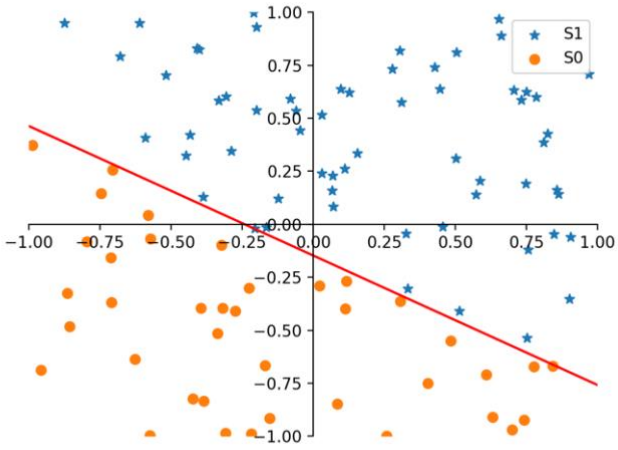
HOMEWORK 2 – REPORT

For n = 100:

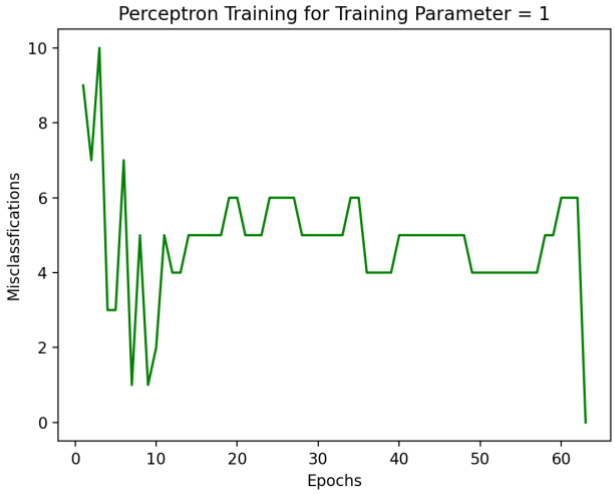
Original Random Weights:- [0.13191724929657883, 0.5465080767819135, 0.8948608097464932]

Classification of S1 and S0:



Perceptron Training Algorithm for training parameter = 1

New Random Weights: [0.6719980462439672, 0.5308896359251096, -0.5083768002064335]
Epochs = 63
Final Weights for training_param = 1 is
[1.6719980462439672, 6.337871227551445, 10.637139663524346]



How does these weights compare to the "optimal" weights [w0, w1, w2]?

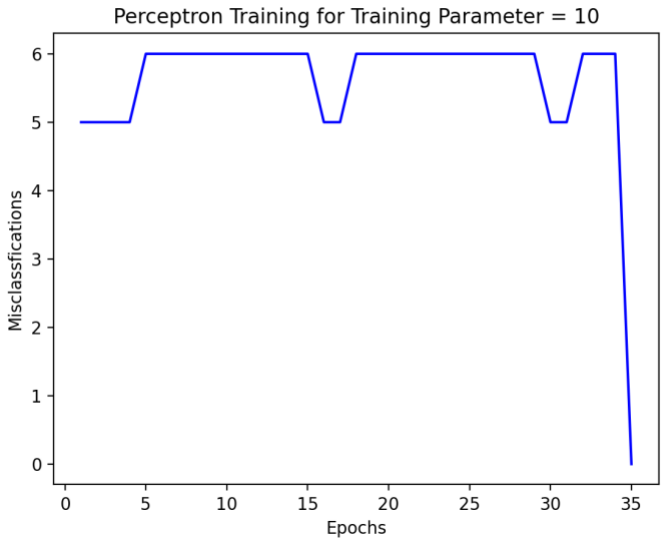
Original Random Weights:- [0.13191724929657883, 0.5465080767819135, 0.8948608097464932]

Final Weights for training_param = 1 is
[1.6719980462439672, 6.337871227551445, 10.637139663524346]

Roughly the final weights are around 12 times the original weights. Hence the ratios are more or less maintained in this case, when the training parameter is kept at 1. If we would've used a smaller training parameter, the multiplying factor would've change, but ratios would be approximately maintained.

