Machine Learning

Single Layer Perceptron

By:

Riza Ega Satyabudhi (15/386859/PA/17055)

Department of Computer Science and Electronics

Faculty of Mathematics and Natural Sciences

Universitas Gadjah Mada

Yogyakarta

2018

Library Used

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math as mt
```

- Pandas = to read the csv file and assign it to a variable
- Numpy = to make array manipulation easier
- Matplotlib = for data visualization, specifically we use the pyplot
- Math = to access the exponent value, when calculating the sigmoid function

Preprocessing

```
dataset = pd.read_csv('irisdata.csv')

def label(row):
    if row['label'] == 'Iris-setosa':
        return 0
    else:
        return 1

dataset['class'] = dataset.apply(lambda row: label(row),axis=1)
dataset
```

	x1	x2	x3	x4	label	class
0	5.1	3.5	1.4	0.2	Iris-setosa	0
1	4.9	3.0	1.4	0.2	Iris-setosa	0
2	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5.0	3.6	1.4	0.2	Iris-setosa	0
5	5.4	3.9	1.7	0.4	Iris-setosa	0

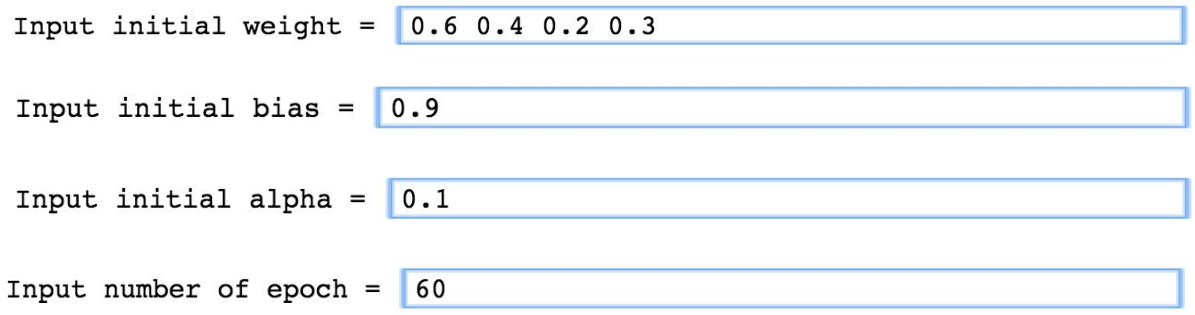
The preprocessing consists of modifying the dataset so that it has several conditions:

1. Add headers to all column (in excel)
2. Remove the Iris-Versicolor data (in excel)
3. Add new column called “class” based on the label. If the label column value is Iris-setosa, then the class value is 0. If the label value is Iris-Virginica, then the class value is 1 (in python)

Initial Variables

```
# Input initial weight, separated by whitespace (theta1, theta2, theta3, theta4)
# ex: 0.2 0.3 0.4 0.5

theta = [float(x) for x in input('Input initial weight = ').split()]
theta = np.array(theta)

# Input initial bias
# ex: 0.9
bias = [float(x) for x in input('Input initial bias = ').split()]
bias = np.array(bias)

# Input alpha
alpha = float(input('Input initial alpha = '))
alpha

# Input number of epoch
```

```
epoch = int(input('Input number of epoch = '))
epoch

error_each_epoch = np.zeros(60)
delta_theta_row = np.zeros(4)
```



We try to get the initial weight, bias, alpha, and number of epoch from the user input. The input for the weight is as the figure above, each weight is separated by a whitespace. After that we convert the weight & bias into an np array, to make us easier when doing the calculation. For the alpha, we only convert the type to float, and for the number of epoch we convert the type to int.

We create empty array of zeros (60 index, because the total number of epoch is 60) to store the error for each epoch later on and store it to a variable called *error_each_epoch*. Later after calculation has been done, this empty array will be filled with error value from each epoch

We also create an empty array of zeros (4 index, because the total number of theta is 4) to store the delta theta. Later after calculation, this empty array will be filled with the newly calculated delta theta from each iteration

Functions

1. H Function

