to note that, picking fewer random examples in each iteration means faster iterations speed, but lower convergence rate. (i.e. Picking too little examples, the calculations may not converge, which means that you can’t get the minimum J(ϴ) after changing ϴs)

The above algorithms use iterations to obtain ϴs in order to minimize J(ϴ). In some particular cases, e.g. least square equation such as Eq. 2.3, we can obtain the ϴs by mathematical calculations instead of computer iterations.

## 2.5 Mathematical Alternative of Gradient Descent

To do so, we’ll need to introduce a few more notations:

∇ϴ J is a vector represented by a “n+1 rows, 1 column, Real Number Matrix”, which means the derivative of J with respect to ϴ, i.e.:

∇ϴ J= ϵ ℝ … (Eq. 2.7)

The notation “ϵ ℝ”, in case of font problem you can’t see it, it means “belong to Real Number”

Then, we rewrite ϴ as:

ϴ := ϴ - α ∇ϴ J … (Eq. 2.8)

We’ll now introduce more notations before we continue:

suppose A ϵ ℝ m×n, in other words, A is a matrix of real numbers with m rows and n columns, and f(A) is a function of A, then:

∇A f(A) = … (Eq. 2.9)

which is also a matrix of real numbers with m rows and n columns.

Now suppose B is a Square Matrix, i.e. B ϵ ℝ n×n

The trace of a square matrix is defined as:

Let’s recall some properties of trace of matrix, assume A, B and C are all square matrices with n rows and columns:

tr(AB) = tr(BA) … (Eq. 2.11)

We’ll try to verify Eq. 2.11 with a simple example by setting n = 2:

A = … (Eq. 2.12)

B = … (Eq. 2.13)

AB = … (Eq. 2.14)

tr(AB) = a11b11+a12b21+a21b12+a22b22 … (Eq. 2.15)

BA = … (Eq. 2.16)

tr(BA) = a11b11+a21b12+a12b21+a22b22 … (Eq. 2.17)

by reordering the 2nd and the 3rd terms, we have

tr(BA) = a11b11+a12b21+a21b12+a22b22 = tr(AB)

By similar examples, we can also verify that:

tr(ABC) = tr(CAB) = tr(BCA) … (Eq. 2.18)

Now, suppose, if we can express f(A) as tr(AB), i.e.:

f(A) = tr(AB) … (Eq. 2.19)

It turns out that:

∇A f(A) = BT … (Eq. 2.20)

Let’s verify Eq. 2.20 by using a particular example of A and B stated in Eq. 2.12 and Eq. 2.13:

In Eq. 2.9, and replace f(A) with tr(AB) in Eq. 2.15, we have:

∇A f(A) =

= = BT

And it is also easy to verify that:

tr(A) = tr(AT) … (Eq. 2.21)

since the Transpose of A does not change the diagonal elements of square matrix A.

Let us introduce 2 more equations here without the actual proving:

(1), if a ϵ ℝ, we have:

tr(a) = a … (Eq. 2.22)

(2),

∇A tr(ABATC) = CAB + CTABT … (Eq. 2.23)

With the help of the above equations: Eq. 2.11, Eq. 2.18, Eq. 2.20, Eq. 2.21, Eq. 2.22 and Eq. 2.23, we are going to learn how we can find ϴs in order to minimize J(ϴ) without using computer iterations.

Let X be a matrix which contains all inputs of each training example (m training examples with n features)

X = … (Eq. 2.24)

=

(Note: )

Then Xϴ will be the hypothesis for each training example

Xϴ = ϴ

=

=

= … (Eq. 2.25)

Define y as a matrix which contains outputs of m training examples

= … (Eq. 2.26)

Then Xϴ - y will be a matrix which contains the difference between hypothesis and actual outputs of m training examples

Xϴ - y = … (Eq. 2.27)

=

=

=

Multiply by one half, we have:

To minimize J, we have to set the derivative of J() closest to 0, which means the slope of J is closest to 0.

= … (Eq. 2.30 Step 1)

= … (Eq. 2.30 Step 2)

= … (Eq. 2.30 Step 3)

For Eq. 2.30 Step 1, expand the :

Note that X is a m x n matrix while is a n x 1 matrix, when is transposed, is a n x m matrix while is a 1 x n matrix.

Therefore, but

For Eq. 2.30 Step 2, we know that is just a real number (See Eq. 2.28), so we can add a trace to it, where a trace of a real number is just itself (See Eq. 2.22).

For Eq. 2.30 Step 3, ignore the one half and divide it into three parts:

First part:

For the first part, apply Eq. 18 to and move to the front, then we have

Second part: 1st

For the second part, we know that is a real number, where and it is a 1 x m matrix, while y is a m x 1 matrix. The transpose of a real number is just itself. Therefore, we can transpose it, then we have .

Third part: 2nd

For the third part, for later calculation, we do not have to change it.

For the , since we cannot differentiate it with respect to , it will be equal to 0 and we can just ignore it.

Add an identity matrix *I*, which will be ignored, to , then apply Eq. 2.23 to it, we have:

= … (Eq. 2.31)

Where refers to A, I refers to B, refers to , and refers to C.

Note: the identity matrix I is a m x m square matrix. Here is an example of identity matrix with 3 rows and columns:

I3 =

By applying Eq. 2.19, we have:

= … (Eq. 2.32)

Where refers to B, refers to A, and refers to

Apply Eq. 2.31 and Eq. 2.32 to Eq. 2.30, we have:

=

=

Then set to 0, we have:

… (Eq. 2.33 – Normal Equation)

Note: Eq. 2.33 is also called the Normal Equation.

By using the normal equation, we can solve for without the use of the iterative Gradient Descent algorithm.

## 2.6 Python Implementation of Gradient Descent

The following is a simple python program to illustrate how we can find all the ϴs in order to minimize the J(ϴ) as described in our last chapter. We shall explain this program briefly after the source codes.

1. # In this Machine Learning Example, we try to use Gradient Descent
2. # to estimate the weight of a person y, with given height x of a person
4. # Define No. of Features of x
5. intN = 1
7. # Define Training Examples
8. aryT = []
9. aryT.append({'x': [20], 'y': 2})
10. aryT.append({'x': [40], 'y': 4})
11. aryT.append({'x': [60], 'y': 6})
12. aryT.append({'x': [120], 'y': 12})
13. aryT.append({'x': [160], 'y': 16})
14. aryT.append({'x': [162], 'y': 16.2})
15. aryT.append({'x': [171], 'y': 17.1})
17. # No. of Training Examples
18. intM = len(aryT)
20. # Learning Rate
21. intAlpha = 0.00001
23. # Init Array Parameters
24. aryTheta = []
25. **for** i **in** range(intN+1):
26. **if** (i == 0):
27. # Init. Theta0 with 0
28. aryTheta.append(0)
29. **else**:
30. # Init. Theta1 with 0
31. aryTheta.append(0)

34. # Hypothesis H, t stands for theta
35. # H(t) = t0 x0 + t1 x1
37. intMaxTrainTimes = 100

40. # Init. Temp vars.
41. intSum = 0
42. intTemp = 0

45. # Iterates for intMaxTrainTimes
46. **for** t **in** range(intMaxTrainTimes):
47. # Iterate for each feature
48. **for** i **in** range(0, intN+1):
49. # Init Summation
50. intSum = 0
52. **for** j **in** range(intM):
53. intTemp = (aryTheta[0] + aryTheta[1] \* aryT[j]['x'][0]) - aryT[j]['y']
54. **if** i == 1:
55. intTemp = intTemp \* aryT[j]['x'][0]
57. **print**('i: 0 | t: ' + str(t) + ' | j: ' + str(j) + ' | intTemp: ' + str(intTemp))
58. intSum += intTemp
59. #End for
61. **print**('intSum: ' + str(intSum))
63. # Calculate New Theta(0)
64. aryTheta[i] = aryTheta[i] - intAlpha \* intSum
66. #print('i: 0 | aryTheta[0]: ' + str(aryTheta[0]))
67. **print**('aryTheta[0]: ' + str(aryTheta[0]))
68. **print**('aryTheta[1]: ' + str(aryTheta[1]))


72. # Print all Theta(s)
73. **print**('aryTheta[0]: ' + str(aryTheta[0]))
74. **print**('aryTheta[1]: ' + str(aryTheta[1]))

Program ML0201.py

Explanations:

1. The intN is the no. of features in an input x of a training example (x, y).
2. The intM is the total no. of training examples.
3. The array aryT contains all the training examples, aryT[0] means the first training example.
4. The “x” of aryT contains all the feature inputs of a training example, e.g. if intN = 2, aryT[3][’x’][1] means the x24 , i.e. The value of the second feature of x, of the 4th training example.
5. The “y” of aryT contains the actual value provided by human being, e.g. aryT[2][’y’] means the y3, i.e. The actual value of the 3rd training example.
6. intAlpha is the Learning Rate.
7. aryTheta is an array that stores all the values of ϴs.
8. intMaxTrainTimes is the no. of times that we want to iterates in order to get the final values of aryTheta.
9. The main objective of this python program is to simulate Eq. 6, so as to find all ϴs.

## 2.7 The influence of the Learning Rate on the Convergence Rate

It is obvious that, we can use the figures provided by the training examples to plot a graph of a straight line with a slope of 0.1 and an intercept on (0,0)：

Figure 2.7.1 – Training Examples

Note: In our first example of machine learning algorithm, we want to use the simplest training examples, so that it is clear that in the above figure, the slope of the line is 0.1 and the intercept is (0,0)

After running this program using python3, we can find that:

aryTheta[0] = 0.00072392

aryTheta[1] = 0.099995

which is very close to the calculated values of 0.00 and 0.10 for these ϴs.

So how about we change the value of the Learning Rate? Will the calculations still converge with larger learning rate? Here is the table that contains the figures of the ϴs for different learning rate:

|  |  |  |
| --- | --- | --- |
| Learning Rate | ϴ0 | ϴ1 |
| 0.000010 | 0.00072392 | 0.099995 |
| 0.000011 | 0.00072381 | 0.099995 |
| 0.000012 | 0.00072369 | 0.099995 |
| 0.000013 | 0.00072357 | 0.099995 |
| 0.000014 | 0.00072345 | 0.099995 |
| 0.000015 | 0.00072333 | 0.099995 |
| 0.000016 | 0.00072322 | 0.099995 |
| 0.000017 | 0.00072310 | 0.099995 |
| 0.000018 | 0.00072298 | 0.099995 |
| 0.000019 | 0.00072268 | 0.099969 |
| 0.000020 | 0.00541728 | -0.746647 |
| 0.000021 | -76.78181 | -10587.30267 |

Table 2.7.1

So, it is quite obvious that, the calculations start to diverge, when the learning rate is >= 0.000019 for these particular training examples.

i.e. For the calculations to converge, we need to set the learning rate <= 0.000018 for these particular hypothesis and training examples.