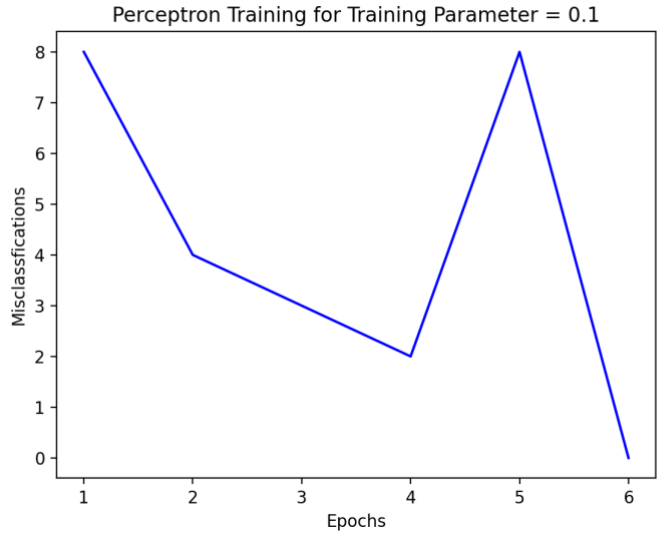
Perceptron Training Algorithm for training parameter = 10

Reinitialised Previous Random Weights: [0.6719980462439672, 0.5308896359251096, -0.5083768002064335]
Epochs = 35
Final Weights for training_param = 10 is
[10.671998046243967, 42.9303647709605, 71.0862967752567]



Perceptron Training Algorithm for training parameter = 0.1

Reinitialised Random Weights: [0.6719980462439672, 0.5308896359251096, -0.5083768002064335]
Epochs = 6
Final Weights for training_param = 0.1 is [0.07199804624396727, 0.27926643480314295, 0.4658755553636738]



Comment on how the changes in η effect the number of epochs needed until convergence.

In this case, the lower the training parameter, the lower is the number of epochs required to reach convergence. This could be due to fact that all the values are in the range –[1,1] and during the perceptron algorithm, the larger training parameter make bigger adjustments to the weights, thus at times going front and back past the point of convergence, rather than slowly reaching convergence. Thus the smallest training parameter in this case seems more effective.

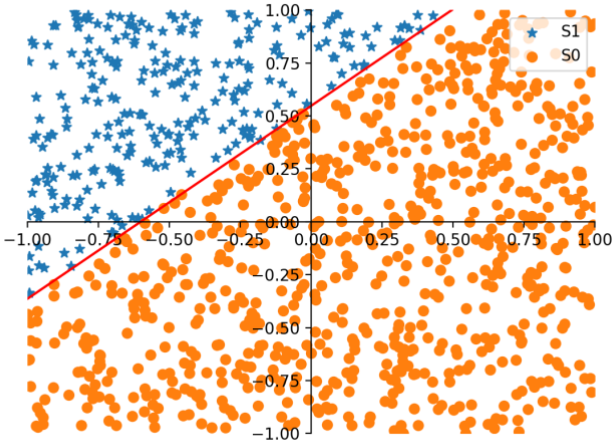
Comment on whether we would get the exact same results (in terms of the effects of η on training performance) if we had started with different w0, w1, w2, S, w0′ , w1′ , w2′ .

Not necessarily, when testing out the code, I had observed conditions where the changes from training parameter from 1 to 10, did not have any significant difference. A more intuitive example could be where if the weights [w0,w1,w2] were in a higher order like [-1000,1000], in that case an algorithm with training parameter of 0.1 could take more number of epochs to reach convergence as compared to one with training parameter of 10.

