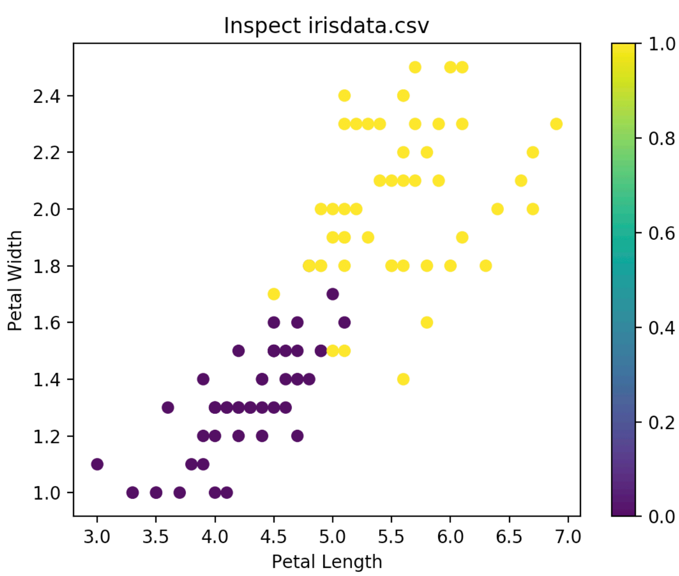
EECS 391
P3 Write-up
Qiwen Luo
Qxl216

Learning curve?
Eps -> far?

Exercise 1. Linear decision boundaries

(a)

```python
dataframe = pd.read_csv('./irisdata.csv')
two_class = dataframe[dataframe['species']!= 'setosa']
two_class.loc[two_class['species'] ==  'versicolor', 'species'] = 0
two_class.loc[two_class['species'] == 'virginica', 'species'] = 1
in_vec = two_class[['petal_length', 'petal_width']]
out_vec = two_class['species']
w1 = [1.88, 20.56]
bias = -38.994

def plot_scatter():
    plt.scatter(in_vec.values[:,0], in_vec.values[:,1], c=out_vec.values)

plot_scatter()
plt.colorbar()
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.title('Inspect irisdata.csv')
plt.show()
```

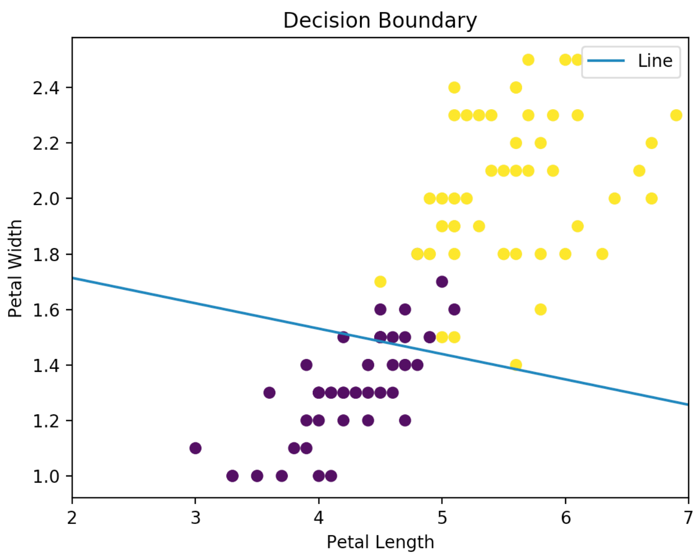


(b)

```python
def mysigmoid(z):
    result = 1.0/(1+math.exp(-z))
    return result
def output_log_single(w, b, in_petal_length, in_petal_width):
    y = mysigmoid(w[0]*in_petal_length+w[1]*in_petal_width+b)
    return y
def output_log(w, b, in_petal_length, in_petal_width):
    y = []
    for i in range(len(in_petal_length)):
        temp = output_log_single(w, b, in_petal_length.iloc[i], in_petal_width.iloc[i])
        y.append(temp)
    return y

def decision_boundary(w, b, myLabel):
    plt.xlim(2.0,7.0)
    x = np.asarray([2.0, 7.0])
    plt.plot(x,(-b-w[0]*x)/w[1], label = myLabel)

decision_boundary(w1, bias, "Line")
plt.scatter(in_vec.values[:,0], in_vec.values[:,1], c=out_vec.values)
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.title('Decision Boundary')
plt.legend()
plt.show()
```