To simplify the selection of the Learning Rate, we can choose an initial learning rate of 0.001 divided by (10 to the power of the order of the maximum x + 1), for example, in our training set, the maximum value of x is 171, so we divide 0.001 by 1000 and set the learning rate to 0.000001. And since the maximum learning rate allowed is 0.000018, by using 0.000001(18 times smaller than the maximum learning rate allowed), we may get the convergence fast and safe enough.

In our later chapters, we’ll try to understand how we can find the optimal value(s) of the Learning Rate using other algorithms.

## 2.8 Finding the Parameters using the Normal Equation

Recall the Normal Equation 2.33 in our previous sections:

And using the training examples provided by the program ml0201.py, we have:

y =

X =

=>

XT =

Substituting these values into Eq. 2.33 we have:

=

=

=

=

=

= 73.3 … (Eq. 2.7.1)

= 10108.5 …(Eq. 2.7.2)

By solving the above 2 equations, we have:

Which are exactly the same answers we found by computer operations.

# 3 Locally Weighted Regression

## 3.1 Realistic Training Examples for Human Height and Weight

In figure 2.6.1, we had a simplified relationship for Human Height and Weight, which is a perfect straight line. Unfortunately, in realistic world, the actual line might not be suitable to be fitted by a straight line, for example:

|  |  |
| --- | --- |
| **Height** | **Weight** |
| 10 | 0.5 |
| 20 | 1.5 |
| 40 | 3.5 |
| 60 | 8 |
| 80 | 12 |
| 100 | 20 |
| 120 | 30 |
| 140 | 40 |
| 160 | 55 |
| 170 | 70 |
| 180 | 76 |

Table 3.1.1 Figure 3.1.1

However, if we want to use the Gradient Descent learned in our last chapter, we may need to fix the line with a hypothesis h(ϴ) of higher order of x, in order to have a more accurate approximation. For example:

Figure 3.1.2

However, as a human being, how do we know which order of x should be used to fit the line? Sometimes there are n features in the training examples, so that we are actually unable to plot a graph with n+1 dimensions, and then guess the dimensions of each feature. In the above case, Should we try x2, and x3, and ……?

Thus we would like to introduce another algorithm, the “Locally Weighted Regression”.

## 3.2 Introduction to Locally Weighted Regression

Suppose we have m training examples (x(j),y(j)), with j from 1 to m, after we trained, we have a target xt, and would like to use this algorithm to find the target yt. The principle of this algorithm is: if the training examples are more close to this target spot, we give them a more heavily weight, for those training examples that are far away from this target spot, we give it a lower weight, the result of doing so is similar to we plot a straight line on the target spot and calculate the slope of this line, and use this line to estimate the yt :

Figure 3.2.1

Recall Eq. 2.3 of our Loss Function in Gradient Descent:

And by Eq. 2.2, hϴ(x(j)) is actually ϴTx(j), so we rewrite Eq. 2.3 as:

Now, let us define the Loss Function for the Locally Weighted Regression as follow:

In which w(j) is called the weight of that particular training example (x(j), y(j)). Recall that we want a bigger w(j) when x(j) is close to our target xt, so how should we define w(j)? We can choose ANY “Bell Shaped Function” for w(j), in our example, we would like to choose:

So if x(j) is close to our target xt, w(j) equals (1 divided by a very small number), i.e. w(j) is very large

On the contrary, if x(j) is very far away from xt, w(j) equals (1 divided by a very large number), i.e. w(j) is very small.

Thus it fulfills our requirement of w(j).

However, in order to fine tone w(j), we would like to modify its definition slightly by defining:

In which τ (the Greek symbol “tau”) is called the “bandwidth”, if τ is small, we’ll have a more narrow function of w(j). If τ is large, w(j) will not be so small when x(j) is far away from xt.

## 3.3 Python Implementation of Locally Weighted Regression

Before we do this, we’ll first evaluate yt by using our Gradient Descent python program, and later, we’ll compare the value with the one found by Locally Weighted Regression. To do this, we’ll modify our previous program as:

1. # In this Machine Learning Example, we try to use Gradient Descent
2. # to estimate the weight of a person y, with given height x of a person
4. # Define No. of Features of x
5. intN = 1
7. # Define Training Examples
8. aryT = []
9. aryT.append({'x': [10], 'y': 0.5})
10. aryT.append({'x': [20], 'y': 1.5})
11. aryT.append({'x': [40], 'y': 3.5})
12. aryT.append({'x': [60], 'y': 8})
13. aryT.append({'x': [80], 'y': 12})
14. aryT.append({'x': [100], 'y': 20})
15. aryT.append({'x': [120], 'y': 30})
16. aryT.append({'x': [140], 'y': 40})
17. aryT.append({'x': [160], 'y': 55})
18. aryT.append({'x': [170], 'y': 70})
19. aryT.append({'x': [180], 'y': 76})
21. # No. of Training Examples
22. intM = len(aryT)
24. # Learning Rate
25. intAlpha = 0.00001
27. # Init Array Parameters
28. aryTheta = []
29. **for** i **in** range(intN+1):
30. **if** (i == 0):
31. # Init. Theta0 with 0
32. aryTheta.append(0)
33. **else**:
34. # Init. Theta1 with 0
35. aryTheta.append(0)

38. # Hypothesis H, t stands for theta
39. # H(t) = t0 x0 + t1 x1
41. intMaxTrainTimes = 50000

44. # Init. Temp vars.
45. intSum = 0
46. intTemp = 0

49. # Iterates for intMaxTrainTimes
50. **for** t **in** range(intMaxTrainTimes):
51. # Iterate for each feature
52. **for** i **in** range(0, intN+1):
53. # Init Summation
54. intSum = 0
56. **for** j **in** range(intM):
57. intTemp = (aryTheta[0] + aryTheta[1] \* aryT[j]['x'][0]) - aryT[j]['y']
58. **if** i == 1:
59. intTemp = intTemp \* aryT[j]['x'][0]
61. intSum += intTemp
62. #End for
64. # Calculate New Theta(0)
65. aryTheta[i] = aryTheta[i] - intAlpha \* intSum
67. # Print all Theta(s)
68. **print**('aryTheta[0]: ' + str(aryTheta[0]))
69. **print**('aryTheta[1]: ' + str(aryTheta[1]))

Program ml0301.py

So we just changed the training examples, and enlarge the no. of iterations such that we’ll have a more accurate estimation of ϴs.

After running this program, we have the following output:

aryTheta[0]: -10.656156741040318

aryTheta[1]: 0.41051524883408785

which is quite close to the values calculated by the Excel Graph Generator:

y = 0.41x – 10.39

So we have, by our Python Gradient Descent program ML0301.py :

y = 0.411x – 10.66 Eq. 3.5

From Eq. 3.4, it is known that w(j) is a constant for each x(j), so we’ll be using similar equation as Eq. 2.6 to find our ϴs.

And the following is our Python program for the Locally Weighted Regression:

1. # In this Machine Learning Example, we try to use Locally Weighted Regression
2. # to estimate the weight of a person y, with given height x of a person
4. **import** math
6. # Define No. of Features of x
7. intN = 1
9. # Define Training Examples
10. aryT = []
11. aryT.append({'x': [10], 'y': 0.5})
12. aryT.append({'x': [20], 'y': 1.5})
13. aryT.append({'x': [40], 'y': 3.5})
14. aryT.append({'x': [60], 'y': 8})
15. aryT.append({'x': [80], 'y': 12})
16. aryT.append({'x': [100], 'y': 20})
17. aryT.append({'x': [120], 'y': 30})
18. aryT.append({'x': [140], 'y': 40})
19. aryT.append({'x': [160], 'y': 55})
20. aryT.append({'x': [170], 'y': 70})
21. aryT.append({'x': [180], 'y': 76})
23. # No. of Training Examples
24. intM = len(aryT)
26. # Learning Rate
27. intAlpha = 0.00001
29. # Init Array Parameters
30. aryTheta = []
31. **for** i **in** range(intN+1):
32. **if** (i == 0):
33. # Init. Theta0 with 0
34. aryTheta.append(0)
35. **else**:
36. # Init. Theta1 with 0
37. aryTheta.append(0)

40. # Hypothesis H, t stands for theta
41. # H(t) = t0 x0 + t1 x1
43. intMaxTrainTimes = 10000

46. # Init. Temp vars.
47. intSum = 0
48. intTemp = 0

51. # Define bandwidth Tau
52. intTau = 5
54. # Set Target X
55. intXTarget = 100
57. # Calculate w(j) for each x(j)
58. aryOmega = []
59. **for** j **in** range(intM):
60. intTemp = math.exp((-1 \* math.pow(aryT[j]['x'][0] - intXTarget, 2)) / (2 \* math.pow(intTau, 2)))
61. aryOmega.append(intTemp)
62. #print(intTemp)
64. # Iterates for intMaxTrainTimes
65. **for** t **in** range(intMaxTrainTimes):
66. # Iterate for each feature
67. **for** i **in** range(0, intN+1):
68. # Init Summation
69. intSum = 0
71. **for** j **in** range(intM):
72. intTemp = ((aryTheta[0] + aryTheta[1] \* aryT[j]['x'][0]) - aryT[j]['y'])
73. **if** i > 0:
74. intTemp = intTemp \* aryT[j]['x'][i-1]
76. # Apply weight
77. intTemp = intTemp \* aryOmega[j]
79. intSum += intTemp
80. #End for
82. # Calculate New Theta(0)
83. aryTheta[i] = aryTheta[i] - intAlpha \* intSum
85. # Print all Theta(s)
86. **print**('aryTheta[0]: ' + str(aryTheta[0]))
87. **print**('aryTheta[1]: ' + str(aryTheta[1]))
89. # Calculate Target Y
90. intY = aryTheta[0] + intXTarget \* aryTheta[1]
92. # Print Target Y:
93. **print**('Target Y: ' + str(intY))

Program: ML0302.py

We may now change the intXTarget (i.e. xt) in the program and print out the corresponding value of Target Y, i.e. yt.

The following is a table comparing the result y for different value of x, using both the Gradient Descent and Locally Weighted Regression learning algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
| **Height** | **Actual Weight** | **Weight predicted by “Gradient Descent”** | **Weight predicted by “Locally Weighted Regression”** |
| 10 | 0.5 | -6.55 | 0.588 |
| 20 | 1.5 | -2.44 | 1.484 |
| 40 | 3.5 | 5.78 | 3.501 |
| 60 | 8 | 14 | 8.000 |
| 80 | 12 | 22.22 | 12.002 |
| 100 | 20 | 30.44 | 20.001 |
| 120 | 30 | 38.66 | 30.001 |
| 140 | 40 | 46.88 | 40.002 |
| 160 | 55 | 55.1 | 56.444 |
| 170 | 70 | 59.21 | 69.122 |
| 180 | 76 | 63.32 | 75.794 |

Table 3.3.1

It is obvious that Locally Weighted Regression is much more accurate than the Gradient Descent(using x to the power of 1 only) in this particular example. As the Height / Weight relationship in our example is not suitable to use straight line to represent.