For n = 1000:

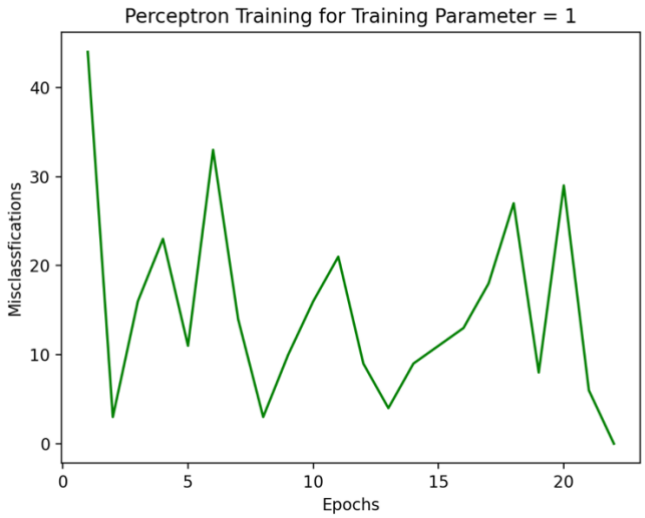
Original Random Weights:- [-0.17926687386380036, -0.2983052345726305, 0.32732333486324383]

Classification of S1 and S0:



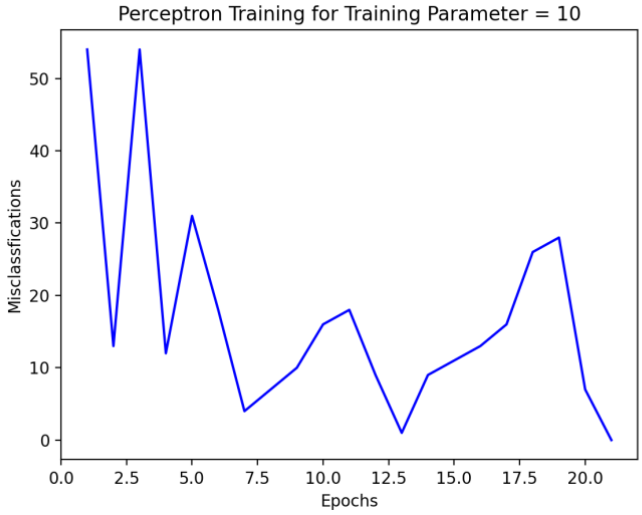
Perceptron Training Algorithm for training parameter = 1

New Random Weights: [0.016532424727276007, -0.17303866078252428, 0.8496849456169608]
Epochs = 22
Final Weights for training_param = 1 is
[-6.983467575272724, -11.505618659380849, 12.870124549232667]



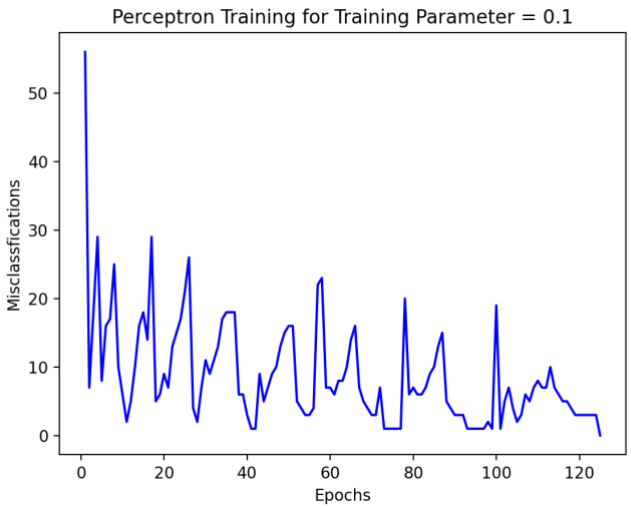
Perceptron Training Algorithm for training parameter = 10

Reinitialised Random Weights: [0.016532424727276007, -0.17303866078252428, 0.8496849456169608]
Epochs = 21
Final Weights for training_param = 10 is
[-69.98346757527273, -116.48653710852369, 128.57409253466497]



Perceptron Training Algorithm for training parameter = 0.1

Reinitialised Random Weights: [0.016532424727276007, -0.17303866078252428, 0.8496849456169608]
Epochs = 125
Final Weights for training_param = 0.1 is
[-1.283467575272724, -2.10387467788554, 2.3449366330321686]



Comment on the differences compared to n = 100.

