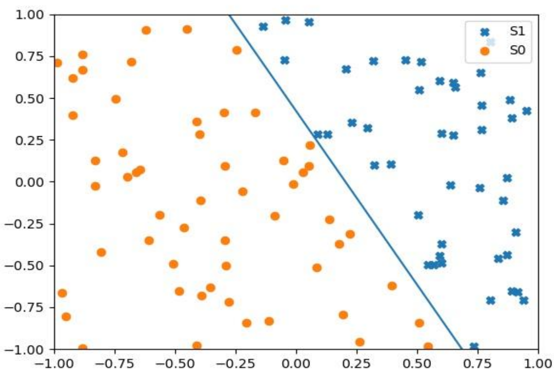
1.
Results for 100 examples:
Training examples being generated 100
optimal weights : [-0.19325576 0.94896619 0.45746927]
initial weights : [0.5674377665341095, 0.07890061529100123, 0.94470660075533]

With n = 1 the PTA algorithm converged at epoch 2 and the weights after convergence were weights at which algorithm converged [-0.43256223 2.60208545 1.11334638]

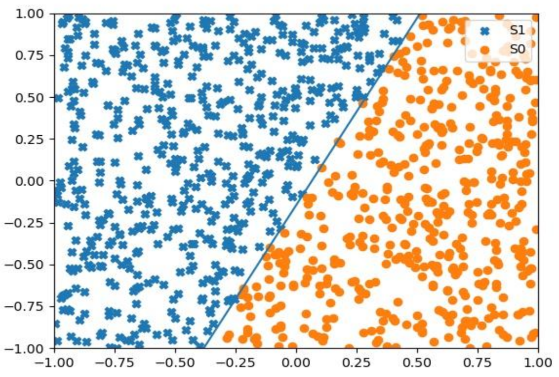
With n = 10 the PTA algorithm converged at epoch 6 and the weights after convergence were weights at which algorithm converged [-9.43256223 50.68880836 23.9399017]

With n = 0.1 the PTA algorithm converged at epoch 3 and the weights after convergence were weights at which algorithm converged [-0.13256223 0.66067771 0.34607665]

g)
Scatter plot for 100 examples



Scatter plot for 1000 examples



h)

ii) initial weights before training for 100 training examples
[0.5674377665341095, 0.07890061529100123, 0.94470660075533]

Initial weights before training for 1000 training examples
[-0.09614875306747894, 0.4274997369995819, 0.31309453273873533]

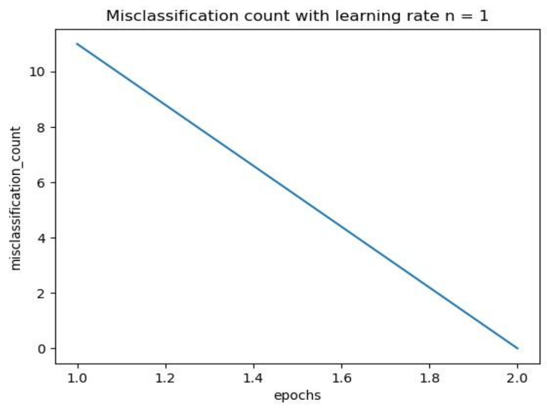
iii) Number of misclassification with initial weights [0.5674377665341095, 0.07890061529100123, 0.94470660075533] were 44 for 100 training examples

Number of misclassification with initial weights [-0.09614875306747894, 0.4274997369995819, 0.31309453273873533] were 700 for 1000 training examples

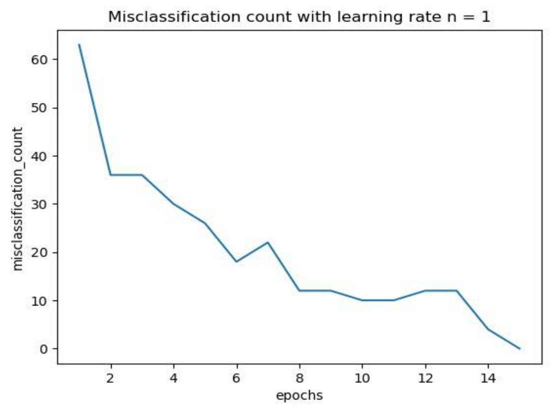
vii) Final weights after training are [-0.43256223 2.60208545 1.11334638] the signs of the weights are similar to the optimal weights. And as optimal weights the second value is the highest.

i)

