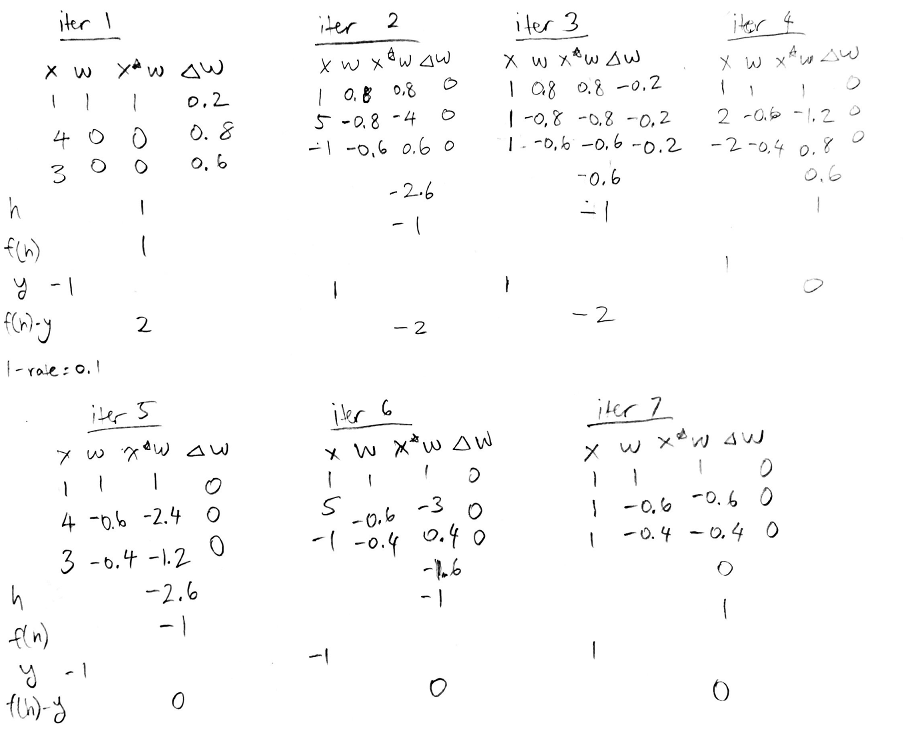
1)
a)



iter 1

	x	w	x*w	Δw
	1	1	1	0.2
	4	0	0	0.8
	3	0	0	0.6
h			1	
f(h)			1	
y	-1			
f(h)-y			2	

l-rate = 0.1

iter 2

	x	w	x*w	Δw
	1	0.8	0.8	0
	5	-0.8	-4	0
	-1	-0.6	0.6	0
h			-2.6	
f(h)			-1	
y	1			
f(h)-y			-2	

iter 3

	x	w	x*w	Δw
	1	0.8	0.8	-0.2
	1	-0.8	-0.8	-0.2
	1	-0.6	-0.6	-0.2
h			-0.6	
f(h)			-1	
y	1			
f(h)-y			-2	

iter 4

	x	w	x*w	Δw
	1	1	1	0
	2	-0.6	-1.2	0
	-2	-0.4	0.8	0
h			0.6	
f(h)			1	
y	1			
f(h)-y			0	

iter 5

	x	w	x*w	Δw
	1	1	1	0
	4	-0.6	-2.4	0
	3	-0.4	-1.2	0
h			-2.6	
f(h)			-1	
y	-1			
f(h)-y			0	

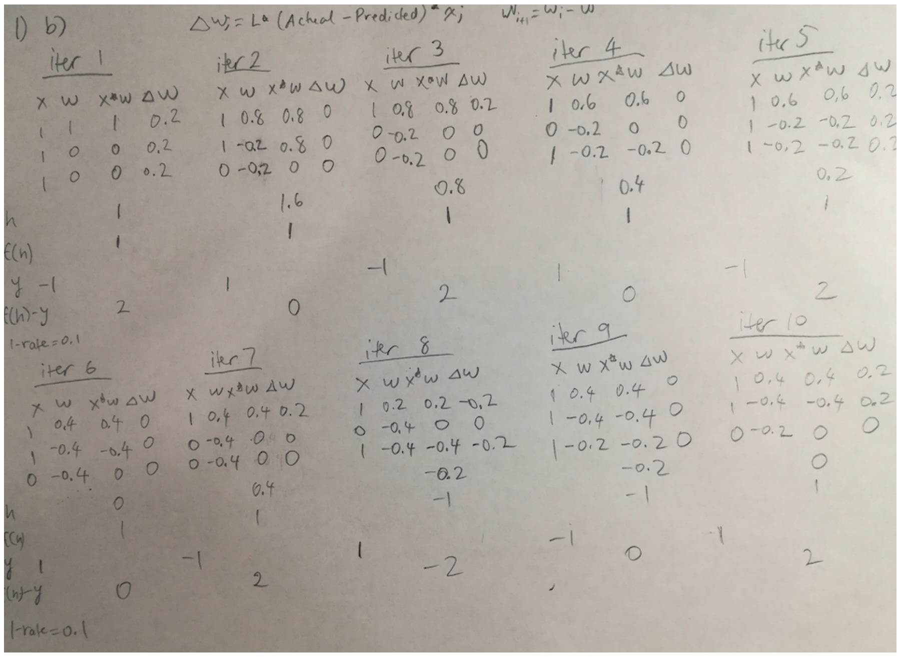
iter 6

	x	w	x*w	Δw
	1	1	1	0
	5	-0.6	-3	0
	-1	-0.4	0.4	0
h			-1.6	
f(h)			-1	
y	-1			
f(h)-y			0	