N = 100		**N = 1000**	
$\eta = 1$	Epochs = 63	$\eta = 1$	Epochs = 22
$\eta = 10$	Epochs = 35	$\eta = 10$	Epochs = 21
$\eta = 0.1$	Epochs = 6	$\eta = 0.1$	Epochs = 125

Thus in these outputs, we see that for the smallest training_parameter the epochs taken for N=100 is much lesser than that taken for N=1000, thus we can attribute this to the difference in the number of samples and the corresponding training_parameter. For more samples, since the training algorithm with parameter = 0.1 makes only small changes, it may be find it hard to satisfy all the samples, but when increased to a certain number, it may find the suitable weights faster. We must not increase the parameter so much as to cause the algorithm to never converge.

Code:

```
import numpy as np
import random
import matplotlib.pyplot as plt

def perceptron_algo(n):
    # (a) Pick (your code should pick it) w0 uniformly at random on [− 41 , 14 ].
    w0 = random.uniform(-0.25,0.25)
    # (b) Pick w1 uniformly at random on [−1, 1].
    w1 = random.uniform(-1,1)
    # (c) Pick w2 uniformly at random on [−1, 1].
    w2 = random.uniform(-1,1)

    weights = [w0,w1,w2]
    print("Original Weights:-",weights)
    # (d) Pick n = 100 vectors x1, . . . , xn independently and uniformly at random on [−1, 1]^2, call the collection of
these vectors S.
    S = []
    for i in range(0,n):
        vector = [random.uniform(-1,1),random.uniform(-1,1)] # [x1,x2]
        S.append(vector)

    # (e) Let S1 ⊂ S denote the collection of all x = [x1 x2] ∈ S satisfying [1 x1 x2][w0 w1, w2]T ≥ 0.
    # (f) Let S0 ⊂ S denote the collection of all x = [x1 x2] ∈ S satisfying [1 x1 x2][w0 w1, w2]T < 0.

    def multiply(vector,wt):
        temp_vector = [1,vector[0],vector[1]]
        sum = 0
        for i in range(0,len(wt)):
            sum+=temp_vector[i]*wt[i]
        return sum


    S1 = []
    S0 = []
    for v in S:
        if multiply(v,weights)<0:
            S0.append(v)
        else:
```

```
        S1.append(v)

    # (g) In one plot, show the line w0 + w1x1 + w2x2 = 0, with x1 being the “x-axis” and x2 being the “y-axis.” In the same plot, show all the points in S1 and all the points in S0. Use different symbols for S0 and S1. Indicate which points belong to which class. An example figure may be as shown in Fig. 1 (My labels look bad, I expect you to do a better job!).

    # Therefore we find a line by calculating (0,x2) & (x1,0)
    p1 = [0,-(w0/w2)]
    p2 = [-(w0/w1),0]

    plt.axis([-1,1,-1,1])
    ax=plt.gca()
    ax.spines['top'].set_color('none')
    ax.spines['left'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['bottom'].set_position('zero')

    plt.axline(p1,p2,color='red')

    s1x1 = [x[0] for x in S1]
    s1x2 = [x[1] for x in S1]
    plt.scatter(s1x1,s1x2,marker='*',label='S1')
    s0x1 = [x[0] for x in S0]
    s0x2 = [x[1] for x in S0]
    plt.scatter(s0x1,s0x2,marker='o',label='S0')
    plt.legend(loc=1)
    plt.show()

    # (h) Use the perceptron training algorithm to find the weights that can separate the two classes S0 and S1 (Obviously you already know such weights, they are w0, w1 and w2 above, but we will find the weights from scratch, and the training sets S0 and S1). In detail,
    print("\n\n********* PERCEPTRON TRAINING ALGORITHM FOR TRAINING PARAMETER = 1 *********\n\n")

    # i. Use the training parameter η = 1.
    training_param = 1

    # ii. Pick w0′ , w1′ , w2′ independently and uniformly at random on [−1, 1]. Write them in your report.
    w0_t = random.uniform(-1,1)
    w1_t = random.uniform(-1,1)
```

```
    w2_t = random.uniform(-1,1)
    weights_t = [w0_t,w1_t,w2_t]
    print("New Random Weights:",weights_t)

    # iii. Record the number of misclassifications if we use the weights [w0′ , w1′ , w2′ ].
    misclassification = 0
    for test_v in S1:
        if multiply(test_v,weights_t)<0:
            misclassification+=1
    for test_v in S0:
        if multiply(test_v,weights_t)>=0:
            misclassification+=1

    print("Misclassification Before Training:",misclassification)