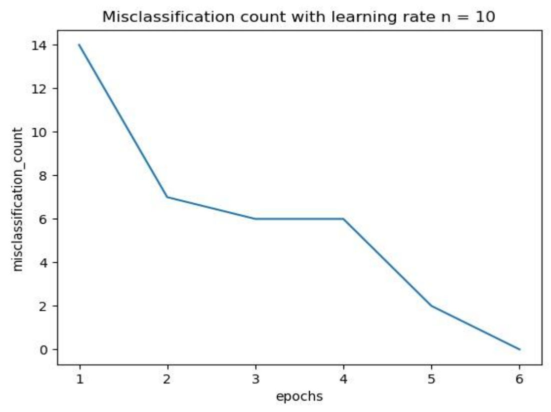
Epochs vs Misclassification for 100 training examples with learning rate 1



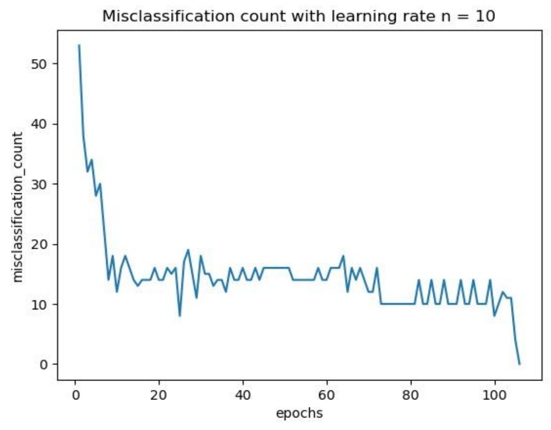
Epochs vs Misclassification for 1000 training examples with learning rate 1



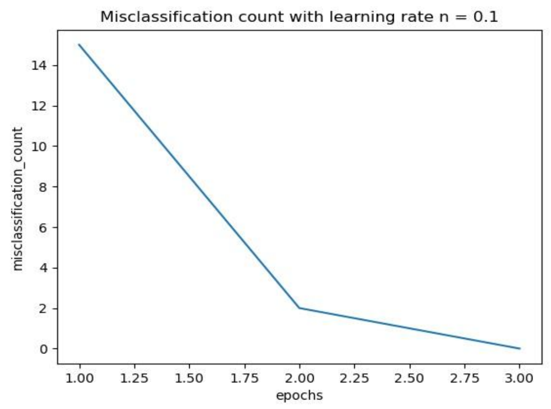
j)
Epochs vs Misclassification for 100 training examples with learning rate 10



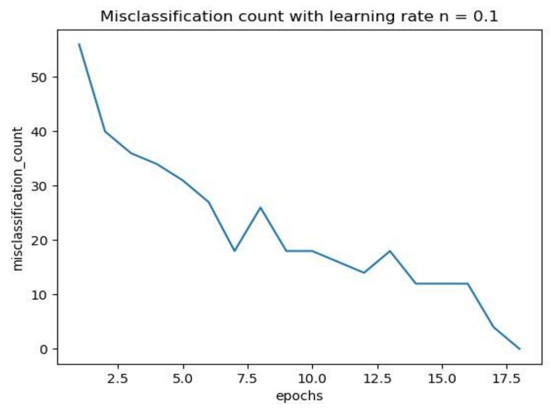
Epochs vs Misclassification for 1000 training examples with learning rate 10



k)
Epochs vs Misclassification for 100 training examples with learning rate 0.1



Epochs vs Misclassification for 1000 training examples with learning rate 0.1



l) For 100 training examples n = 1 performed the best it took just 2 epochs of training to converge. So n=1 was the ideal learning rate out of three. And for n=0.1 it took 3 epochs of training to converge so the initial weight chosen was close to weights after convergence hence n=1,0.1 took very less time. Whereas for n=10 it took a longer time to converge it took 6 epochs of training to converge. The reason may be because the learning rate was high and the weights started oscillating before reaching the weights at which misclassification count is zero.

m)
For different weights we get different performance.

The results when using different weights we can see that learning rate n=10 performed better whereas before n=1 performed better.

training examples being generated 100
optimal weights [0.19294025 0.57716313 -0.15054374]
initial weights [-0.41275095853438937, -0.6565720240152257, 0.5008266231859726]

With n = 1 the PTA algorithm converged at epoch 5 and the weights after convergence were weights at which algorithm converged [1.58724904 4.49949037 -1.21756099]

With n = 10 the PTA algorithm converged at epoch 2 and the weights after convergence were weights at which algorithm converged [9.58724904 29.07200739 -8.31139219]

With n = 0.1 the PTA algorithm converged at epoch 6 and the weights after convergence were weights at which algorithm converged [0.18724904 0.57282863 -0.14344318]

n)