It should be noted that, there is no “pre-training” for the Locally Weighted Regression, since w(j) is related to the chosen xt, we must iterate again and again by using the whole training set for each xt, so this learning algorithm can be regarded as an expensive algorithm if the number of training example is huge.

# 4 Introduction to Classification Problems

## 4.1 Examples of Classification Problems

In the previous chapters, we have talked about using Gradient Descent Algorithm to find out hypothesis which best fits the dataset given. However, the dataset given, or the hypothesis, are both continuous data.

For Classification Problems, we are handling discrete data, an example of that is, the answer is either 0 or 1, true or false, yes or no, etc.

In the reality, there are many things which are related Classification problems. For example, to determine if a student pass or failed the exam, the answer must either be “Yes” or “No”. Therefore, knowing how to use learning algorithms to solve classification problems is important in machine learning.

## 4.2 Probability Interpretation in Linear Regression

In this section, we are going to talk about probability interpretation in linear regression first, as it is related to the concept of logistic regression. In the next section, we will start talking about logistic regression.

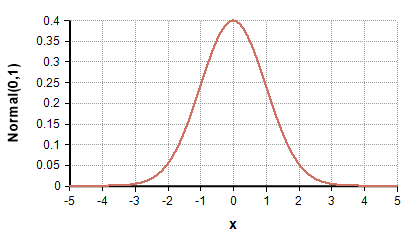
Assume , where (j) means jth training example, and means error. The error might be some features theta which we have not recorded, or something cannot be predicted like mood of a person, accident, etc.

Here we assume that the error is normally distributed with mean 0 and variance , which means that the standard deviation is .

The equation of the normal distribution (Gaussian distribution) looks like this:

Note: is the mean

And the graph of the normal distribution looks like a bell-shaped curve:



(From Google Image)

Then, the probability of will be equal to:

Assume the probability of given in and parameter is:

=

Then, will be normally distributed with mean , which is the hypothesis we made, and variance :

Before the next calculation, we have to assume that all errors are IID, which means they are Independently and Identically Distributed.

The probability of y will be the product from j=1 to m of probability of :

We will also let the likelihood of theta to be the probability of y:

The meaning of likelihood and probability are actually very similar, but we would be more likely to say the likelihood of a parameter, for example, the likelihood of theta. And we would say the probability of a dataset, for example, the probability of y.

Then, we have to maximize the likelihood of theta, which means to maximize the probability of y getting close to the hypothesis.

To maximize likelihood, we change to maximize . And first, we let lower case L to be the log of :

So, to maximize the likelihood, what we need to do is to minimize:

And this is the probability interpretation of linear regression.

## 4.3 Introduction to Logistic Regression and its Probabilistic Interpretation

We know that the output of logistic regression is either 0 or 1. To do this, we will choose a suitable function to limit the range from 0 to 1, and this is the logistic function:

The graph of the logistic function looks like this:

图片包含 照片, 看着, 黑暗, 房间

描述已自动生成

(From Google Image)

Then, we can change our hypothesis with the use of the logistic function:

Now, we are going to explain logistic regression with probabilistic interpretation.

Assume that the probability of y =1 is:

Then, the probability of y = 0 will be:

We can write the above 2 equations together:

If y = 1, then will be eliminated.

If y = 0, then will be eliminated.

The formula is written in the form of the Bernoulli distribution, which it outputs the probability of y getting 0 or 1 only. It is widely used in logistic regression for classifying yes or no, true or false, 0 or 1 questions.

Let the likelihood of theta again:

To find , we have to use gradient descent:

Note: this time we use the “+” operator instead of the “-” operator as we are maximizing the likelihood now, but we are minimizing the error before.

We will start finding each :

Take out in the summation and simplify it:

Take partial derivative to the simplified formula:

For the partial derivative to all training examples:

Lastly, we got the gradient descent equation for finding all thetas:

By using the above equation, we can find all parameters theta which can maximize the likelihood of theta, or the probability of y.

## 4.4 Python Implementation of Logistic Regression

On an 8 x 8 square graph, there will be some points on it. If the number of upper points (row 5-8) is larger than the number of lower points (1-4), then the answer will be 1. On the contrary, the answer will be 0.

The mission of the program is to give out a correct answer without counting number of upper points and lower points. It has to gradient descent to find out all parameters theta to determine if the answer is 1 or 0 instead.

Below is a simple python program to illustrate how we can find all the ϴs in order to maximize the likelihood of theta. We shall explain this program briefly after the source codes.

1. # In this Machine Learning Example, we try to use Gradient Descent
2. # to predict if there are more upper or lower points
4. import math
6. # Define No. of Features of x
7. intN = 64
9. # Define Training Examples
10. aryT = []
11. aryT.append({'x': [
12. 1, 1, 1, 1, 1, 1, 1, 1,
13. 1, 1, 1, 1, 1, 1, 1, 1,
14. 1, 1, 1, 1, 1, 1, 1, 1,
15. 1, 1, 1, 1, 1, 1, 1, 1,
16. 0, 0, 0, 0, 0, 0, 0, 0,
17. 0, 0, 0, 0, 0, 0, 0, 0,
18. 0, 0, 0, 0, 0, 0, 0, 0,
19. 0, 0, 0, 0, 0, 0, 0, 0,
20. ], 'y': 1})
21. aryT.append({'x': [
22. 0, 0, 0, 0, 0, 0, 0, 0,
23. 0, 0, 0, 0, 0, 0, 0, 0,
24. 0, 0, 0, 0, 0, 0, 0, 0,
25. 0, 0, 0, 0, 0, 0, 0, 0,
26. 1, 1, 1, 1, 1, 1, 1, 1,
27. 1, 1, 1, 1, 1, 1, 1, 1,
28. 1, 1, 1, 1, 1, 1, 1, 1,
29. 1, 1, 1, 1, 1, 1, 1, 1,
30. ], 'y': 0})

33. # No. of Training Examples
34. intM = len(aryT)
36. # Learning Rate
37. intAlpha = 0.1
39. # Init Array Parameters
40. aryTheta = [
41. 0, 0, 0, 0, 0, 0, 0, 0,
42. 0, 0, 0, 0, 0, 0, 0, 0,
43. 0, 0, 0, 0, 0, 0, 0, 0,
44. 0, 0, 0, 0, 0, 0, 0, 0,
45. 0, 0, 0, 0, 0, 0, 0, 0,
46. 0, 0, 0, 0, 0, 0, 0, 0,
47. 0, 0, 0, 0, 0, 0, 0, 0,
48. 0, 0, 0, 0, 0, 0, 0, 0,
49. ]
51. # Hypothesis H, t stands for theta
52. # G(x) = t0 x0 + t1 x1 + ... +
53. # H(z) = 1 / (1 + e^(-z))
55. intTrainTimes = 100
57. # Iterates for intTrainTimes
58. **for** t in range(intTrainTimes):
60. # Iterates for each feature
61. **for** i in range(intN):
63. # Sum of error between Y and Hypothesis
64. intSum = 0
66. # Iterates for each training example
67. **for** j in range(intM):
69. intTemp1 = 0
70. intTemp2 = 0
72. # Sum up to get Theta transposed X
73. **for** z in range(intN):
74. intTemp1 += aryTheta[z] \* aryT[j]['x'][z]
76. # Error between Y and Hypothesis in jth training example
77. intTemp2 = aryT[j]['y'] - (1 / (1 + math.exp(-intTemp1)))
78. intTemp2 = intTemp2 \* aryT[j]['x'][i]
79. intSum += intTemp2
81. # Update Theta i
82. aryTheta[i] = aryTheta[i] + intAlpha \* intSum
84. # Print all Theta(s)
85. print('aryTheta: ' + str(aryTheta))
87. # Testing
89. # Array which stores testing values
90. aryTest = [
91. 0, 1, 0, 0, 1, 0, 1, 1,
92. 0, 1, 0, 1, 0, 1, 1, 1,
93. 1, 1, 1, 0, 1, 0, 1, 1,
94. 0, 0, 0, 1, 1, 1, 0, 0,
95. 0, 1, 0, 0, 1, 0, 1, 0,
96. 0, 1, 0, 1, 0, 0, 0, 0,
97. 1, 0, 1, 0, 1, 0, 0, 0,
98. 1, 1, 0, 0, 0, 1, 0, 0,
99. ]
101. intTemp1 = 0
103. **for** i in range(0, intN):
104. intTemp1 += aryTheta[i] \* aryTest[i]
106. # The output of the hypothesis
107. intAnswer = 1 / (1 + math.exp(-intTemp1))
109. # Print the output given by the hypothesis
110. print("Test Answer: " + str(intAnswer))

Program: ML0401.py

In this program, aryT stores 2 training examples only. The first one stores all upper points while the second one stores all lower points.

Then the program loops 100 times and using the gradient descent algorithm, finding out all values of thetas. Note that there are 64 features and 64 thetas in this program, and the thetas are related to every points on a graph.

Lastly, randomly insert some 1s and 0s into aryTest, then the program will output the predicted value, which is between 0 and 1. If it is higher than 0.5, we know that there are more upper points. On the contrary, we know that there are more lower points. If the value is very close to 0.5, we know that the number of upper and lower points are close.

And this is the a simple python program which uses gradient descent algorithm in logistic regression.

# 5 Practical Review Exercise

## 5.1 Gradient Descent Practical Exercise

**Question:**

According to Table 3.1.1:

|  |  |
| --- | --- |
| Height | Weight |
| 10 | 0.5 |
| 20 | 1.5 |
| 40 | 3.5 |
| 60 | 8 |
| 80 | 12 |
| 100 | 20 |
| 120 | 30 |
| 140 | 40 |
| 160 | 55 |
| 170 | 70 |
| 180 | 76 |

Table 3.1.1

Write a python program ML0501.py, to find the equation of the curve as shown in Figure 3.1.2:

Figure 3.1.2

i.e.

y = 0.000010x3 + 0.000057x2 + 0.089824x – 0.292563

**Answer:**

1. # In this Machine Learning Example, we try to use Gradient Descent
2. # to draw a curve, with given weights and heights of people
4. **import** math
5. **from** matplotlib **import** pyplot as plt
7. # Define No. of Features of x
8. intN = 3
10. # Define Learning Rate
11. aryAlpha = [
12. 0.1,
13. 0.00001,
14. 0.000000001,
15. 0.0000000000001,
16. ]
18. # Define Training Examples
19. aryT = []
20. aryT.append({'x': [10], 'y': 0.5})
21. aryT.append({'x': [20], 'y': 1.5})
22. aryT.append({'x': [40], 'y': 3.5})
23. aryT.append({'x': [60], 'y': 8})
24. aryT.append({'x': [80], 'y': 12})
25. aryT.append({'x': [100], 'y': 20})
26. aryT.append({'x': [120], 'y': 30})
27. aryT.append({'x': [140], 'y': 40})
28. aryT.append({'x': [160], 'y': 55})
29. aryT.append({'x': [170], 'y': 70})
30. aryT.append({'x': [180], 'y': 76})
32. # No. of Training Examples
33. intM = len(aryT)
35. # Init Array Parameters
36. aryTheta = [
37. 0,
38. 0,
39. 0,
40. 0,
41. ]
43. # Init maximum training times
44. intMaxTrainTimes = 10000