    # iv. After one epoch of the perceptron training algorithm, you will find a new set of weights [w0″, w1″, w2″]
    misclassification_list=[]
    for test_v in S:
        if test_v in S1 and multiply(test_v,weights_t)<0: # Step Function Condition (Implemented in a single
step,instead of taking output as 1 and 0)
            weights_t[0]+=(training_param*1)
            weights_t[1]+=(training_param*test_v[0])
            weights_t[2]+=(training_param*test_v[1])
        elif test_v in S0 and multiply(test_v,weights_t)>=0: # Step Function Condition (Implemented in a single
step,instead of taking output as 1 and 0)
            weights_t[0]-=(training_param*1)
            weights_t[1]-=(training_param*test_v[0])
            weights_t[2]-=(training_param*test_v[1])

    misclassification_new = 0
    for test_v in S1:
        if multiply(test_v,weights_t)<0:
            misclassification_new+=1
    for test_v in S0:
        if multiply(test_v,weights_t)>=0:
            misclassification_new+=1

    print("Weights after 1 epoch:",weights_t)

    # v. Record the number of misclassifications if we use the weights [w″, w″, w″].
```

```
    print("Misclassification after 1 epoch Training:",misclassification_new)
    misclassification_list.append(misclassification_new)

    # vi. Do another epoch of the perceptron training algorithm, find a new set of weights, record the number of misclassifications, and so on, until convergence.
    non_convergence = True
    epoch = 1
    while non_convergence:
        epoch+=1
        for test_v in S:
            if test_v in S1 and multiply(test_v,weights_t)<0: # Step Function Condition (Implemented in a single step,instead of taking output as 1 and 0)
                weights_t[0]+=(training_param*1)
                weights_t[1]+=(training_param*test_v[0])
                weights_t[2]+=(training_param*test_v[1])
            elif test_v in S0 and multiply(test_v,weights_t)>=0: # Step Function Condition (Implemented in a single step,instead of taking output as 1 and 0)
                weights_t[0]-=(training_param*1)
                weights_t[1]-=(training_param*test_v[0])
                weights_t[2]-=(training_param*test_v[1])

        misclassification_new = 0
        for test_v in S1:
            if multiply(test_v,weights_t)<0:
                misclassification_new+=1
        for test_v in S0:
            if multiply(test_v,weights_t)>=0:
                misclassification_new+=1

        print("Weights after",epoch,"epoch:",weights_t)
        print("Misclassification after",epoch,"epoch Training:",misclassification_new)
        misclassification_list.append(misclassification_new)
        if misclassification_new == 0:
            non_convergence = False

    # vii. Write down the final weights you obtain in your report. How does these weights compare to the “optimal” weights [w0, w1, w2]?
    print("\n\nFinal Weights for training_param =",training_param,"is\n"+str(weights_t))
```

```
    # (i) Regarding the previous step, draw a graph that shows the epoch number vs the number of misclassifications.
    epochs_1 = [i for i in range(1,epoch+1)]
    plt.title("Perceptron Training for Training Parameter = 1")
    plt.xlabel("Epochs")
    plt.ylabel("Misclassfications")
    plt.plot(epochs_1, misclassification_list, color ="green")
    plt.show()

    # (j) Repeat the same experiment with η = 10. Do not change w0, w1, w2, S, w0′ , w1′ , w2′ . As in the case η = 1, draw a graph that shows the epoch number vs the number of misclassifications.
    print("\n\n********* PERCEPTRON TRAINING ALGORITHM FOR TRAINING PARAMETER = 10 *********\n\n")
    # Use the training parameter η = 10.
    training_param = 10
    weights_t = [w0_t,w1_t,w2_t]
    misclassification_list = []
    print("Reinitialised Weights:",weights_t)
    misclassification = 0
    for test_v in S1:
        if multiply(test_v,weights_t)<0:
            misclassification+=1
    for test_v in S0:
        if multiply(test_v,weights_t)>=0:
            misclassification+=1

    print("Misclassification Before Training:",misclassification)
    non_convergence = True