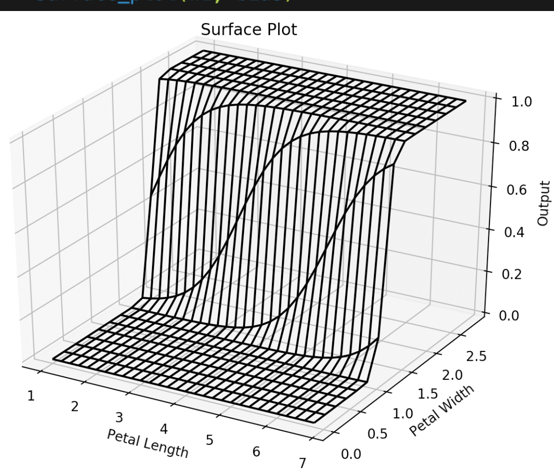
(c)


```python
def mysigmoid(z):
    result = 1.0/(1+math.exp(-z))
    return result
def output_log_single(w, b, in_petal_length, in_petal_width):
    y = mysigmoid(w[0]*in_petal_length+w[1]*in_petal_width+b)
    return y
def output_log(w, b, in_petal_length, in_petal_width):
    y = []
    for i in range(len(in_petal_length)):
        temp = output_log_single(w, b, in_petal_length.iloc[i], in_petal_width.iloc[i])
        y.append(temp)
    return y

def decision_boundary(w, b, myLabel):
    plt.xlim(2.0,7.0)
    x = np.asarray([2.0, 7.0])
    plt.plot(x,(-b-w[0]*x)/w[1], label = myLabel)

decision_boundary(w1, bias, "Line")
plt.scatter(in_vec.values[:,0], in_vec.values[:,1], c=out_vec.values)
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.title('Decision Boundary')
plt.legend()
plt.show()
```



(d)

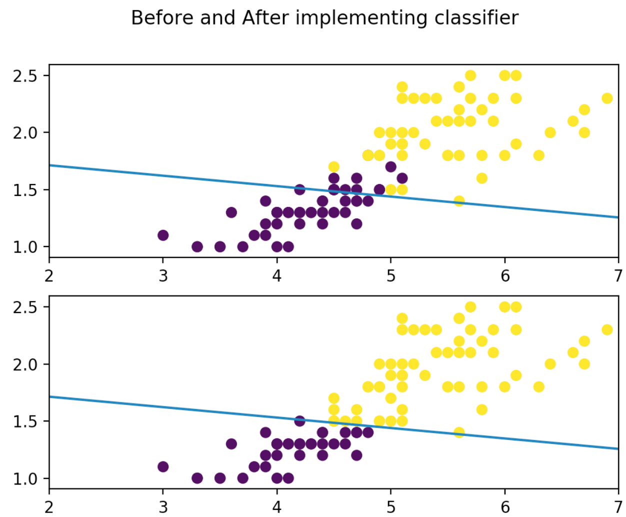
```python
def surface_plot(w, b):
    fig = plt.figure()
    ax = Axes3D(fig)
    x1 = np.arange(1, 7, 0.2)
    x2 = np.arange(0, 3, 0.2)
    X1, X2 = np.meshgrid(x1, x2)
    y = np.array([output_log_single(w, b, x1, x2) for x1, x2 in zip(np.ravel(X1), np.ravel(X2))])
    Y = y.reshape(X1.shape)
    #ax.view_init(elev=20,azim=15)
    ax.plot_wireframe(X1, X2, Y, rstride=1, cstride=1, color = 'k')
    ax.set_xlabel('Petal Length')
    ax.set_ylabel('Petal Width')
    ax.set_zlabel('Output')
    ax.set_title('Surface Plot')
    plt.show()

# ww = [1.8, 4]
# bb = -15.3
surface_plot(w1, bias)
```