iter 7

	x	w	x*w	Δw
	1	1	1	0
	1	-0.6	-0.6	0
	1	-0.4	-0.4	0
h			0	
f(h)			1	
y	1			
f(h)-y			0	

b)
No, a single perceptron cannot solve the problem. The dataset is not linearly separable, which you can clearly see from plotting it. After 10 iterations, the does not converge. Perceptron learning algorithm can solve problems where the data set is linearly separable. It cannot solve problems where more complex boundaries are required.



1) b) $\Delta w_j = L * (\text{Actual} - \text{Predicted}) * x_j$ $W_{i+1} = W_i - W$

iter 1

	x	w	x*w	Δw
	1	1	1	0.2
	1	0	0	0.2
	1	0	0	0.2
h			1	
f(h)			1	
y	-1			
f(h)-y			2	

l-rate = 0.1

iter 2

	x	w	x*w	Δw
	1	0.8	0.8	0
	1	-0.2	0.8	0
	0	-0.2	0	0
h			1.6	
f(h)			1	
y	1			
f(h)-y			0	

iter 3

	x	w	x*w	Δw
	1	0.8	0.8	0.2
	0	-0.2	0	0
	0	-0.2	0	0
h			0.8	
f(h)			1	
y	-1			
f(h)-y			2	

iter 4

	x	w	x*w	Δw
	1	0.6	0.6	0
	0	-0.2	0	0
	1	-0.2	-0.2	0
h			0.4	
f(h)			1	
y	1			
f(h)-y			0	

iter 5

	x	w	x*w	Δw
	1	0.6	0.6	0.2
	1	-0.2	-0.2	0.2
	1	-0.2	-0.2	0.2
h			0.2	
f(h)			1	
y	-1			
f(h)-y			2	

iter 6

	x	w	x*w	Δw
	1	0.4	0.4	0
	1	-0.4	-0.4	0
	0	-0.4	0	0
h			0	
f(h)			1	
y	1			
f(h)-y			0	

l-rate = 0.1

iter 7

	x	w	x*w	Δw
	1	0.4	0.4	0.2
	0	-0.4	0	0
	0	-0.4	0	0
h			0.4	
f(h)			1	
y	-1			
f(h)-y			2	

iter 8

	x	w	x*w	Δw
	1	0.2	0.2	-0.2
	0	-0.4	0	0
	1	-0.4	-0.4	-0.2
h			-0.2	
f(h)			-1	
y	1			
f(h)-y			-2	

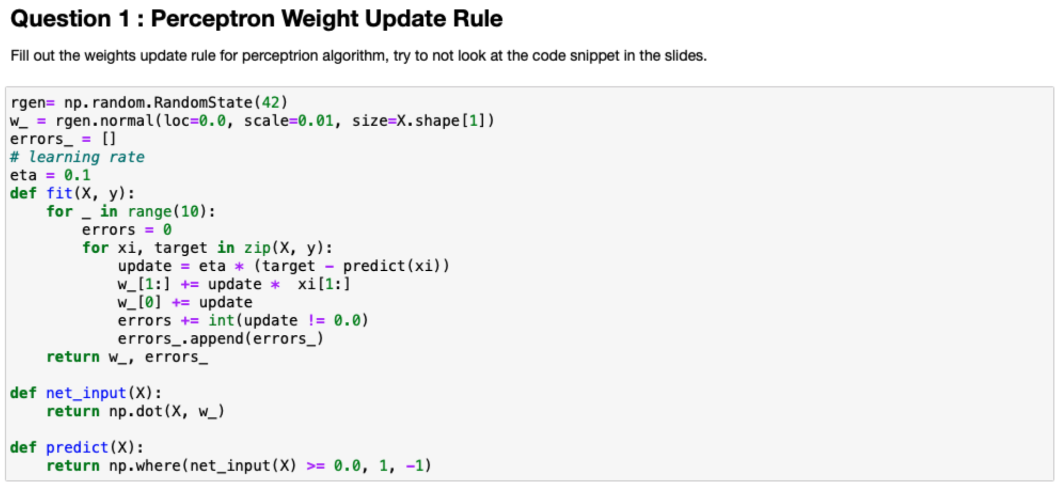
iter 9

	x	w	x*w	Δw
	1	0.4	0.4	0
	1	-0.4	-0.4	0
	1	-0.2	-0.2	0
h			-0.2	
f(h)			-1	
y	-1			
f(h)-y			0	

iter 10

	x	w	x*w	Δw
	1	0.4	0.4	0.2
	1	-0.4	-0.4	0.2
	0	-0.2	0	0
h			0	
f(h)			1	
y	-1			
f(h)-y			2	

c)
Here is the perceptron code we have to fill in.