```
def hFunc(theta,bias,r):
    return np.dot(dataset.iloc[r,:4],np.transpose(theta))+bias
```

This defines the h function for the prediction.

Parameters :

- Theta = the initially defined theta (user input) is used for the first iteration, then it is updated in each iteration
- Bias = the initially defined bias is used for the first iteration, then it is updated in each iteration
- r = the number of data row (iterator)

We use numpy to multiply the x1,x2,x3,x4 with the theta1,theta2,theta3,theta4. But we have to transpose the theta first so the matrix is in the same dimension. We then add it with the bias.

2. Sigmoid Function

```
def sigmoid(hFunc):
    return 1/(1+mt.exp(-hFunc))
```

This defines the sigmoid function for the normalization of the prediction we get from the h function.

Parameter:

- hFunc = the value of prediction we get from the h function

We also use mt.exp (math library) to get the constant value of exponent.

3. Error Function

```
def errorFunc(prediction,r):
    return (prediction-dataset.iloc[r,4])**2
```

This defines the error function for calculating the error.

Parameters:

- Prediction = the normalized prediction value we get from the sigmoid function
- r = the number of data row (iterator)

4. Delta Theta Function

```
def delta_theta(fact,prediction,x_row):
    return 2*(prediction-fact)*(1-prediction)*prediction*x_row
```

This defines the function to calculate delta theta.

Parameters:

- Fact = the fact class of the data (class 0 or class 1)
- Prediction = the normalized prediction value we get from the sigmoid function
- X_row = the x value from each data (eg: x1, x2, x3, x4)

5. New Theta Function

```
def new_theta(theta,alpha,delta_theta):
    return theta-alpha*delta_theta
```

This defines the function to calculate the new theta

Parameters:

- Theta = the initially defined theta (user input) is used for the first iteration, then it is updated in each iteration
- Alpha = the initially defined alpha
- Delta_theta = the delta theta value we get from the delta_theta function

6. Delta Bias Function

```
def delta_bias(fact,prediction):
    return 2*(prediction-fact)*(1-prediction)*prediction
```

This defines the function for calculating delta bias

Parameters:

- Fact = the fact class of the data (class 0 or class 1)
- Prediction = the normalized prediction value we get from the sigmoid function

7. New Bias Function

```
def new_bias(bias,alpha,delta_bias):
```

```
    return bias - alpha*delta_bias
```

This defines the function for calculating new bias

Parameters:

- Bias = the initially defined bias is used for the first iteration, then it is updated in each iteration
- Alpha = the alpha initially defined alpha
- Delta_bias = the delta bias value we get from the delta_bias function

Calculation

```
# Calculation
for e in range(epoch):
    error_sum = 0

    for r in range(len(dataset)):
        x_row = np.array(dataset.iloc[r,:4])
        fact_row = np.array(dataset.iloc[r,5])


        prediction_val = hFunc(theta,bias,r)
        sigmoid_val = sigmoid(prediction_val)

        error_val = errorFunc(sigmoid_val,r)
        error_sum += error_val


        # x = index for x1, x2, x3, x4
        for x in range(len(x_row)):
            delta_theta_row[x] = delta_theta(fact_row,sigmoid_val,x_row[x])

        # x = index for theta1, theta2, theta3, theta4
        for x in range(len(theta)):
            theta[x] = new_theta(theta[x],alpha,delta_theta_row[x])


        delta_bias_val = delta_bias(fact_row,sigmoid_val)
        bias = new_bias(bias,alpha,delta_bias_val)