48. # Iterates for intMaxTrainTimes
49. **for** t **in** range(intMaxTrainTimes):
51. # Iterate for each feature
52. **for** i **in** range(0, intN+1):
53. # Init Summation
54. intSum = 0
56. # Iterate for each training example
57. **for** j **in** range(intM):
59. # Init temp for calculation
60. intTemp = 0
62. # Sum up to get predicted value of the hypothesis
63. **for** n **in** range(0, intN+1):
64. **if** n > 0:
65. intTemp += aryTheta[n] \* math.pow(aryT[j]['x'][0], n)
66. **else**:
67. intTemp += aryTheta[n]
69. # Calculate Error
70. intTemp = intTemp - aryT[j]['y']
72. **if** i > 0:
73. intTemp = intTemp \* math.pow(aryT[j]['x'][0], i)
75. intSum += intTemp
76. #End for
78. # Calculate New Theta(0)
79. aryTheta[i] = aryTheta[i] - (1 / intM) \* aryAlpha[i] \* intSum


83. # Print all Theta(s)
84. **print**('aryTheta[0]: ' + str(aryTheta[0]))
85. **print**('aryTheta[1]: ' + str(aryTheta[1]))
86. **print**('aryTheta[2]: ' + str(aryTheta[2]))
87. **print**('aryTheta[3]: ' + str(aryTheta[3]))


91. # Check Error
92. intSum = 0
93. **for** m **in** range(intM):
94. intTemp = 0
95. **for** n **in** range(0, intN+1):
96. **if** n > 0:
97. intTemp += aryTheta[n] \* math.pow(aryT[m]['x'][0], n)
98. **else**:
99. intTemp += aryTheta[n]
100. intSum += math.pow(intTemp - aryT[m]['y'], 2)
101. **print**("Error: " + str(intSum))


105. # Plot graph
106. aryX = []
107. aryY = []
108. aryH = []
109. **for** i **in** range(intM):
110. intTemp = 0
111. **for** j **in** range(0, intN+1):
112. **if** j > 0:
113. intTemp += aryTheta[j] \* math.pow(aryT[i]['x'][0], j)
114. **else**:
115. intTemp += aryTheta[j]
116. aryX.append(aryT[i]['x'][0])
117. aryY.append(aryT[i]['y'])
118. aryH.append(intTemp)
120. plt.plot(aryX, aryY, 'bx')
121. plt.plot(aryX, aryH, 'g')
122. plt.show()

Program: ML0501.py

In the beginning of the program, we add a line:

from matplotlib import pyplot as plt

We import pyplot for plotting a graph with our hypothesis in the end.

Different from the previous programs, there are separate learning rates for thetas this time. This is because learning rates are set regarding the input values. If the input values are large, the learning rates must be small since a large learning rate will lead to a great change to theta and the hypothesis, which may make it hard to converge. In this program, the learning rate decreases as the power of x increases.

If all learning rates are set to a very small value, the thetas will converge slowly. It may not converge after 100,000 iterations. Therefore, learning rates have to be small for large inputs and they have to be large enough for small inputs, then the thetas will be able to converge quickly and accurately.

After the program has run all iterations and print the thetas out, it checks error. This means it compares the output of the hypothesis and actual answer of each training example. If we run the program, we get 21.61 for our hypothesis. If we change the thetas to values provided by Excel:

1. # Check Error
2. intSum = 0
3. aryTheta = [
4. 0.292563,
5. 0.089824,
6. 0.000057,
7. 0.000010,
8. ]
9. **for** m **in** range(intM):
10. intTemp = 0
11. **for** n **in** range(0, intN+1):
12. **if** n > 0:
13. intTemp += aryTheta[n] \* math.pow(aryT[m]['x'][0], n)
14. **else**:
15. intTemp += aryTheta[n]
16. intSum += math.pow(intTemp - aryT[m]['y'], 2)
17. **print**("Error: " + str(intSum))

We get 24.03, which is larger than the error value output by the hypothesis. This means our hypothesis fits the training data better than Excel.

Lastly, the program will plot the graph using pyplot, which is mentioned above:

图片包含 游戏机

描述已自动生成

Figure 5.1.1

## 5.2 Logistic Regression Practical Exercise

**Question:**

Write a Python Program ML0502.py, to recognize Numeric Digits from 0 to 4, shown in a 8 x 8 LED panel. Each digit should have at least 10 training examples. (i.e. a total of at least 50 training examples)

**Answer:**

1. # In this Machine Learning Example, we try to use Logistic Regression
2. # to distinguish numeric digits shown in a 8 x 8 LED panel
4. **import** math
6. # Define No. of Features of x
7. intN = 64
9. # Define Training Examples
10. aryT = []
12. # Define Total No. of Digits, N means from 0 to N - 1
13. intNumOfDigits = 5

16. # Zero
18. # 0:1
19. aryT.append({'x': [
20. 0, 0, 1, 1, 1, 1, 0, 0,
21. 0, 1, 0, 0, 0, 0, 1, 0,
22. 0, 1, 0, 0, 0, 0, 1, 0,
23. 0, 1, 0, 0, 0, 0, 1, 0,
24. 0, 1, 0, 0, 0, 0, 1, 0,
25. 0, 1, 0, 0, 0, 0, 1, 0,
26. 0, 1, 0, 0, 0, 0, 1, 0,
27. 0, 0, 1, 1, 1, 1, 0, 0,
28. ], 'y': 0})
29. # 0:2
30. aryT.append({'x': [
31. 0, 1, 1, 1, 1, 1, 0, 0,
32. 1, 0, 0, 0, 0, 0, 1, 0,
33. 1, 0, 0, 0, 0, 0, 1, 0,
34. 1, 0, 0, 0, 0, 0, 1, 0,
35. 1, 0, 0, 0, 0, 0, 1, 0,
36. 1, 0, 0, 0, 0, 0, 1, 0,
37. 1, 0, 0, 0, 0, 0, 1, 0,
38. 0, 1, 1, 1, 1, 1, 0, 0,
39. ], 'y': 0})
40. # 0:3
41. aryT.append({'x': [
42. 0, 0, 1, 1, 1, 1, 1, 0,
43. 0, 1, 0, 0, 0, 0, 0, 1,
44. 0, 1, 0, 0, 0, 0, 0, 1,
45. 0, 1, 0, 0, 0, 0, 0, 1,
46. 0, 1, 0, 0, 0, 0, 0, 1,
47. 0, 1, 0, 0, 0, 0, 0, 1,
48. 0, 1, 0, 0, 0, 0, 0, 1,
49. 0, 0, 1, 1, 1, 1, 1, 0,
50. ], 'y': 0})
51. # 0:4
52. aryT.append({'x': [
53. 0, 1, 1, 1, 1, 1, 1, 0,
54. 0, 1, 0, 0, 0, 0, 1, 0,
55. 0, 1, 0, 0, 0, 0, 1, 0,
56. 0, 1, 0, 0, 0, 0, 1, 0,
57. 0, 1, 0, 0, 0, 0, 1, 0,
58. 0, 1, 0, 0, 0, 0, 1, 0,
59. 0, 1, 0, 0, 0, 0, 1, 0,
60. 0, 1, 1, 1, 1, 1, 1, 0,
61. ], 'y': 0})
62. # 0:5
63. aryT.append({'x': [
64. 0, 0, 0, 1, 1, 0, 0, 0,
65. 0, 0, 1, 0, 0, 1, 0, 0,
66. 0, 0, 1, 0, 0, 1, 0, 0,
67. 0, 0, 1, 0, 0, 1, 0, 0,
68. 0, 0, 1, 0, 0, 1, 0, 0,
69. 0, 0, 1, 0, 0, 1, 0, 0,
70. 0, 0, 1, 0, 0, 1, 0, 0,
71. 0, 0, 0, 1, 1, 0, 0, 0,
72. ], 'y': 0})
73. # 0:6
74. aryT.append({'x': [
75. 0, 0, 1, 1, 1, 1, 0, 0,
76. 0, 0, 1, 0, 0, 1, 0, 0,
77. 0, 0, 1, 0, 0, 1, 0, 0,
78. 0, 0, 1, 0, 0, 1, 0, 0,
79. 0, 0, 1, 0, 0, 1, 0, 0,
80. 0, 0, 1, 0, 0, 1, 0, 0,
81. 0, 0, 1, 0, 0, 1, 0, 0,
82. 0, 0, 1, 1, 1, 1, 0, 0,
83. ], 'y': 0})
84. # 0:7
85. aryT.append({'x': [
86. 0, 0, 1, 1, 1, 1, 0, 0,
87. 0, 1, 1, 1, 1, 1, 1, 0,
88. 0, 1, 1, 0, 0, 1, 1, 0,
89. 0, 1, 1, 0, 0, 1, 1, 0,
90. 0, 1, 1, 0, 0, 1, 1, 0,
91. 0, 1, 1, 0, 0, 1, 1, 0,
92. 0, 1, 1, 1, 1, 1, 1, 0,
93. 0, 0, 1, 1, 1, 1, 0, 0,
94. ], 'y': 0})
95. # 0:8
96. aryT.append({'x': [
97. 0, 1, 1, 1, 1, 0, 0, 0,
98. 1, 1, 1, 1, 1, 1, 0, 0,
99. 1, 1, 0, 0, 1, 1, 0, 0,
100. 1, 1, 0, 0, 1, 1, 0, 0,
101. 1, 1, 0, 0, 1, 1, 0, 0,
102. 1, 1, 0, 0, 1, 1, 0, 0,
103. 1, 1, 1, 1, 1, 1, 0, 0,
104. 0, 1, 1, 1, 1, 0, 0, 0,
105. ], 'y': 0})
106. # 0:9
107. aryT.append({'x': [
108. 0, 0, 0, 1, 1, 1, 1, 0,
109. 0, 0, 1, 1, 1, 1, 1, 1,
110. 0, 0, 1, 1, 0, 0, 1, 1,
111. 0, 0, 1, 1, 0, 0, 1, 1,
112. 0, 0, 1, 1, 0, 0, 1, 1,
113. 0, 0, 1, 1, 0, 0, 1, 1,
114. 0, 0, 1, 1, 1, 1, 1, 1,
115. 0, 0, 0, 1, 1, 1, 1, 0,
116. ], 'y': 0})
117. # 0:10
118. aryT.append({'x': [
119. 0, 1, 1, 1, 1, 1, 1, 0,
120. 1, 0, 0, 0, 0, 0, 0, 1,
121. 1, 0, 0, 0, 0, 0, 0, 1,
122. 1, 0, 0, 0, 0, 0, 0, 1,
123. 1, 0, 0, 0, 0, 0, 0, 1,
124. 1, 0, 0, 0, 0, 0, 0, 1,
125. 1, 0, 0, 0, 0, 0, 0, 1,
126. 0, 1, 1, 1, 1, 1, 1, 0,
127. ], 'y': 0})