(e)

```python
def round_output(input):
    if input >= 0.5:
        return 1
    else:
        return 0

print()
test1 = [6, 2.5]
print("Unambigious 3rd class")
print(round_output(output_log_single(w1, bias, test1[0], test1[1])))

test2 = [3.3, 1]
print("Unambigious 2rd class")
print(round_output(output_log_single(w1, bias, test2[0], test2[1])))

test3 = [4.9, 1.5]
print("Ambigious 2rd class")
print(round_output(output_log_single(w1, bias, test1[0], test1[1])))

test3 = [5.6, 1.4]
print("Ambigious 3rd class")
print(round_output(output_log_single(w1, bias, test1[0], test1[1])))
print()

fig, axs = plt.subplots(2)
fig.suptitle('Before and After implementing classifier')
axs[0].scatter(in_vec.values[:,0], in_vec.values[:,1], c=out_vec.values)
temp = output_log(w1, bias, two_class['petal_length'], two_class['petal_width'])
for i in range(len(temp)):
    temp[i] = round_output(temp[i])
axs[1].scatter(in_vec.values[:,0], in_vec.values[:,1], c=pd.Series(temp).values)
x = np.asarray([2.0, 7.0])
axs[0].set_xlim([2.0,7.0])
axs[0].plot(x,(-bias-w1[0]*x)/w1[1])
axs[1].set_xlim([2.0,7.0])
axs[1].plot(x,(-bias-w1[0]*x)/w1[1])
plt.show()
```

```
Unambigious 3rd class
1
Unambigious 2rd class
0
Ambigious 2rd class
1
Ambigious 3rd class
1
```



Exercise 2. Neural networks

(a)

```
def cal_mean_square(petal_lengths, petal_widths, species, w, b, pattern):
    new_y = output_log(w, b, petal_lengths, petal_widths)
    result = 0
    for i in range(len(petal_lengths)):
        result += (new_y[i] - species.iloc[i])**2
    return result/len(petal_lengths)

print("w1=%f w2=%f bias=%f"%(w1[0],w1[1],bias))
print("Mean-squared error of the iris data")
print(cal_mean_square(two_class['petal_length'], two_class['petal_width'], two_class['species'], w1, bias, ["versicolor", "virginica"])
print()
```

```
w1=1.880000 w2=20.560000 bias=-38.994000
Mean-squared error of the iris data
0.08893297501096674
```

(b)

```
w2 = [2.87, 27.56]
bias2 = -48.994
print("w1=%f w2=%f bias=%f"%(w2[0],w2[1],bias2))
print("Mean-squared error of the iris data")
print(cal_mean_square(two_class['petal_length'], two_class['petal_width'], two_class['species'], w2, bias2, ["versicolor", "virginica"]
print()

w3 = [3.67, 34.56]
bias3 = -52.78
print("w1=%f w2=%f bias=%f"%(w3[0],w3[1],bias3))
print("Mean-squared error of the iris data")
print(cal_mean_square(two_class['petal_length'], two_class['petal_width'], two_class['species'], w3, bias3, ["versicolor", "virginica"]
print()
```

```
w1=2.870000 w2=27.560000 bias=-48.994000
Mean-squared error of the iris data
0.217088014858913

w1=3.670000 w2=34.560000 bias=-52.780000
Mean-squared error of the iris data
0.40014214708135043
```

(c)For detailed steps, please refer to (d).

$$\frac{\partial MSE}{\partial w} = \frac{\partial(\frac{1}{N}\sum_{n=1}^{N}(sigmoid(wx) - y)^2)}{\partial w} = \frac{1}{N}\sum_{n=1}^{N} 2(sigmoid(wx) - y)sigmoid(wx)\big(1 - sigmoid(wx)\big)(x)$$

where $sigmoid(z) = \frac{1}{1+e^{-z}}$

(d) $MSE = \frac{1}{N}\sum_{n=1}^{N}(\frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}} - y)^2$

Scalar:

$$\frac{\partial MSE}{\partial w_1} = \frac{1}{N}\sum_{n=1}^{N} 2(\frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}} - y)\frac{-1}{(1+e^{-(w_1x_1+w_2x_2+b)})^2} e^{-(w_1x_1+w_2x_2+b)}(-x_1)$$

$$\frac{\partial MSE}{\partial w_2} = \frac{1}{N}\sum_{n=1}^{N} 2(\frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}} - y)\frac{-1}{(1+e^{-(w_1x_1+w_2x_2+b)})^2} e^{-(w_1x_1+w_2x_2+b)}(-x_2)$$

$$\frac{\partial MSE}{\partial w_i} = \frac{1}{N}\sum_{n=1}^{N} 2(\frac{1}{1 + e^{-((\sum_{i=1}^{2} w_ix_i)+b)}} - y)\frac{-1}{(1 + e^{-((\sum_{i=1}^{2} w_ix_i)+b)})^2} e^{-((\sum_{i=1}^{2} w_ix_i)+b)}(-x_i)$$