    epoch = 0
    while non_convergence:
        epoch+=1
        for test_v in S:
            if test_v in S1 and multiply(test_v,weights_t)<0: # Step Function Condition (Implemented in a single step,instead of taking output as 1 and 0)
                weights_t[0]+=(training_param*1)
                weights_t[1]+=(training_param*test_v[0])
                weights_t[2]+=(training_param*test_v[1])
            elif test_v in S0 and multiply(test_v,weights_t)>=0: # Step Function Condition (Implemented in a single step,instead of taking output as 1 and 0)
                weights_t[0]-=(training_param*1)
                weights_t[1]-=(training_param*test_v[0])
```

```
            weights_t[2]-=(training_param*test_v[1])

        misclassification_new = 0
        for test_v in S1:
            if multiply(test_v,weights_t)<0:
                misclassification_new+=1
        for test_v in S0:
            if multiply(test_v,weights_t)>=0:
                misclassification_new+=1

        print("Weights after",epoch,"epoch:",weights_t)
        print("Misclassification after",epoch,"epoch Training:",misclassification_new)
        misclassification_list.append(misclassification_new)
        if misclassification_new == 0:
            non_convergence = False

    print("\n\nFinal Weights for training_param =",training_param,"is\n"+str(weights_t))
    epochs_10 = [i for i in range(1,epoch+1)]
    plt.title("Perceptron Training for Training Parameter = 10")
    plt.xlabel("Epochs")
    plt.ylabel("Misclassfications")
    plt.plot(epochs_10, misclassification_list, color ="blue")
    plt.show()

    # Repeat the same experiment with η = 0.1. Do not change w0, w1, w2, S, w0′ , w1′ , w2′ . As in the case η =
1, draw a graph that shows the epoch number vs the number of misclassifications.

    print("\n\n********* PERCEPTRON TRAINING ALGORITHM FOR TRAINING PARAMETER = 0.1 *********\n\n")
    # Use the training parameter η = 0.1
    training_param = 0.1
    weights_t = [w0_t,w1_t,w2_t]
    misclassification_list = []
    print("Reinitialised Weights:",weights_t)
    misclassification = 0
    for test_v in S1:
        if multiply(test_v,weights_t)<0:
            misclassification+=1
    for test_v in S0:
        if multiply(test_v,weights_t)>=0:
            misclassification+=1
```

```
    print("Misclassification Before Training:",misclassification)
    non_convergence = True
    epoch = 0
    while non_convergence:
        epoch+=1
        for test_v in S:
            if test_v in S1 and multiply(test_v,weights_t)<0: # Step Function Condition (Implemented in a single
step,instead of taking output as 1 and 0)
                weights_t[0]+=(training_param*1)
                weights_t[1]+=(training_param*test_v[0])
                weights_t[2]+=(training_param*test_v[1])
            elif test_v in S0 and multiply(test_v,weights_t)>=0: # Step Function Condition (Implemented in a single
step,instead of taking output as 1 and 0)
                weights_t[0]-=(training_param*1)
                weights_t[1]-=(training_param*test_v[0])
                weights_t[2]-=(training_param*test_v[1])

        misclassification_new = 0
        for test_v in S1:
            if multiply(test_v,weights_t)<0:
                misclassification_new+=1
        for test_v in S0:
            if multiply(test_v,weights_t)>=0:
                misclassification_new+=1

        print("Weights after",epoch,"epoch:",weights_t)
        print("Misclassification after",epoch,"epoch Training:",misclassification_new)
        misclassification_list.append(misclassification_new)
        if misclassification_new == 0:
            non_convergence = False
    print("\n\nFinal Weights for training_param =",training_param,"is\n"+str(weights_t))
    epochs_10 = [i for i in range(1,epoch+1)]
    plt.title("Perceptron Training for Training Parameter = 0.1")
    plt.xlabel("Epochs")
    plt.ylabel("Misclassfications")
    plt.plot(epochs_10, misclassification_list, color ="blue")
    plt.show()

perceptron_algo(100) # For n = 100
```

```
perceptron_algo(1000) # For n = 1000
```