**Introduction to Machine Learning**

**And its Python Implementation**

Author: Li Hong Zephan

Version: 1.0

**Table of Content**

[1 Introduction to Machine Learning 3](#_Toc48139456)

[1.1 What is NOT Machine Learning 3](#_Toc48139457)

[1.2 What is Machine Learning 8](#_Toc48139458)

[2 Introduction to Gradient Descent 9](#_Toc48139459)

[2.1 Notation 9](#_Toc48139460)

[2.2 Standard Gradient Descent 13](#_Toc48139461)

[2.3 Stochastic Gradient Descent 16](#_Toc48139462)

[2.4 Mathematical Alternative of Gradient Descent 17](#_Toc48139463)

[2.5 Python Implementation of Gradient Descent 27](#_Toc48139464)

[2.6 The influence of the Learning Rate on the Convergence Rate 30](#_Toc48139465)

[2.7 Finding the Parameters using the Normal Equation 33](#_Toc48139466)

[3 Locally Weighted Regression 35](#_Toc48139467)

[3.1 Realistic Training Examples for Human Height and Weight 35](#_Toc48139468)

[3.2 Introduction to Locally Weighted Regression 36](#_Toc48139469)

# 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 is each digit 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 Notation

m = # training examples

x = “input” variables / features

y = “output” variables / “target” variables (Figures inputted by human)

(x, y) = a training example.

The jth training example = (x(j), y(j))

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!

## 2.2 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**”.

## 2.3 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.

## 2.4 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.

## 2.5 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.

## 2.6 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 2.6.1 – 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 |

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.

## 2.7 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 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.