First Practice for Machine Learning:Details about Codes

[Teacher : Dr. Ahmad Ali Abin] [Name : Hossein Simchi]
Student Number : 98443119
Computer Science and Engineering , Shahid Beheshti University of Tehran , Iran
[h.simchi@mail.sbu.ac.ir]

April 11, 2020

1 introduction

The structure of this practice is as follows :
In section [2] the first practice is explained. The data sets is explained in detail in section [3]. The code is explained in detail in section [4,5,6,7]. the Conclusion is provided in section [8].

2 FIRST PRACTICE

We produce Sixteen different types of data with Nampay Library.
For each data set, first with the help of a closed equation, we get the line for the final answer of the linear regression . Also we try to use the Gradient Descent method to optimize the response of linear regression parameters ($y = m * x + c$) by using 0.001 0.01, 0.2, 0.5, and 0.9 for different values of learning rate.
For each data set, we draw a three-dimensional surface of the error function and for each learning rate, we show the movement of the Gradient Descent method in the optimal direction on the three-dimensional shape and its two-dimensional contour.
The three error functions are used as follows :

$$MSE = \frac{1}{m}\sum(y_1 - y_2)^2 \tag{1}$$

$$MAE = \frac{1}{m}\sum|y_1 - y_2| \tag{2}$$

$$UKE = \frac{1}{m}\sum(y_1 - y_2) \tag{3}$$

we defined y_1 that is predicted output and y_2 that is desired output of each sample .

3 DETAILS ABOUT DATA SETS

We start to define our data sets that is created by NumPy library :

3.1 CIRCLE

$$
\begin{aligned}
&angle = np.arange(0, a_1, a_2)\\
&np.random.seed(1)\\
&r = a_4 + np.random.randint(0, a_3, angle.shape[0])\\
&x = r * np.cos(angle) + 4\\
&y = r * np.sin(angle) + 4
\end{aligned}
$$

The a_1 shows if we need to have a complete circle we should do it : $a_1 = 7$, and a_2 displays data scatter
The a_3 shows the amount of nested circles. For example, if we set it to 3, it draws three circles inside each other with a common center.
The value of a_4 indicates the radius of our circle. With a random r and angles , you can easily calculate the value of x and y through mathematical relations

3.2 CIRCLE INTO CRCLE

The formula used is exactly the same as the circle , section [3.1], except that we set the value of a_3 to 10.

3.3 LINE

$$
\begin{aligned}
&x = 2 * np.random.rand(a_1, 1)\\
&y = 2.5 + 4 * x + np.random.rand(a_2, 1)
\end{aligned}
$$

To draw a line, the above formula can be used. By generating the random values that we have placed equal to a_1 and a_2, we can produce the random data. In this exercise, we put this number to 1000. Using the equation $y = x$, and some numerical shifts plus the random values, the values for y can also be obtained.

3.4 MOON

Circular data that is introduced in section [3.1] can be used to generate the moon. The only difference is that we set the value of $a1$ to 3, which creates the shape of the moon

3.5 MOON 2

If we combine the circle [3.1] and moon [3.4] data together, we can complete the figure for the second moon.

3.6 SPIRAL CIRCLE

The formula is as follow :

$$th = np.linspace(0, 20, 1000)$$
$$x = a * exp(b * th) * cos(th) + 10$$
$$y = a * exp(b * th) * sin(th) + 10$$

The formula used to generate snail-shaped data is described above. a and b are random numbers and we defined th that is equal to angles.

3.7 X

If we use the formula related to section [3.3] twice, we can easily produce this data as well, With the difference that in one of the formulas related to y, multiply it by a negative one (-1) to produce the data that we want.

3.8 XOR

if we use the formula related to section [3.1] four times , we can easily produce this data as well .

3.9 OTHER DATA SETS

Based on the codes described, we generated other data for each of the defined data sets by manipulating the variables that do not need to be redefined.

4 CLOSED EQUATION

In two ways, we solve the generated data sets using linear regression.
In this section, we examine closed equation, by setting the equation to zero then we look for the coefficients of m and c. Next, we draw the line related to it.
You can see the closed equation as below :

$$y = m * x + c \tag{4}$$

The code used for this is below :

$$
\begin{aligned}
&xtest = np.array([[2],[0.1],[1.75],[1.25],[1.7],[1.2],[22.5],[27.6]])\\
&x = np.c_[np.ones((len(x1),1)),x1]\\
&zaviye = np.linalg.inv(x.T.dot(x)).dot(x.T).dot(y1)\\
&xtestb = np.c_[np.ones((len(xtest),1)),xtest]\\
&ypred = xtestb.dot(zaviye)\\
&plt.title('2DimensionalClosedEquation')\\
&plt.plot(xtest,ypred,'r')\\
&plt.show()
\end{aligned}
$$

In this code we need to define our test data ($xtest$), which is completely optional and it is used to determine the correctness of the decision boundary.
In the second part, we add a column to the values of x, which consists entirely of the number one
In order to get the angle ($zaviye$) in the third part, we get the transpose of x and multiply it in itself then we multiply it in y
Similar to the work done in the second part, in the fourth part, we add a column to the values of $xtest$ which is called $xtestb$
In the fifth part, in order to predict the location of $xtestb$ in the two-dimensional space, we have to multiply angle ($zaviye$) by $xtestb$ which is called $ypred$.
And finally, you can draw a line using matplotlib's library.