Results for 1000 examples:
Training examples being generated 1000
optimal weights : [0.0495926 -0.752677   0.33668206]
initial weights : [-0.09614875306747894, 0.4274997369995819, 0.31309453273873533]

With n = 1 the PTA algorithm converged at epoch 15 and the weights after convergence were weights at which algorithm converged [0.90385125 -13.57987855   6.09551786]

With n = 10 the PTA algorithm converged at epoch 106 and the weights after convergence were weights at which algorithm converged [19.90385125 -302.0943902   135.42634291]

With n = 0.1 the PTA algorithm converged at epoch 18 and the weights after convergence were weights at which algorithm converged [0.10385125 -1.55929853 0.69277165]

For 1000 examples the convergence took 15 epochs for learning rate 1 which is the best one among the learning rates. And for learning rate 10 it took 106 epochs and by looking at the graph we can see that there were a lot of oscillations before reaching convergence.

In comparison with 100 examples, for both the best learning rate was n=1. And the behavior of both was same for different learning rates like for both n=0.1 was second best learning rate and it was very close to n=1 and n=10 took more time to converge for both of them.

Code:

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


seed = 47
np.random.seed(seed)

N = [100, 1000]
```

```
for training_examples in N:
    print("training examples being generated", training_examples)
    w0 = np.random.uniform(-0.25, 0.25)
    w1 = np.random.uniform(-1, 1)
    w2 = np.random.uniform(-1, 1)
    W = np.array([w0, w1, w2])
    print("optimal weights", W)
    X = np.random.uniform(low=[-1, -1], high=[1, 1], size=(training_examples, 2))

    S1, S0 = np.empty((0, 2)), np.empty((0, 2))
    Xy = np.empty((0, 2))

    for x in X:
        x1 = np.insert(x, 0, 1)
        if np.matmul(x1, np.transpose(W)) >= 0:
            S1 = np.append(S1, np.array([x]), axis=0)
            Xy = np.append(Xy, np.array([[x, np.array([1])]], dtype="object"), axis=0)
        else:
            S0 = np.append(S0, np.array([x]), axis=0)
            Xy = np.append(Xy, np.array([[x, np.array([0])]], dtype="object"), axis=0)

    x = np.linspace(-1, 1)
    plt.scatter(S1[:, 0], S1[:, 1], marker='X', label="S1")
    plt.scatter(S0[:, 0], S0[:, 1], marker='o', label="S0")
    plt.plot(x, -(w0+w1*x)/w2)
    plt.ylim(-1, 1)
    plt.xlim(-1, 1)
    plt.legend(loc="upper right")
    plt.savefig("scatterplot_{}.jpg".format(training_examples))
    plt.show()

    # training weights
    w01 = np.random.uniform(-1, 1)
    w11 = np.random.uniform(-1, 1)
    w21 = np.random.uniform(-1, 1)

    W1 = [w01, w11, w21]

    print("initial weights",W1)
    initial_misclassifaction_count = 0

    for x,y in Xy:
        x1 = np.insert(x, 0, 1)
        if np.matmul(x1,np.transpose(W1)) >= 0:
            if y != 1:
                initial_misclassifaction_count += 1
        else:
            if y != 0:
                initial_misclassifaction_count += 1
    print("Misclassification count before training",initial_misclassifaction_count)

    Learning_rate = [1, 10, 0.1]

    # Perceptron training algorithm for different learning rates
    for n in Learning_rate:
        W_n = W1
        epoch = 0
        converged_epoch = 0
        misclassification_count_per_epoch = []
        # used while loop as convergence epoch varies for each learning rate
        while True:
            misclassification_count = 0
            for x,y in Xy:
                x1 = np.insert(x, 0, 1)
                if np.matmul(x1,np.transpose(W_n)) >= 0:
                    if y != 1:
                        W_n = W_n - n * x1
                        misclassification_count += 1
                else:
                    if y != 0:
                        W_n = W_n + n * x1
                        misclassification_count += 1
            epoch += 1
            misclassification_count_per_epoch.append(misclassification_count)
            if misclassification_count == 0:
                converged_epoch = epoch
                print("converged at epoch", epoch)
                print("learning rate",n)
                print("weights at which algorithm converged",W_n)
                break

        epochs = list(range(1, converged_epoch+1))
        plt.plot(epochs, misclassification_count_per_epoch)
        plt.xlabel("epochs")
        plt.ylabel("misclassification_count")
        plt.title("Misclassification count with learning rate n = " + str(n) + '')
        plt.savefig("lr_{}_te_{}.jpg".format(n, training_examples))
    plt.show()
```