Vector:

$$\frac{\partial MSE}{\partial w} = \frac{1}{N}\sum_{n=1}^{N} 2(\frac{1}{1 + e^{-(wx+b)}} - y)\frac{-1}{(1 + e^{-(wx+b)})^2} e^{-(wx+b)}(-x)$$

(e)

```python
def cal_summed_grad(w, b):
    x1 = two_class['petal_length']
    x2 = two_class['petal_width']
    y = two_class['species']
    result = [0,0]
    for i in range(len(two_class['petal_length'])):
        temp = math.exp(-(w[0]*x1.iloc[i]+w[1]*x2.iloc[i]+b))
        result[0] += (2*(1.0/(1+temp)-y.iloc[i])*(-1.0/(1+temp)**2)*temp*(-x1.iloc[i]))
        result[1] += (2*(1.0/(1+temp)-y.iloc[i])*(-1.0/(1+temp)**2)*temp*(-x2.iloc[i]))
    return result

print("Summed gradient")
print(cal_summed_grad(w1, bias))
print()
```

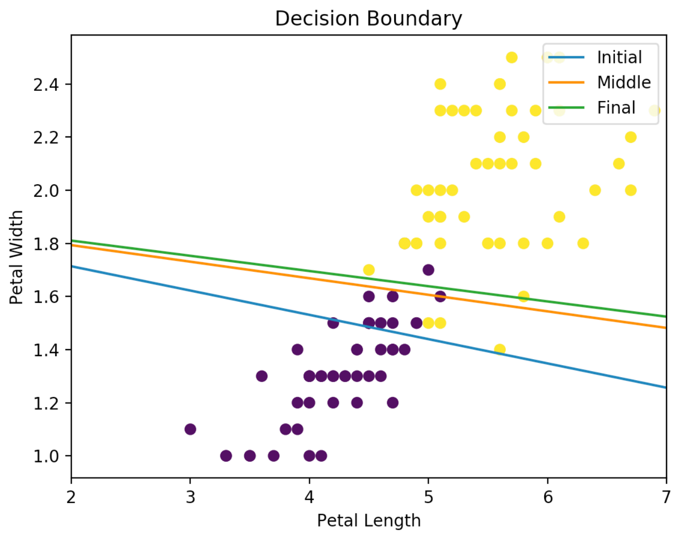
```
Summed gradient
[13.927085798657577, 4.643215588757857]
```

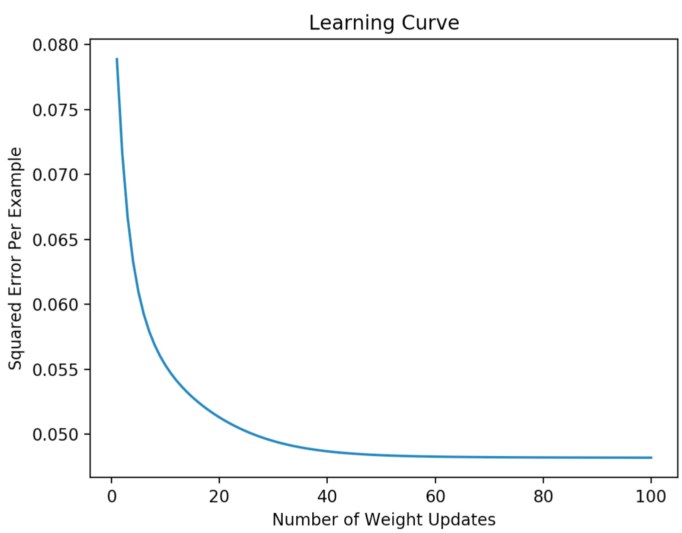
Exercise 3. Learning a decision boundary through optimization

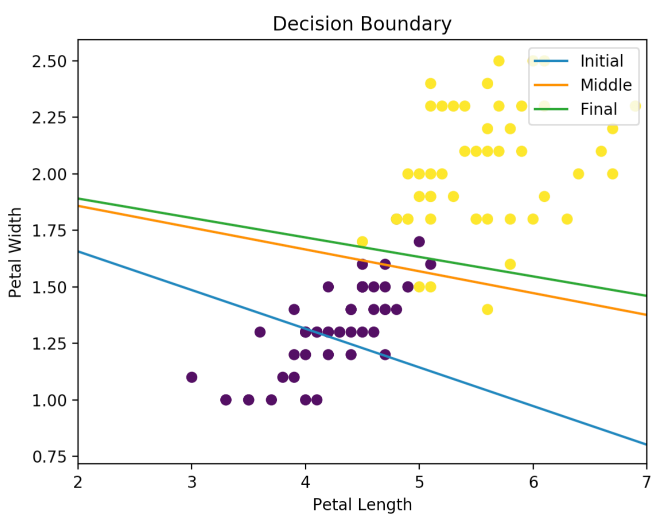
(a)(b)