Question 1 : Perceptron Weight Update Rule

Fill out the weights update rule for perceptrion algorithm, try to not look at the code snippet in the slides.

```
rgen= np.random.RandomState(42)
w_ = rgen.normal(loc=0.0, scale=0.01, size=X.shape[1])
errors_ = []
# learning rate
eta = 0.1
def fit(X, y):
    for _ in range(10):
        errors = 0
        for xi, target in zip(X, y):
            update = eta * (target - predict(xi))
            w_[1:] += update *  xi[1:]
            w_[0] += update
            errors += int(update != 0.0)
            errors_.append(errors_)
    return w_, errors_

def net_input(X):
    return np.dot(X, w_)

def predict(X):
    return np.where(net_input(X) >= 0.0, 1, -1)
```

2)

2) a)

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$\frac{\partial}{\partial \theta^{(i)}} J(\theta) = \frac{1}{2} \frac{\partial}{\partial \theta^{(i)}} \left(\sum_{i=1}^{m} h_\theta(x^{(i)} - y^{(i)})^2 \right.$$

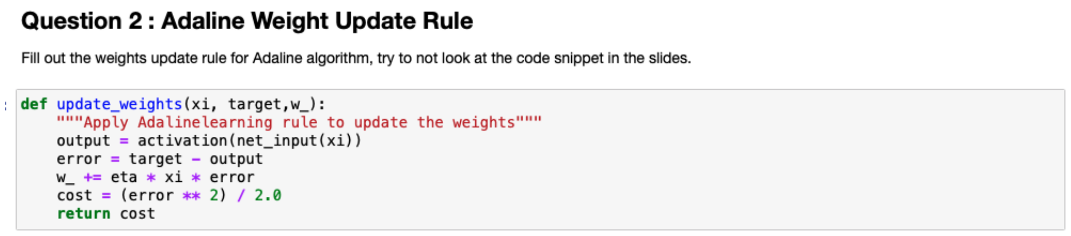
$$= \frac{1}{2} \cdot 2 \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \frac{\partial}{\partial \theta^{(i)}} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$= \frac{1}{2} \cdot 2 \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \frac{\partial}{\partial \theta^{(i)}} \left(\sum_{i=1}^{m} \theta^{(i)} x_j^{(i)} - y^{(i)} \right)$$

$$= \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

$$\theta^{(i)} = \theta^{(i)} - n \frac{\partial}{\partial \theta^{(i)}} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

b)
Here is the adaline code we have to fill in.

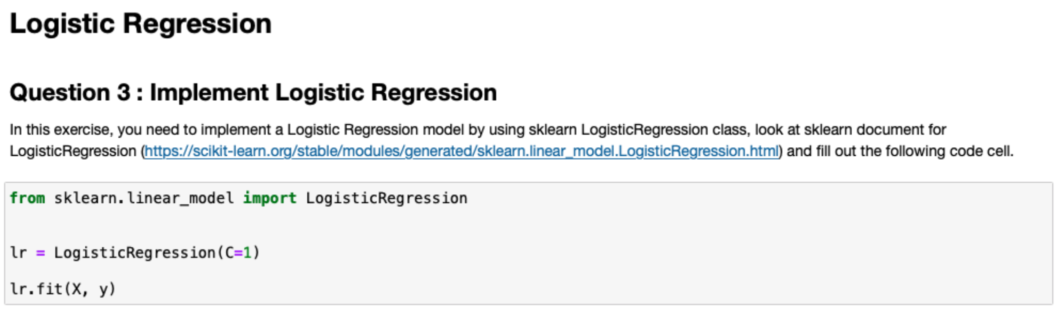


Question 2 : Adaline Weight Update Rule

Fill out the weights update rule for Adaline algorithm, try to not look at the code snippet in the slides.

```
def update_weights(xi, target,w_):
    """Apply Adalinelearning rule to update the weights"""
    output = activation(net_input(xi))
    error = target - output
    w_ += eta * xi * error
    cost = (error ** 2) / 2.0
    return cost
```

3)
a) Here is the logistic regression code we have to fill in.



Logistic Regression

Question 3 : Implement Logistic Regression

In this exercise, you need to implement a Logistic Regression model by using sklearn LogisticRegression class, look at sklearn document for LogisticRegression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) and fill out the following code cell.

```
from sklearn.linear_model import LogisticRegression


lr = LogisticRegression(C=1)

lr.fit(X, y)
```

b)
I think the best is to include all 4 features: sepal_length, sepal_width, petal_length, petal_width. That uses all the information we have, and 4 features is not that many.

4)
For the perceptron algorithm, the decision boundary was closer to the red (-1). For the adaline algorithm, the decision boundary was closer to the blue (+1). The separating line had a positive slope. The decision boundary for logistic regression had a negative slope unlike the other two. It was also slightly closer (depending on the C value) to the middle way point between the two closest -1 and +1 points.