5 GRADIENT DESCENT METHOD

Based on the equation in (4), In each iteration we update m and c, we expect the line drawn to have the best decision boundary. Also, the line should have the lowest error rate. In fact, at every iteration, we expect the error to be minimized to reach the local minimum.
The code used for this is below :

$loop(epochs)$:

$$
\begin{aligned}
&y_pred = m * x1 + c \\
&d_m = (2/n) * sum(x1 * (ypred - y1[i])) \\
&d_c = (2/n) * sum(ypred - y1[i]) \\
&m = m - l * d_m \\
&c = c - l * d_c \\
&M.append(m) \\
&C.append(c) \\
&plotline(ypred, x1, i)
\end{aligned}
$$

We first define a loop that draws a line for each weight update. In this exercise, we have considered *epochs* equal to 30.
In the second part, we have to update the values of m and c based on a special formula that we accept as a principle. For the first time, we consider both to be zero.
In the third part, we have to update according to the learning rate (l). If we consider the learning rate to be too small, the speed of reaching the local minimum will increase, and if we consider it too high, it will cause a lot of fluctuations around the local minimum.
In the last part, we have to save the values of m and c in a separate list each time, we use them in the next parts. As you can see, there is a function in the last line that draws the line for us in every iteration. The code for this function is as follows :

$plotline(y, datapoints, i)$:

$$
\begin{aligned}
&xvalues = datapoints \\
&yvalues = y \\
if 0 <= i < 10 : \\
&plt.plot(xvalues, yvalues, 'orange') \\
if 10 <= i < 20 : \\
&plt.plot(xvalues, yvalues, 'blue') \\
if 20 <= i < 30 : \\
&plt.plot(xvalues, yvalues, 'green')
\end{aligned}
$$

Each time the function is called, it is given three input parameters and then the line is drawn in three colors: orange, blue or green. (The colors are selected in the same way as the question).

The last line is usually selected as the best line for detecting data classification, in which case we select the last line of m ($M[29]$) and c ($C[29]$) as the optimal m and the optimal c.

6 ERROR FUNCTIONS

As described at the beginning of the practice, in exchange for all the values of m and c that are obtained, we must obtain the error between the predicted values and the desired values. using all error values and summing all the values in the list, we get the total amount of error available.

6.1 MAE

Using the formula in equation (1), the code for that is as follows :

$$MSE = (1/len(ypredd)) * (((y1[i] - ypredd[i]) * (y1[i] - ypredd[i])))$$

6.2 MAE

Using the formula in equation (2), the code for that is as follows :

$$MAE = (1/len(ypredd)) * abs(((y1[i] - ypredd[i])))$$

6.3 UKE

Using the formula in equation (3), the code for that is as follows :

$$UKE = (1/len(ypredd)) * ((y1[i] - ypredd[i]))$$

7 3 DIMENSIONAL PLOT

The code is as follows :

$draw3dplot()$:

$x11, x12 = np.meshgrid(sm, sm)$
$z11, z12 = np.meshgrid(M, C)$
$fig = plt.figure(figsize = (8, 8))$
$ax = fig.gca(projection =' 3d')$
$s = ax.plotsurface(z11, z12, x11, cmap = plt.cm.rainbow)$
$cset = ax.contour(z11, z12, x11, zdir =' z', offset = 0, cmap = plt.cm.rainbow)$
$plt.title('3DimensionalErrorFunctionwithdraw3Dimensionalcontour')$
$plt.ylabel('C : fromy_predred = x * M + C')$
$plt.xlabel('M : fromy_predred = x * M + C')$
$plt.show()$
$plt.title('2Dimensionalcontourbyshowingdirectionofconvergence')$
$plt.xlabel('M : fromy_predred = x * M + C')$
$plt.ylabel('Error')$
$plt.contourf(z11, x12, x11)$
$plt.colorbar()$
$plt.scatter(z11, x11, marker =' v', color = "yellow")$
$plt.show()$

First, we need to convert the dimensions of sm (sum of error), M (list of different m from $y = m * x + c$) and C (list of different c from $y = m * x + c$) into 2 Dimensional representations with equal number of rows and columns. For this purpose, we use *meshgrid* from NumPy library.
Then we define a three-dimensional surface, the first dimension of which is equal to M, the second dimension is equal to C, and the last dimension is equal to sm (sum of error). To draw a two-dimensional contour, we also use the *contourf* command, the x-axis of which indicates the amount of change in M, and the y-axis of which is equal to the amount of error change in each line update.

8 CONCLUSION

In this file, we tried to fully describe all the points related to the written code. The results related to the output of the code's file are described in detail in a separate file (report paper) and uploaded to the courseware. At the end, I would like to thank all the friends, especially Dr. Ahmad Ali Abin.

WISH THE BEST, HOSSEIN SIMCHI (STUDENT NO. 98443119)