130. # One
132. # 1:1
133. aryT.append({'x': [
134. 0, 0, 0, 1, 0, 0, 0, 0,
135. 0, 0, 0, 1, 0, 0, 0, 0,
136. 0, 0, 0, 1, 0, 0, 0, 0,
137. 0, 0, 0, 1, 0, 0, 0, 0,
138. 0, 0, 0, 1, 0, 0, 0, 0,
139. 0, 0, 0, 1, 0, 0, 0, 0,
140. 0, 0, 0, 1, 0, 0, 0, 0,
141. 0, 0, 0, 1, 0, 0, 0, 0,
142. ], 'y': 1})
143. # 1:2
144. aryT.append({'x': [
145. 0, 0, 0, 0, 1, 0, 0, 0,
146. 0, 0, 0, 0, 1, 0, 0, 0,
147. 0, 0, 0, 0, 1, 0, 0, 0,
148. 0, 0, 0, 0, 1, 0, 0, 0,
149. 0, 0, 0, 0, 1, 0, 0, 0,
150. 0, 0, 0, 0, 1, 0, 0, 0,
151. 0, 0, 0, 0, 1, 0, 0, 0,
152. 0, 0, 0, 0, 1, 0, 0, 0,
153. ], 'y': 1})
154. # 1:3
155. aryT.append({'x': [
156. 0, 0, 1, 0, 0, 0, 0, 0,
157. 0, 0, 1, 0, 0, 0, 0, 0,
158. 0, 0, 1, 0, 0, 0, 0, 0,
159. 0, 0, 1, 0, 0, 0, 0, 0,
160. 0, 0, 1, 0, 0, 0, 0, 0,
161. 0, 0, 1, 0, 0, 0, 0, 0,
162. 0, 0, 1, 0, 0, 0, 0, 0,
163. 0, 0, 1, 0, 0, 0, 0, 0,
164. ], 'y': 1})
165. # 1:4
166. aryT.append({'x': [
167. 0, 0, 0, 0, 0, 1, 0, 0,
168. 0, 0, 0, 0, 0, 1, 0, 0,
169. 0, 0, 0, 0, 0, 1, 0, 0,
170. 0, 0, 0, 0, 0, 1, 0, 0,
171. 0, 0, 0, 0, 0, 1, 0, 0,
172. 0, 0, 0, 0, 0, 1, 0, 0,
173. 0, 0, 0, 0, 0, 1, 0, 0,
174. 0, 0, 0, 0, 0, 1, 0, 0,
175. ], 'y': 1})
176. # 1:5
177. aryT.append({'x': [
178. 0, 0, 0, 1, 1, 0, 0, 0,
179. 0, 0, 0, 1, 1, 0, 0, 0,
180. 0, 0, 0, 1, 1, 0, 0, 0,
181. 0, 0, 0, 1, 1, 0, 0, 0,
182. 0, 0, 0, 1, 1, 0, 0, 0,
183. 0, 0, 0, 1, 1, 0, 0, 0,
184. 0, 0, 0, 1, 1, 0, 0, 0,
185. 0, 0, 0, 1, 1, 0, 0, 0,
186. ], 'y': 1})
187. # 1:6
188. aryT.append({'x': [
189. 0, 0, 0, 1, 1, 0, 0, 0,
190. 0, 0, 1, 1, 1, 0, 0, 0,
191. 0, 0, 0, 1, 1, 0, 0, 0,
192. 0, 0, 0, 1, 1, 0, 0, 0,
193. 0, 0, 0, 1, 1, 0, 0, 0,
194. 0, 0, 0, 1, 1, 0, 0, 0,
195. 0, 0, 0, 1, 1, 0, 0, 0,
196. 0, 1, 1, 1, 1, 1, 1, 0,
197. ], 'y': 1})
198. # 1:7
199. aryT.append({'x': [
200. 0, 0, 1, 1, 0, 0, 0, 0,
201. 0, 0, 1, 1, 0, 0, 0, 0,
202. 0, 0, 1, 1, 0, 0, 0, 0,
203. 0, 0, 1, 1, 0, 0, 0, 0,
204. 0, 0, 1, 1, 0, 0, 0, 0,
205. 0, 0, 1, 1, 0, 0, 0, 0,
206. 0, 0, 1, 1, 0, 0, 0, 0,
207. 0, 0, 1, 1, 0, 0, 0, 0,
208. ], 'y': 1})
209. # 1:8
210. aryT.append({'x': [
211. 0, 0, 1, 1, 0, 0, 0, 0,
212. 0, 1, 1, 1, 0, 0, 0, 0,
213. 0, 0, 1, 1, 0, 0, 0, 0,
214. 0, 0, 1, 1, 0, 0, 0, 0,
215. 0, 0, 1, 1, 0, 0, 0, 0,
216. 0, 0, 1, 1, 0, 0, 0, 0,
217. 0, 0, 1, 1, 0, 0, 0, 0,
218. 1, 1, 1, 1, 1, 1, 0, 0,
219. ], 'y': 1})
220. # 1:9
221. aryT.append({'x': [
222. 0, 0, 0, 0, 1, 1, 0, 0,
223. 0, 0, 0, 0, 1, 1, 0, 0,
224. 0, 0, 0, 0, 1, 1, 0, 0,
225. 0, 0, 0, 0, 1, 1, 0, 0,
226. 0, 0, 0, 0, 1, 1, 0, 0,
227. 0, 0, 0, 0, 1, 1, 0, 0,
228. 0, 0, 0, 0, 1, 1, 0, 0,
229. 0, 0, 0, 0, 1, 1, 0, 0,
230. ], 'y': 1})
231. # 1:10
232. aryT.append({'x': [
233. 0, 0, 0, 0, 1, 1, 0, 0,
234. 0, 0, 0, 1, 1, 1, 0, 0,
235. 0, 0, 0, 0, 1, 1, 0, 0,
236. 0, 0, 0, 0, 1, 1, 0, 0,
237. 0, 0, 0, 0, 1, 1, 0, 0,
238. 0, 0, 0, 0, 1, 1, 0, 0,
239. 0, 0, 0, 0, 1, 1, 0, 0,
240. 0, 0, 1, 1, 1, 1, 1, 1,
241. ], 'y': 1})

244. # Two
246. # 2:1
247. aryT.append({'x': [
248. 0, 0, 1, 1, 1, 1, 0, 0,
249. 0, 1, 0, 0, 0, 0, 1, 0,
250. 0, 0, 0, 0, 0, 0, 1, 0,
251. 0, 0, 0, 0, 1, 1, 0, 0,
252. 0, 0, 1, 1, 0, 0, 0, 0,
253. 0, 1, 0, 0, 0, 0, 0, 0,
254. 0, 1, 0, 0, 0, 0, 1, 0,
255. 0, 0, 1, 1, 1, 1, 0, 0,
256. ], 'y': 2})
257. # 2:2
258. aryT.append({'x': [
259. 0, 1, 1, 1, 1, 1, 0, 0,
260. 0, 1, 0, 0, 0, 0, 1, 0,
261. 0, 0, 0, 0, 0, 0, 1, 0,
262. 0, 0, 0, 0, 1, 1, 1, 0,
263. 0, 1, 1, 1, 0, 0, 0, 0,
264. 0, 1, 0, 0, 0, 0, 0, 0,
265. 0, 1, 0, 0, 0, 0, 1, 0,
266. 0, 0, 1, 1, 1, 1, 1, 0,
267. ], 'y': 2})
268. # 2:3
269. aryT.append({'x': [
270. 0, 0, 1, 1, 1, 1, 0, 0,
271. 0, 0, 0, 0, 0, 1, 0, 0,
272. 0, 0, 0, 0, 0, 1, 0, 0,
273. 0, 0, 1, 1, 1, 1, 0, 0,
274. 0, 0, 1, 0, 0, 0, 0, 0,
275. 0, 0, 1, 0, 0, 0, 0, 0,
276. 0, 0, 1, 0, 0, 0, 0, 0,
277. 0, 0, 1, 1, 1, 1, 0, 0,
278. ], 'y': 2})
279. # 2:4
280. aryT.append({'x': [
281. 0, 0, 1, 1, 1, 1, 0, 0,
282. 0, 0, 0, 0, 0, 1, 0, 0,
283. 0, 0, 0, 0, 0, 1, 0, 0,
284. 0, 0, 0, 0, 0, 1, 0, 0,
285. 0, 0, 1, 1, 1, 1, 0, 0,
286. 0, 0, 1, 0, 0, 0, 0, 0,
287. 0, 0, 1, 0, 0, 0, 0, 0,
288. 0, 0, 1, 1, 1, 1, 0, 0,
289. ], 'y': 2})
290. # 2:5
291. aryT.append({'x': [
292. 0, 1, 1, 1, 1, 0, 0, 0,
293. 0, 0, 0, 0, 1, 0, 0, 0,
294. 0, 0, 0, 0, 1, 0, 0, 0,
295. 0, 1, 1, 1, 1, 0, 0, 0,
296. 0, 1, 0, 0, 0, 0, 0, 0,
297. 0, 1, 0, 0, 0, 0, 0, 0,
298. 0, 1, 0, 0, 0, 0, 0, 0,
299. 0, 1, 1, 1, 1, 0, 0, 0,
300. ], 'y': 2})
301. # 2:6
302. aryT.append({'x': [
303. 0, 0, 0, 1, 1, 1, 1, 0,
304. 0, 0, 0, 0, 0, 0, 1, 0,
305. 0, 0, 0, 0, 0, 0, 1, 0,
306. 0, 0, 0, 1, 1, 1, 1, 0,
307. 0, 0, 0, 1, 0, 0, 0, 0,
308. 0, 0, 0, 1, 0, 0, 0, 0,
309. 0, 0, 0, 1, 0, 0, 0, 0,
310. 0, 0, 0, 1, 1, 1, 1, 0,
311. ], 'y': 2})
312. # 2:7
313. aryT.append({'x': [
314. 0, 1, 1, 1, 1, 0, 0, 0,
315. 0, 0, 0, 0, 1, 0, 0, 0,
316. 0, 0, 0, 0, 1, 0, 0, 0,
317. 0, 0, 0, 0, 1, 0, 0, 0,
318. 0, 1, 1, 1, 1, 0, 0, 0,
319. 0, 1, 0, 0, 0, 0, 0, 0,
320. 0, 1, 0, 0, 0, 0, 0, 0,
321. 0, 1, 1, 1, 1, 0, 0, 0,
322. ], 'y': 2})
323. # 2:8
324. aryT.append({'x': [
325. 0, 0, 0, 1, 1, 1, 1, 0,
326. 0, 0, 0, 0, 0, 0, 1, 0,
327. 0, 0, 0, 0, 0, 0, 1, 0,
328. 0, 0, 0, 0, 0, 0, 1, 0,
329. 0, 0, 0, 1, 1, 1, 1, 0,
330. 0, 0, 0, 1, 0, 0, 0, 0,
331. 0, 0, 0, 1, 0, 0, 0, 0,
332. 0, 0, 0, 1, 1, 1, 1, 0,
333. ], 'y': 2})
334. # 2:9
335. aryT.append({'x': [
336. 0, 1, 1, 1, 1, 1, 1, 0,
337. 1, 0, 0, 0, 0, 0, 0, 1,
338. 0, 0, 0, 0, 0, 0, 0, 1,
339. 0, 1, 1, 1, 1, 1, 1, 0,
340. 1, 0, 0, 0, 0, 0, 0, 0,
341. 1, 0, 0, 0, 0, 0, 0, 0,
342. 1, 0, 0, 0, 0, 0, 0, 1,
343. 0, 1, 1, 1, 1, 1, 1, 0,
344. ], 'y': 2})
345. # 2:10
346. aryT.append({'x': [
347. 0, 1, 1, 1, 1, 1, 1, 0,
348. 1, 0, 0, 0, 0, 0, 0, 1,
349. 0, 0, 0, 0, 0, 0, 0, 1,
350. 0, 0, 0, 0, 0, 0, 0, 1,
351. 0, 1, 1, 1, 1, 1, 1, 0,
352. 1, 0, 0, 0, 0, 0, 0, 0,
353. 1, 0, 0, 0, 0, 0, 0, 1,
354. 0, 1, 1, 1, 1, 1, 1, 0,
355. ], 'y': 2})