    error_each_epoch[e] = error_sum
```

We run the functions we have defined inside a nested for loop. The first for loop is for the epoch number (60) , which has the iterator variable of *e*. First we declare a variable called error_sum to store the sum of error from each iteration.

After that we use nested for loop again, this time is a loop for each row in our dataset. It takes the length of our dataset as the parameter, which is 100. The iterator variable is *r*

We first assign data value of each row which is x1, x2, x3, x4 in a variable called x_row.
Then we assign the fact class for each row in a variable called fact_row,

After that we run the hFunc function, and store the value to a variable called prediction_val. Then we we run the sigmoid function (prediction_val as the parameter value) and store the value to a variable called sigmoid_val.

After that we calculate the error value using the errorFunc function, using sigmoid_val as the parameter, and store it in a variable called error_val. Next we create another variable to store sum of error value from each epoch in a variable called error_sum.

Next, we calculate the delta_theta for each theta, using for loop where the number of iteration is the length of x_row (x1,x2,x3,x4). We use the delta_theta function and store value to a variable called delta_theta_row we have defined earlier using the np.zeros.

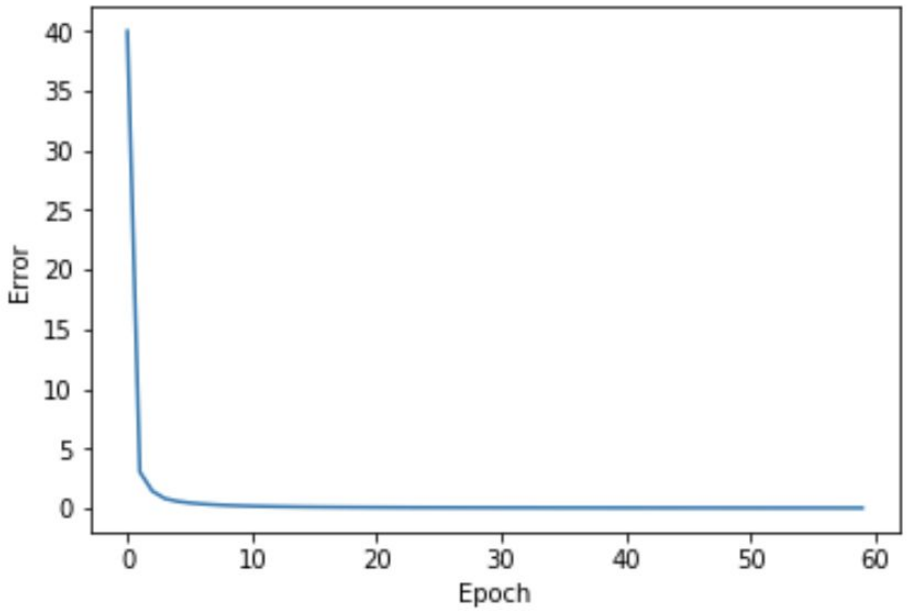
Next we calculate the new theta using the new_theta function, and store the value in a variable we have defined earlier called theta.

We calculate the delta bias using the delta_bias function and store the value in a variable called delta_bias_val. Then we update our bias using the new_bias function.

Finally, we assign the error_sum value from each epoch to error_each_epoch variable, using the epoch iterator as the index.

Visualization

```
import matplotlib.pyplot as plt

plt.plot(error_each_epoch)
plt.ylabel('Error')
plt.xlabel('Epoch')
plt.show()
```



For the visualization, we use matplotlib library, specifically the pyplot function. We take the error_each_epoch as the parameter value. Then we give label to the plot, x axis is for number of epoch, y axis is for number of error

If we print the error_each_epoch, this is the output