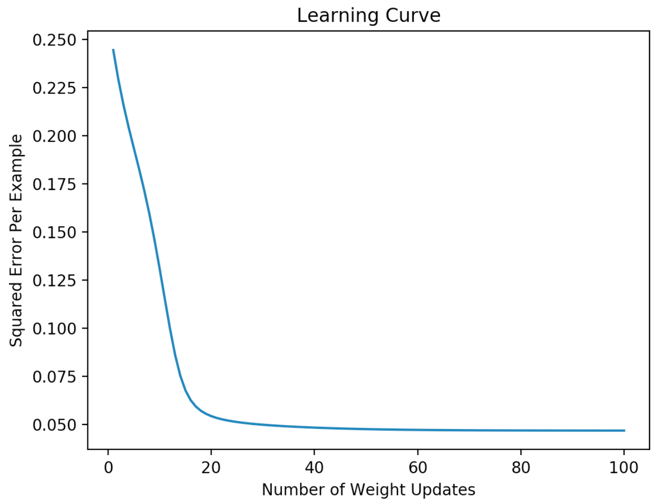
```python
def learning_curve(iter, mean_squares):
    x = list(range(1, iter+1))
    plt.plot(x,mean_squares)

def optimize_decision_boundary(w, b):
    iter = 100
    mean_squares = []
    previous = [w[0],w[1]]
    current = [w[0], w[1]]
    plot_scatter()
    decision_boundary(current, b, "Initial")
    eps = 0.5/len(two_class['petal_length'])
    for i in range(iter):
        current[0] = previous[0] - eps * cal_summed_grad(previous, b)[0]
        current[1] = previous[1] - eps * cal_summed_grad(previous, b)[1]
        previous[0] = current[0]
        previous[1] = current[1]
        mean_squares.append(cal_mean_square(two_class['petal_length'], two_class['petal_width'], two_class['species'],current, b, ["versicolor", "virginica"]))
        if i == round(iter/3, 0):
            decision_boundary(current, b, "Middle")
    decision_boundary(current, b, "Final")
    plt.xlabel('Petal Length')
    plt.ylabel('Petal Width')
    plt.title('Decision Boundary')
    plt.legend()
    plt.legend( loc = 'upper right')
    plt.show()
    learning_curve(iter, mean_squares)
    return current
```

(c)


Decision Boundary
Initial
Middle
Final
Petal Width
Petal Length


Learning Curve
Squared Error Per Example
Number of Weight Updates


Decision Boundary
Initial
Middle
Final
Petal Width
Petal Length



d) Explain how you chose the gradient step size. 5 P.

The step size that I chose is $\frac{1}{N}$. The step size is appropriate since

1) Learning rate is not too small since the gradient of the learning curve is not too small.
2) Learning rate is not too large since the learning curve continues going down instead of going up.

e) Explain how you chose a stopping criterion. 5 P.

My stopping criterion is that the weight has been updated 100 times. Originally, I considered the stopping criterion to be when MSE goes to 0 (or $\approx$ 0). But the number of iteration may be large since randomly chosen weights may not be appropriate enough. In order to shorten the running time for updating the weight, I change the stopping criterion to be updating times. According to the learning curves, the final errors both go to nearly 0. Therefore, the stopping criterion that I chose is appropriate.

Exercise 4. Extra credit: Using a deep learning toolbox

(a) Please look at the last part of P3_qxl216.py
(b)
The correction for the classification is 143/150.s

```
Epoch1: 0 | Loss1: 0.015654
Epoch1: 100 | Loss1: 0.011847
Epoch1: 200 | Loss1: 0.010292
Epoch1: 300 | Loss1: 0.009632
Epoch1: 400 | Loss1: 0.009340
Epoch1: 500 | Loss1: 0.009216
Epoch1: 600 | Loss1: 0.009174
Epoch1: 700 | Loss1: 0.009173
Epoch1: 800 | Loss1: 0.009194
Epoch1: 900 | Loss1: 0.009224
Correct classifications: 143/150
```