358. # Three
360. # 3:1
361. aryT.append({'x': [
362. 0, 0, 1, 1, 1, 1, 0, 0,
363. 0, 1, 0, 0, 0, 0, 1, 0,
364. 0, 0, 0, 0, 0, 0, 1, 0,
365. 0, 0, 1, 1, 1, 1, 0, 0,
366. 0, 0, 0, 0, 0, 0, 1, 0,
367. 0, 0, 0, 0, 0, 0, 1, 0,
368. 0, 1, 0, 0, 0, 0, 1, 0,
369. 0, 0, 1, 1, 1, 1, 0, 0,
370. ], 'y': 3})
371. # 3:2
372. aryT.append({'x': [
373. 0, 1, 1, 1, 1, 0, 0, 0,
374. 1, 0, 0, 0, 0, 1, 0, 0,
375. 0, 0, 0, 0, 0, 1, 0, 0,
376. 0, 1, 1, 1, 1, 0, 0, 0,
377. 0, 0, 0, 0, 0, 1, 0, 0,
378. 0, 0, 0, 0, 0, 1, 0, 0,
379. 1, 0, 0, 0, 0, 1, 0, 0,
380. 0, 1, 1, 1, 1, 0, 0, 0,
381. ], 'y': 3})
382. # 3:3
383. aryT.append({'x': [
384. 0, 0, 0, 1, 1, 1, 1, 0,
385. 0, 0, 1, 0, 0, 0, 0, 1,
386. 0, 0, 0, 0, 0, 0, 0, 1,
387. 0, 0, 0, 1, 1, 1, 1, 0,
388. 0, 0, 0, 0, 0, 0, 0, 1,
389. 0, 0, 0, 0, 0, 0, 0, 1,
390. 0, 0, 1, 0, 0, 0, 0, 1,
391. 0, 0, 0, 1, 1, 1, 1, 0,
392. ], 'y': 3})
393. # 3:4
394. aryT.append({'x': [
395. 0, 0, 1, 1, 1, 1, 0, 0,
396. 0, 1, 0, 0, 0, 0, 1, 0,
397. 0, 0, 0, 0, 0, 0, 1, 0,
398. 0, 0, 1, 1, 1, 1, 0, 0,
399. 0, 0, 1, 1, 1, 1, 1, 0,
400. 0, 0, 0, 0, 0, 0, 1, 0,
401. 0, 1, 0, 0, 0, 0, 1, 0,
402. 0, 0, 1, 1, 1, 1, 0, 0,
403. ], 'y': 3})
404. # 3:5
405. aryT.append({'x': [
406. 0, 1, 1, 1, 1, 0, 0, 0,
407. 1, 0, 0, 0, 0, 1, 0, 0,
408. 0, 0, 0, 0, 0, 1, 0, 0,
409. 0, 1, 1, 1, 1, 0, 0, 0,
410. 0, 1, 1, 1, 1, 1, 0, 0,
411. 0, 0, 0, 0, 0, 1, 0, 0,
412. 1, 0, 0, 0, 0, 1, 0, 0,
413. 0, 1, 1, 1, 1, 0, 0, 0,
414. ], 'y': 3})
415. # 3:6
416. aryT.append({'x': [
417. 0, 0, 0, 1, 1, 1, 1, 0,
418. 0, 0, 1, 0, 0, 0, 0, 1,
419. 0, 0, 0, 0, 0, 0, 0, 1,
420. 0, 0, 0, 1, 1, 1, 1, 0,
421. 0, 0, 0, 1, 1, 1, 1, 1,
422. 0, 0, 0, 0, 0, 0, 0, 1,
423. 0, 0, 1, 0, 0, 0, 0, 1,
424. 0, 0, 0, 1, 1, 1, 1, 0,
425. ], 'y': 3})
426. # 3:7
427. aryT.append({'x': [
428. 0, 0, 1, 1, 1, 1, 0, 0,
429. 0, 1, 0, 0, 0, 0, 1, 0,
430. 0, 0, 0, 0, 0, 0, 1, 0,
431. 0, 0, 0, 0, 0, 0, 1, 0,
432. 0, 0, 1, 1, 1, 1, 0, 0,
433. 0, 0, 0, 0, 0, 0, 1, 0,
434. 0, 1, 0, 0, 0, 0, 1, 0,
435. 0, 0, 1, 1, 1, 1, 0, 0,
436. ], 'y': 3})
437. # 3:8
438. aryT.append({'x': [
439. 0, 1, 1, 1, 1, 0, 0, 0,
440. 1, 0, 0, 0, 0, 1, 0, 0,
441. 0, 0, 0, 0, 0, 1, 0, 0,
442. 0, 0, 0, 0, 0, 1, 0, 0,
443. 0, 1, 1, 1, 1, 0, 0, 0,
444. 0, 0, 0, 0, 0, 1, 0, 0,
445. 1, 0, 0, 0, 0, 1, 0, 0,
446. 0, 1, 1, 1, 1, 0, 0, 0,
447. ], 'y': 3})
448. # 3:9
449. aryT.append({'x': [
450. 0, 0, 0, 1, 1, 1, 1, 0,
451. 0, 0, 1, 0, 0, 0, 0, 1,
452. 0, 0, 0, 0, 0, 0, 0, 1,
453. 0, 0, 0, 0, 0, 0, 0, 1,
454. 0, 0, 0, 1, 1, 1, 1, 0,
455. 0, 0, 0, 0, 0, 0, 0, 1,
456. 0, 0, 1, 0, 0, 0, 0, 1,
457. 0, 0, 0, 1, 1, 1, 1, 0,
458. ], 'y': 3})
459. # 3:10
460. aryT.append({'x': [
461. 0, 1, 1, 1, 1, 1, 1, 0,
462. 1, 0, 0, 0, 0, 0, 0, 1,
463. 0, 0, 0, 0, 0, 0, 0, 1,
464. 0, 0, 1, 1, 1, 1, 1, 0,
465. 0, 0, 0, 0, 0, 0, 0, 1,
466. 0, 0, 0, 0, 0, 0, 0, 1,
467. 1, 0, 0, 0, 0, 0, 0, 1,
468. 0, 1, 1, 1, 1, 1, 1, 0,
469. ], 'y': 3})

472. # Four
474. # 4:1
475. aryT.append({'x': [
476. 1, 1, 0, 0, 0, 0, 1, 1,
477. 1, 1, 0, 0, 0, 0, 1, 1,
478. 1, 1, 0, 0, 0, 0, 1, 1,
479. 1, 1, 1, 1, 1, 1, 1, 1,
480. 0, 1, 1, 1, 1, 1, 1, 1,
481. 0, 0, 0, 0, 0, 0, 1, 1,
482. 0, 0, 0, 0, 0, 0, 1, 1,
483. 0, 0, 0, 0, 0, 0, 1, 1,
484. ], 'y': 4})
485. # 4:2
486. aryT.append({'x': [
487. 1, 0, 0, 0, 0, 0, 0, 1,
488. 1, 0, 0, 0, 0, 0, 0, 1,
489. 1, 0, 0, 0, 0, 0, 0, 1,
490. 0, 1, 1, 1, 1, 1, 1, 1,
491. 0, 0, 0, 0, 0, 0, 0, 1,
492. 0, 0, 0, 0, 0, 0, 0, 1,
493. 0, 0, 0, 0, 0, 0, 0, 1,
494. 0, 0, 0, 0, 0, 0, 0, 1,
495. ], 'y': 4})
496. # 4:3
497. aryT.append({'x': [
498. 0, 1, 0, 0, 0, 0, 1, 0,
499. 0, 1, 0, 0, 0, 0, 1, 0,
500. 0, 1, 0, 0, 0, 0, 1, 0,
501. 0, 0, 1, 1, 1, 1, 1, 0,
502. 0, 0, 0, 0, 0, 0, 1, 0,
503. 0, 0, 0, 0, 0, 0, 1, 0,
504. 0, 0, 0, 0, 0, 0, 1, 0,
505. 0, 0, 0, 0, 0, 0, 1, 0,
506. ], 'y': 4})
507. # 4:4
508. aryT.append({'x': [
509. 1, 0, 0, 0, 0, 1, 0, 0,
510. 1, 0, 0, 0, 0, 1, 0, 0,
511. 1, 0, 0, 0, 0, 1, 0, 0,
512. 0, 1, 1, 1, 1, 1, 0, 0,
513. 0, 0, 0, 0, 0, 1, 0, 0,
514. 0, 0, 0, 0, 0, 1, 0, 0,
515. 0, 0, 0, 0, 0, 1, 0, 0,
516. 0, 0, 0, 0, 0, 1, 0, 0,
517. ], 'y': 4})
518. # 4:5
519. aryT.append({'x': [
520. 0, 0, 1, 0, 0, 0, 0, 1,
521. 0, 0, 1, 0, 0, 0, 0, 1,
522. 0, 0, 1, 0, 0, 0, 0, 1,
523. 0, 0, 0, 1, 1, 1, 1, 1,
524. 0, 0, 0, 0, 0, 0, 0, 1,
525. 0, 0, 0, 0, 0, 0, 0, 1,
526. 0, 0, 0, 0, 0, 0, 0, 1,
527. 0, 0, 0, 0, 0, 0, 0, 1,
528. ], 'y': 4})
529. # 4:6
530. aryT.append({'x': [
531. 0, 1, 0, 0, 0, 1, 0, 0,
532. 0, 1, 0, 0, 0, 1, 0, 0,
533. 0, 1, 0, 0, 0, 1, 0, 0,
534. 0, 0, 1, 1, 1, 1, 0, 0,
535. 0, 0, 0, 0, 0, 1, 0, 0,
536. 0, 0, 0, 0, 0, 1, 0, 0,
537. 0, 0, 0, 0, 0, 1, 0, 0,
538. 0, 0, 0, 0, 0, 1, 0, 0,
539. ], 'y': 4})
540. # 4:7
541. aryT.append({'x': [
542. 0, 0, 1, 0, 0, 0, 1, 0,
543. 0, 0, 1, 0, 0, 0, 1, 0,
544. 0, 0, 1, 0, 0, 0, 1, 0,
545. 0, 0, 0, 1, 1, 1, 1, 0,
546. 0, 0, 0, 0, 0, 0, 1, 0,
547. 0, 0, 0, 0, 0, 0, 1, 0,
548. 0, 0, 0, 0, 0, 0, 1, 0,
549. 0, 0, 0, 0, 0, 0, 1, 0,
550. ], 'y': 4})
551. # 4:8
552. aryT.append({'x': [
553. 0, 0, 0, 0, 1, 0, 0, 0,
554. 0, 0, 0, 1, 1, 0, 0, 0,
555. 0, 0, 1, 0, 1, 0, 0, 0,
556. 0, 1, 0, 0, 1, 0, 0, 0,
557. 1, 1, 1, 1, 1, 1, 1, 1,
558. 0, 0, 0, 0, 1, 0, 0, 0,
559. 0, 0, 0, 0, 1, 0, 0, 0,
560. 0, 0, 0, 0, 1, 0, 0, 0,
561. ], 'y': 4})
562. # 4:9
563. aryT.append({'x': [
564. 0, 1, 0, 0, 1, 0, 0, 0,
565. 0, 1, 0, 0, 1, 0, 0, 0,
566. 0, 1, 0, 0, 1, 0, 0, 0,
567. 0, 1, 0, 0, 1, 0, 0, 0,
568. 0, 1, 1, 1, 1, 1, 1, 1,
569. 0, 0, 0, 0, 1, 0, 0, 0,
570. 0, 0, 0, 0, 1, 0, 0, 0,
571. 0, 0, 0, 0, 1, 0, 0, 0,
572. ], 'y': 4})
573. # 4:10
574. aryT.append({'x': [
575. 1, 0, 0, 1, 0, 0, 0, 0,
576. 1, 0, 0, 1, 0, 0, 0, 0,
577. 1, 0, 0, 1, 0, 0, 0, 0,
578. 1, 0, 0, 1, 0, 0, 0, 0,
579. 1, 1, 1, 1, 1, 1, 1, 0,
580. 0, 0, 0, 1, 0, 0, 0, 0,
581. 0, 0, 0, 1, 0, 0, 0, 0,
582. 0, 0, 0, 1, 0, 0, 0, 0,
583. ], 'y': 4})