```
print(error_each_epoch)
```

```
[  4.00078871e+01   3.04171264e+00   1.43901571e+00   8.14508244e-01
   5.78868966e-01   4.49692564e-01   3.59762306e-01   2.94120959e-01
   2.45131888e-01   2.07964669e-01   1.79327288e-01   1.56900527e-01
   1.39041116e-01   1.24584121e-01   1.12699855e-01   1.02791711e-01
   9.44251449e-02   8.72790530e-02   8.11126126e-02   7.57425617e-02
   7.10274882e-02   6.68568365e-02   6.31431151e-02   5.98162977e-02
   5.68197481e-02   5.41072148e-02   5.16405887e-02   4.93882087e-02
   4.73235691e-02   4.54243221e-02   4.36715007e-02   4.20489086e-02
   4.05426358e-02   3.91406720e-02   3.78325955e-02   3.66093199e-02
   3.54628889e-02   3.43863063e-02   3.33733966e-02   3.24186882e-02
   3.15173167e-02   3.06649431e-02   2.98576851e-02   2.90920587e-02
   2.83649287e-02   2.76734660e-02   2.70151116e-02   2.63875449e-02
   2.57886568e-02   2.52165262e-02   2.46693994e-02   2.41456726e-02
   2.36438759e-02   2.31626600e-02   2.27007839e-02   2.22571040e-02
   2.18305654e-02   2.14201930e-02   2.10250841e-02   2.06444019e-02]
```

It is an array with a total index of 60 (1 index is for 1 epoch) , where each index has a value of sum error value at that particular epoch

The index of the array, which is from 0 to 60 is used for the x axis, while the value of error in each epoch is used for the y axis. Therefore, those variables can plot a graph

Experiment

1. **First Experiment**

For the first experiment, we use these initial variables

- Theta 1 = 0.6
- Theta 2 = 0.4
- Theta 3 = 0.2
- Theta 4 = 0.3
- Bias = 0.9
- **Alpha = 0.8**
- Epoch = 60

Following is the output of the error value of the final epoch (epoch 60)

Final epoch error value = 0.00292287380968

```
print(error_sum)
```

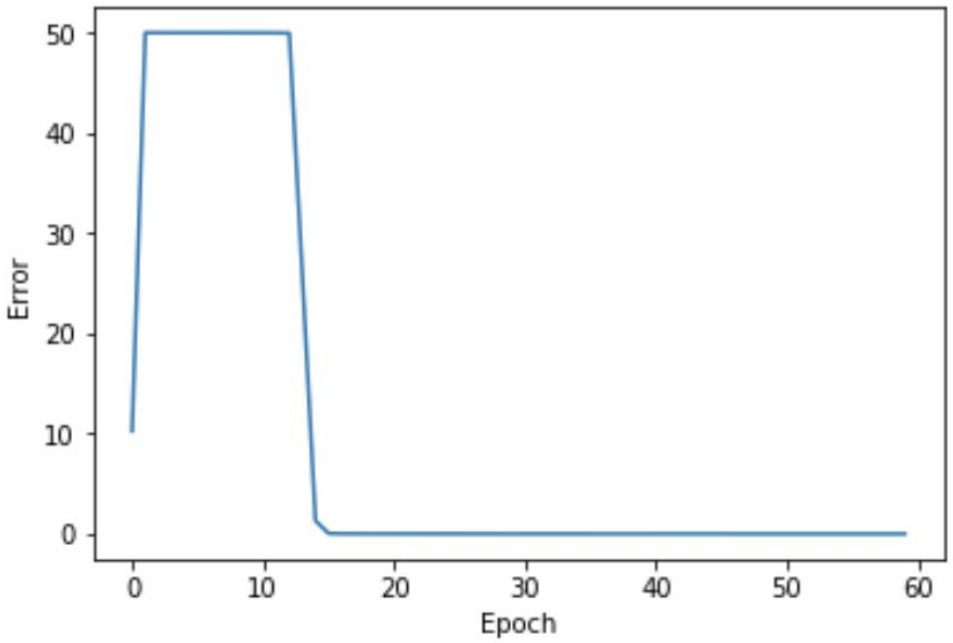
```
0.00292287380968
```

Following is the output of the error from each epoch