587. # No. of Training Examples
588. intM = len(aryT)
590. # Learning Rate
591. intAlpha = 0.1
593. # Init Array Parameters
594. aryTheta = []
595. **for** i **in** range(0, intNumOfDigits):
596. aryTheta.append([
597. 0, 0, 0, 0, 0, 0, 0, 0,
598. 0, 0, 0, 0, 0, 0, 0, 0,
599. 0, 0, 0, 0, 0, 0, 0, 0,
600. 0, 0, 0, 0, 0, 0, 0, 0,
601. 0, 0, 0, 0, 0, 0, 0, 0,
602. 0, 0, 0, 0, 0, 0, 0, 0,
603. 0, 0, 0, 0, 0, 0, 0, 0,
604. 0, 0, 0, 0, 0, 0, 0, 0,
605. ])
607. # Hypothesis H, t stands for theta
608. # G(x) = t0 x0 + t1 x1 + ... +
609. # H(z) = 1 / (1 + e^(-z))
611. intTrainTimes = 100
613. # Iterates for Each Digits
614. **for** u **in** range(intNumOfDigits):
615. # Iterates for intTrainTimes
616. **for** t **in** range(intTrainTimes):
618. # Iterates for each feature
619. **for** i **in** range(intN):
621. # Sum of error between Y and Hypothesis
622. intSum = 0
624. # Iterates for each training example
625. **for** j **in** range(intM):
627. intTemp1 = 0
628. intTemp2 = 0
630. # Sum up to get Theta transposed X
631. **for** z **in** range(intN):
632. intTemp1 += aryTheta[u][z] \* aryT[j]['x'][z]
634. # Error between Y and Hypothesis in jth training example
636. # Get Y
637. intTempY = 0
638. **if** (u == aryT[j]['y']):
639. intTempY = 1
640. **else**:
641. intTempY = 0
643. intTemp2 = intTempY - (1 / (1 + math.exp(-intTemp1)))
644. intTemp2 = intTemp2 \* aryT[j]['x'][i]
645. intSum += intTemp2
647. # Update Theta i
648. aryTheta[u][i] = aryTheta[u][i] + intAlpha \* intSum
650. # Print all Theta(s)
651. # print('aryTheta: ' + str(aryTheta))
653. # Testing
655. # Array which stores testing values
656. aryTest = [
657. 0, 0, 1, 0, 0, 0, 1, 0,
658. 0, 1, 0, 0, 0, 0, 1, 0,
659. 1, 0, 0, 0, 0, 0, 1, 0,
660. 1, 1, 1, 1, 1, 1, 1, 0,
661. 0, 0, 0, 0, 0, 0, 1, 0,
662. 0, 0, 0, 0, 0, 0, 1, 0,
663. 0, 0, 0, 0, 0, 0, 1, 0,
664. 0, 0, 0, 0, 0, 0, 1, 0,
665. ]
667. intTemp1 = 0
668. aryAnswer = []
669. intAnswer = 0
670. intU = 0
672. **for** u **in** range(0, intNumOfDigits):
673. intTemp1 = 0
674. **for** i **in** range(0, intN):
675. intTemp1 += aryTheta[u][i] \* aryTest[i]
677. # The output of the hypothesis
678. aryAnswer.append(1 / (1 + math.exp(-intTemp1)))
680. # Print the output given by the hypothesis
681. **print**("Probability of Getting " + str(u) + " : " + str(aryAnswer[u]))
683. **if** (aryAnswer[u] > intAnswer):
684. intAnswer = aryAnswer[u]
685. intU = u
687. **print**('I guess the Answer is: ' + str(intU))

Program: ML0502.py

After running the program, the output is:

1. Probability of Getting 0 : 0.0016814319429384072
2. Probability of Getting 1 : 9.794016051396545e-06
3. Probability of Getting 2 : 0.00012834724012942346
4. Probability of Getting 3 : 0.004065269658928395
5. Probability of Getting 4 : 0.9996635676653031
6. I guess the Answer **is**: 4

So, as expected, the answer is 4!

To explain what is going on inside this program, I would like to use the following table to visualize the data stored in the training example array aryT, each ‘Dot’ in the pictures represent a ‘1’ in the particular position of the array.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Digits** | **Pictures** | | | | |
| 0 | 图片包含 游戏机, 桌子  描述已自动生成 | 图片包含 游戏机, 桌子  描述已自动生成 | 图片包含 游戏机, 桌子, 华美, 飞行  描述已自动生成 | 图片包含 游戏机, 桌子  描述已自动生成 | 图片包含 游戏机, 桌子  描述已自动生成 |
| 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机, 电脑, 桌子, 食物  描述已自动生成 | 图片包含 游戏机, 电脑, 桌子, 灯光  描述已自动生成 | 图片包含 游戏机, 电脑, 桌子, 食物  描述已自动生成 | 图片包含 游戏机, 桌子, 华美, 飞行  描述已自动生成 |
| 1 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机  描述已自动生成 |
| 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机, 过滤网  描述已自动生成 | 图片包含 游戏机, 电脑, 彩色, 桌子  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机  描述已自动生成 |
| 2 | 图片包含 游戏机, 飞行, 桌子, 雨  描述已自动生成 | 图片包含 游戏机, 桌子, 飞行  描述已自动生成 | 图片包含 游戏机, 桌子  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机, 桌子, 电脑  描述已自动生成 |
| 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机, 桌子, 电脑  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 飞行, 游戏机, 华美, 大  描述已自动生成 | 图片包含 飞行, 华美, 桌子, 游戏机  描述已自动生成 |
| 3 | 图片包含 游戏机, 飞行, 雨  描述已自动生成 | 图片包含 游戏机, 雨  描述已自动生成 | 图片包含 游戏机, 飞行, 雨  描述已自动生成 | 图片包含 游戏机, 雨, 飞行  描述已自动生成 | 图片包含 游戏机, 桌子, 一群  描述已自动生成 |
| 图片包含 游戏机, 飞行  描述已自动生成 | 图片包含 游戏机, 飞行  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机, 飞行, 华美, 桌子  描述已自动生成 | 图片包含 飞行, 游戏机, 华美, 大  描述已自动生成 |
| 4 | 图片包含 游戏机, 食物  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机, 桌子  描述已自动生成 | 图片包含 游戏机, 桌子, 一群  描述已自动生成 | 图片包含 游戏机  描述已自动生成 |
| 图片包含 游戏机, 桌子, 一群  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机  描述已自动生成 | 图片包含 游戏机, 桌子  描述已自动生成 | 图片包含 游戏机, 桌子  描述已自动生成 |

Table 5.2.1

After using the learning algorithm “Logistic Regression” to learn the above training examples, we use the array aryTest to store the “dots” for the target to be tested:

|  |  |
| --- | --- |
| aryTest | 图片包含 游戏机  描述已自动生成 |

Table 5.2.2

And then we try to determine the “possibility” of aryTest being 0, 1, 2, 3, 4 respectively, and set the highest one as the result, which is a “4” as expected.

It is very important to note that in this example, the aryTest is not the same as one the picture of digit 4 in Table 5.2.1, but the “shape” of the dots in aryTest is somehow, in the sense of human being, similar to those pictures in the row of digit 4, this is how we can distinguish digits using the method of machine learning, instead of explicitly “program” the “shape” of each alphabet (by using Non-Machine Learning programming techniques, which is actually nearly impossible to do so).

**Try Your Own Tests:**

1. Try to modify the array aryTest, so that it “looks like” any digit from 0 to 4, and see whether our program can recognize this digit correctly or not.
2. Add some training examples for digits 5 to 9 (Pls. also remember to change the variable intNumOfDigits to 10), and see whether the program can recognize any digits from 0 to 9.
3. You may even add some training examples for alphabets from “a” to “z” (And from “A” to “Z” as well), and symbols (such as “!@#$%^&\*()” etc., to see whether our program can recognize them.

You should now discover the power of Machine Learning after understanding the python program ML0502.py, we’ll talk about more Learning Algorithm in our later chapters for solving other problems by using Machine Learning Algorithms, and its implementation in Python.