```
print(error_each_epoch)
```

```
[  1.02644276e+01   4.99971409e+01   4.99968824e+01   4.99965721e+01
   4.99961927e+01   4.99957183e+01   4.99951077e+01   4.99942925e+01
   4.99931481e+01   4.99914231e+01   4.99885190e+01   4.99825554e+01
   4.99622795e+01   2.57268466e+01   1.31597522e+00   5.20981875e-02
   2.00286145e-02   1.76433477e-02   1.58229568e-02   1.43268115e-02
   1.30774965e-02   1.20212648e-02   1.11182761e-02   1.03385139e-02
   9.65905729e-03   9.06217545e-03   8.53398094e-03   8.06346854e-03
   7.64182096e-03   7.26190128e-03   6.91787774e-03   6.60494284e-03
   6.31910103e-03   6.05700625e-03   5.81583655e-03   5.59319639e-03
   5.38703969e-03   5.19560889e-03   5.01738622e-03   4.85105439e-03
   4.69546480e-03   4.54961147e-03   4.41260974e-03   4.28367858e-03
   4.16212590e-03   4.04733633e-03   3.93876087e-03   3.83590825e-03
   3.73833755e-03   3.64565190e-03   3.55749316e-03   3.47353732e-03
   3.39349051e-03   3.31708563e-03   3.24407931e-03   3.17424941e-03
   3.10739271e-03   3.04332294e-03   2.98186908e-03   2.92287381e-03]
```

Following is the output of the plot



We can see that the error value increases to 50 until approximately epoch 15, then it starts decreasing to 0 afterwards.

2. Second Experiment

For the second experiment, we use these initial variables

- Theta 1 = 0.6
- Theta 2 = 0.4
- Theta 3 = 0.2
- Theta 4 = 0.3
- Bias = 0.9
- **Alpha = 0.1**
- Epoch = 60

Following is the output of the error value of the final epoch (epoch 60)

Final epoch error value = 0.0206444019205

```
print(error_sum)
```

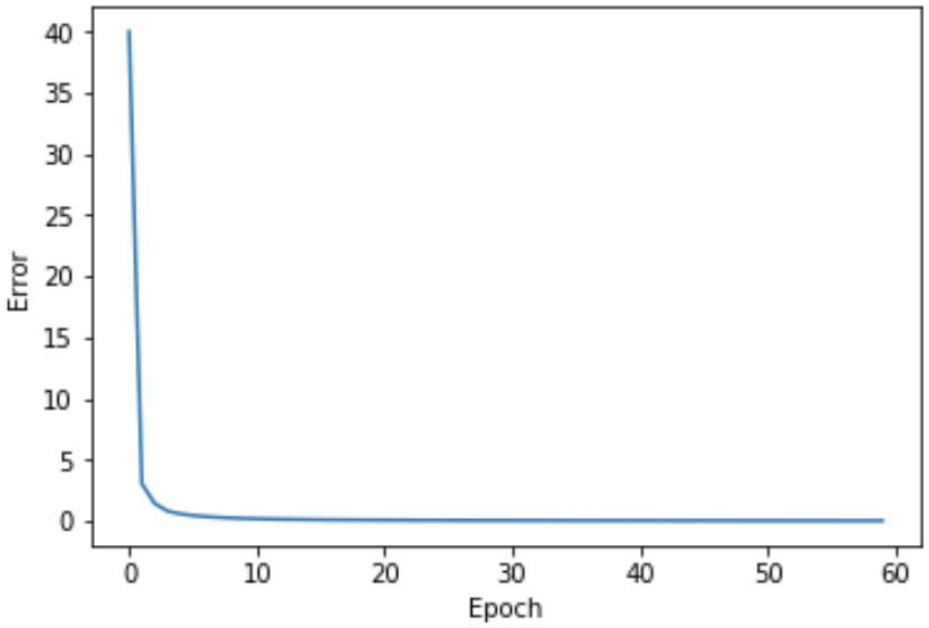
```
0.02064440192 05
```

Following is the output of the error from each epoch

```
print(error_each_epoch)
```

```
0.0206444019205
[  4.00078871e+01   3.04171264e+00   1.43901571e+00   8.14508244e-01
   5.78868966e-01   4.49692564e-01   3.59762306e-01   2.94120959e-01
   2.45131888e-01   2.07964669e-01   1.79327288e-01   1.56900527e-01
   1.39041116e-01   1.24584121e-01   1.12699855e-01   1.02791711e-01
   9.44251449e-02   8.72790530e-02   8.11126126e-02   7.57425617e-02
   7.10274882e-02   6.68568365e-02   6.31431151e-02   5.98162977e-02
   5.68197481e-02   5.41072148e-02   5.16405887e-02   4.93882087e-02
   4.73235691e-02   4.54243221e-02   4.36715007e-02   4.20489086e-02
   4.05426358e-02   3.91406720e-02   3.78325955e-02   3.66093199e-02
   3.54628889e-02   3.43863063e-02   3.33733966e-02   3.24186882e-02
   3.15173167e-02   3.06649431e-02   2.98576851e-02   2.90920587e-02
   2.83649287e-02   2.76734660e-02   2.70151116e-02   2.63875449e-02
   2.57886568e-02   2.52165262e-02   2.46693994e-02   2.41456726e-02
   2.36438759e-02   2.31626600e-02   2.27007839e-02   2.22571040e-02
   2.18305654e-02   2.14201930e-02   2.10250841e-02   2.06444019e-02]
```

Following is the output of the plot



We can see that the error value decreases immediately right from the start until it reaches 0

Conclusion

When alpha = 0.8, the error from each epoch increases at first, then it starts decreasing after approximately epoch 15. On the other hand, when alpha = 0.1, the error from each epoch immediately decreases right from the start.

Moreover, when alpha = 0.8, its error reaches near 0 after epoch 15, while for alpha = 0.1, its error reaches near 0 only after approximately epoch 2 or 3.

Based on the experiment, we can conclude that the lower the alpha value, the better the error output.

Full Code

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math as mt


dataset = pd.read_csv('irisdata.csv')
Dataset

# Add new class column

def label(row):
    if row['label'] == 'Iris-setosa':
        return 0
    else:
        return 1

dataset['class'] = dataset.apply (lambda row: label(row),axis=1)
dataset

# Initial Variables



# Input initial weight, separated by whitespace (theta1, theta2, theta3, theta4)
# ex: 0.2 0.3 0.4 0.5

theta = [float(x) for x in input('Input initial weight = ').split()]
theta = np.array(theta)


# Input initial bias
# ex: 0.9
bias = [float(x) for x in input('Input initial bias = ').split()]
bias = np.array(bias)

# Input alpha
alpha = float(input('Input initial alpha = '))
alpha

# Input number of epoch
epoch = int(input('Input number of epoch = '))
epoch
```

```
error_each_epoch = np.zeros(60)
delta_theta_row = np.zeros(4)

# Functions

def hFunc(theta,bias,r):
    return np.dot(dataset.iloc[r,:4],np.transpose(theta))+bias

def sigmoid(hFunc):
    return 1/(1+mt.exp(-hFunc))


def errorFunc(prediction,r):
    return (prediction-dataset.iloc[r,5])**2

def delta_theta(fact,prediction,x_row):
    return 2*(prediction-fact)*(1-prediction)*prediction*x_row

def new_theta(theta,alpha,delta_theta):
    return theta-alpha*delta_theta

def delta_bias(fact,prediction):
    return 2*(prediction-fact)*(1-prediction)*prediction

def new_bias(bias,alpha,delta_bias):
    return bias - alpha*delta_bias


# Calculation
for e in range(epoch):
    error_sum = 0

    for r in range(len(dataset)):
        x_row = np.array(dataset.iloc[r,:4])
        fact_row = np.array(dataset.iloc[r,5])


        prediction_val = hFunc(theta,bias,r)
        sigmoid_val = sigmoid(prediction_val)

        error_val = errorFunc(sigmoid_val,r)
        error_sum += error_val
```

```
        # x = index for x1, x2, x3, x4
        for x in range(len(x_row)):
            delta_theta_row[x] = delta_theta(fact_row,sigmoid_val,x_row[x])

        # x = index for theta1, theta2, theta3, theta4
        for x in range(len(theta)):
            theta[x] = new_theta(theta[x],alpha,delta_theta_row[x])


        delta_bias_val = delta_bias(fact_row,sigmoid_val)
        bias = new_bias(bias,alpha,delta_bias_val)



    error_each_epoch[e] = error_sum
# Visualization

import matplotlib.pyplot as plt

plt.plot(error_each_epoch)
plt.ylabel('Error')
plt.xlabel('Epoch')
plt.show()
```