# 6 Newton’s Method

## 6.1 Another Method for Finding Thetas

Newton’s method, which converges more quickly and accurately, is another method for finding s. Below figure shows how Newton’s method works:

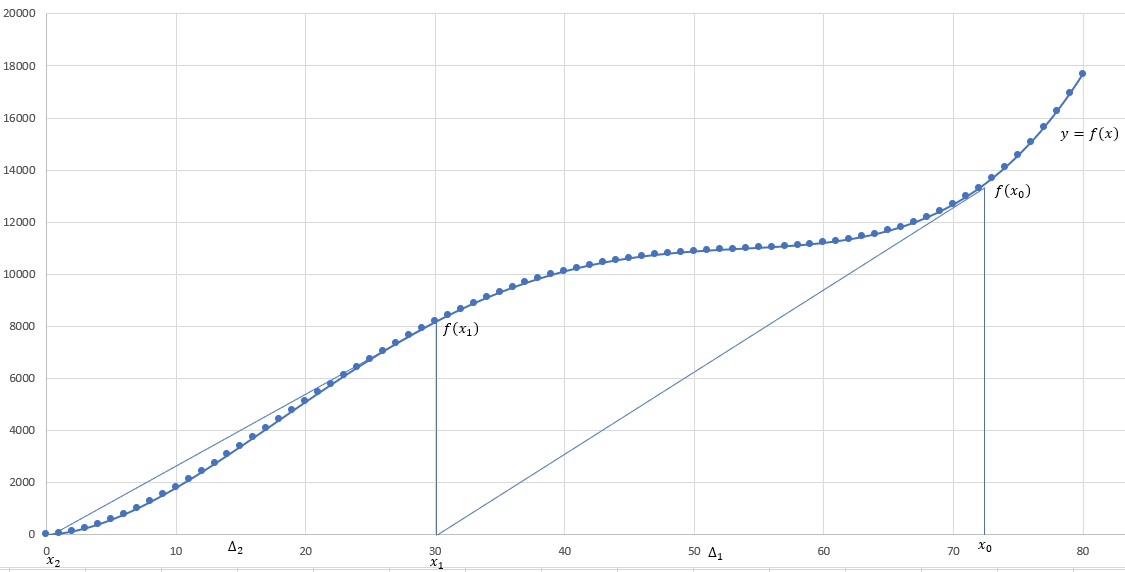


Figure 6.1.1

For a curve ,

First, we have a random value x0, then we find .

Second, we find the slope of at point , then we can find x1, where the slope from x1 to is equal to the slope of at point . And this is the first iteration of Newton’s method.

We repeat the first second step for x1…n, and will be closer to 0 as more iterations are done.

That is how Newton’s method works.

In Figure 6.1.1, the distance between x0 and x1 is (Delta), Delta means difference.

We know that , so we can find by the following formula:

In Newton’s method, for all iterations:

Where xt represents the current x, and xt+1 refers to the new x calculated in 1 iteration.

Now we know how Newton’s method works and its formula.

So, what is ? Actually is just , the log likelihood of . It is a quadratic curve which will only have a maximum point. And we can apply Newton’s method to maximize the likelihood of just like what Gradient Descent or Gradient Ascent does.

For all iterations using Newton’s method in maximizing likelihood:

What the above formula does is just and .

Then, we get a nice-looking formula which can also maximize the log likelihood of .

## 6.2 Python Implementation of Newton’s Method

Below is a program using Newton’s Method, similar to Program ML0501.py, to find out a curve which fits the data in Table 3.1.1:

1. # In this Machine Learning Example, we try to use Newton's Method
2. # to draw a curve, with given weights and heights of people
4. **import** math
5. **from** matplotlib **import** pyplot as plt
7. # Define No. of Features of x
8. intN = 3
10. # Define Training Examples
11. aryT = []
12. aryT.append({'x': [10], 'y': 0.5})
13. aryT.append({'x': [20], 'y': 1.5})
14. aryT.append({'x': [40], 'y': 3.5})
15. aryT.append({'x': [60], 'y': 8})
16. aryT.append({'x': [80], 'y': 12})
17. aryT.append({'x': [100], 'y': 20})
18. aryT.append({'x': [120], 'y': 30})
19. aryT.append({'x': [140], 'y': 40})
20. aryT.append({'x': [160], 'y': 55})
21. aryT.append({'x': [170], 'y': 70})
22. aryT.append({'x': [180], 'y': 76})
24. # No. of Training Examples
25. intM = len(aryT)
27. # Init Array Parameters
28. aryTheta = [
29. 0,
30. 0,
31. 0,
32. 0,
33. ]
35. # Init maximum training times
36. intMaxTrainTimes = 10000


40. # Iterates for intMaxTrainTimes
41. **for** t **in** range(intMaxTrainTimes):
43. # Iterate for each feature
44. **for** i **in** range(0, intN+1):
45. # Init Summation
46. intSum = 0
47. intSum2 = 0
49. # Iterate for each training example
50. **for** j **in** range(intM):
52. # Init temp for calculation
53. intTemp = 0
55. # Sum up to get predicted value of the hypothesis
56. **for** n **in** range(0, intN+1):
57. **if** n > 0:
58. intTemp += aryTheta[n] \* math.pow(aryT[j]['x'][0], n)
59. **else**:
60. intTemp += aryTheta[n]
62. # Calculate Error
63. intTemp = intTemp - aryT[j]['y']
65. **if** i > 0:
66. intTemp = intTemp \* math.pow(aryT[j]['x'][0], i)
68. intSum += intTemp
70. **if** i > 0:
71. intSum2 += math.pow(aryT[j]['x'][0], 2 \* i)
72. **else**:
73. intSum2 += 1
74. #End for
76. # Calculate New Theta(0)
77. aryTheta[i] = aryTheta[i] - intSum / intSum2


81. # Print all Theta(s)
82. **print**('aryTheta[0]: ' + str(aryTheta[0]))
83. **print**('aryTheta[1]: ' + str(aryTheta[1]))
84. **print**('aryTheta[2]: ' + str(aryTheta[2]))
85. **print**('aryTheta[3]: ' + str(aryTheta[3]))


89. # Check Error
90. intSum = 0
91. **for** m **in** range(intM):
92. intTemp = 0
93. **for** n **in** range(0, intN+1):
94. **if** n > 0:
95. intTemp += aryTheta[n] \* math.pow(aryT[m]['x'][0], n)
96. **else**:
97. intTemp += aryTheta[n]
98. intSum += math.pow(intTemp - aryT[m]['y'], 2)
99. **print**("Error: " + str(intSum))


103. # Plot graph
104. aryX = []
105. aryY = []
106. aryH = []
107. **for** i **in** range(intM):
108. intTemp = 0
109. **for** j **in** range(0, intN+1):
110. **if** j > 0:
111. intTemp += aryTheta[j] \* math.pow(aryT[i]['x'][0], j)
112. **else**:
113. intTemp += aryTheta[j]
114. aryX.append(aryT[i]['x'][0])
115. aryY.append(aryT[i]['y'])
116. aryH.append(intTemp)
118. plt.plot(aryX, aryY, 'bx')
119. plt.plot(aryX, aryH, 'g')
120. plt.show()

Program: ML0601.py

The biggest difference between Newton’s method and Gradient Descent is that there is no learning rate in Newton’s method. This reduces the time for finding suitable learning rate(s) for each training sets.

## 6.3 Difference between Gradient Descent and Newton’s Method

We can compare the outputs of Program ML0501.py and ML0601.py to find out the difference between Gradient Descent and Newton’s method, since they are both using the same training set from Table 3.1.1. Below shows a statistics for iterations used and error value for both programs.

|  |  |  |
| --- | --- | --- |
| Iterations Error | Gradient Descent | Newton’s Method |
| 10 | 39.4997 | 173.9306 |
| 100 | 22.7992 | 24.4321 |
| 1000 | 22.4345 | 22.3818 |
| 10000 | 21.6119 | 21.4451 |

Table 6.3.1

From the above table, we can see that the converging speed of Newton’s Method is much quicker that that of Gradient Descent, especially from 10th iteration to 100th iteration.

In addition, it is obvious that Newton’s method is more accurate than Gradient Descent start from 1000th iteration.

It seems like Newton’s method is a lot better than Gradient Descent according to Table 6.3.1. However, there is one straight requirement for using Newton’s method: must exist. Besides, if the calculating of is very complicated, it may cost a lot more time to calculate one iteration for Newton’s method.

In conclusion, choosing a suitable method for maximizing the likelihood of is important. A wrong choice of method may lead to much more calculations and iterations or even a wrong output.

# 7 The Exponential Family and the Generalized Linear Model

## 7.1 Introduction to The Exponential Family

In the previous chapters, we have introduced the Gaussian distribution and the Bernoulli distribution for probability interpretation of linear and logistic regression. And the two distributions are distributions in the exponential family.

We say the class of distributions is in the exponential family if it can be written in the following form:

Where:

is called the natural parameter

is called the sufficient statistic. Usually,

are functions that are chosen by people.

We will write Gaussian distribution and Bernoulli distribution in the form of exponential family below.

## 7.2 Gaussian Distribution in the Exponential Family

The formula of Gaussian distribution is:

For the simplicity of calculation, we will set , which means .

Where:

## 7.3 Bernoulli Distribution in the Exponential Family

Below we will write the Bernoulli distribution in the form of the exponential family.

The formula of Bernoulli distribution will be:

Where:

Knowing that , we get:

So,

## 7.4 Introduction to Generalized Linear Models

In this section, we are going to use the exponential family to derive a generalized Linear Model (GLM).

Assume:

1. Given x, our goal is to output the expected value of : , which means we want our hypothesis to output the expected value of :
2. , where if is a vector.

## 7.5 Generalized Linear Model for Gaussian Distribution

For Gaussian Distribution, we know that y is distributed with mean and variance , which we assumed to be 1:

Where , found in section 7.2, and , what we have assumed in the above section.

So, we get:

=

Following steps in section 4.2, we will at last maximize the likelihood of , which we need to minimize:

by using gradient ascent.

And this is how we derive a generalized linear model for linear regression with Gaussian distribution.

## 7.6 Generalized Linear Model for Bernoulli Distribution

For any fixed value of x and , the hypothesis will output the expected value of y:

… (line 2)

… (line 3)

… (line 4)

… (line 5)

In line 2, the expected value of a Bernoulli random variable is just equal to the probability of y getting 1.

In line 3, the probability of y getting 1 is equal to because is the parameter of the Bernoulli distribution, which is defined in section 7.3.

In line 4, we already know that in section 7.3.

In line 5, we assume in section 7.4.

And this is how we build up a generalized linear model for logistic regression with the Bernoulli distribution.

# 8 Softmax Regression

## 8.1 Introduction to Multinomial

## 8.2 Derivation of Softmax Regression

## 8.3 Python Implementation of Softmax Regression