E9-253: Neural Networks and Learning Systems-I
HW-1

Submitted by - Ocima Kamboj
Serial Number - 06-02-01-10-51-18-1-15899

Note- The implementation of the Perceptron was done using various functions (coded in MATLAB) that have been attached in the Appendix. The respective questions contain the information about which functions were used to solve it.

Problem-4

(a) Names of Functions used - onlinePerceptron, batchPerceptron, accuracy.

AND

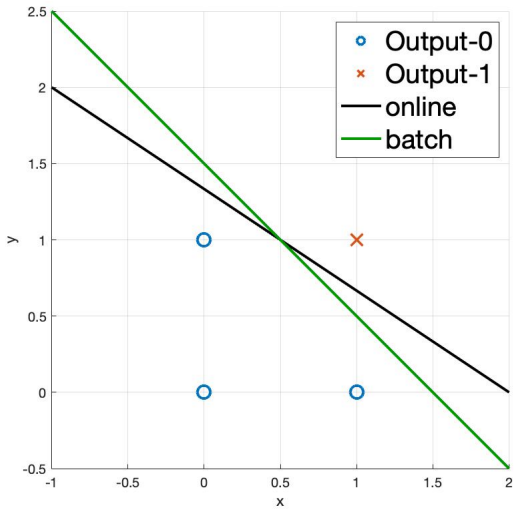

Figure 1: Implementation of AND Logic

OR

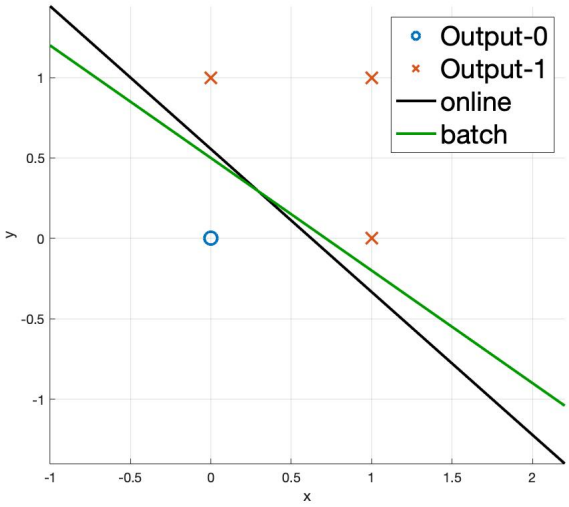

Figure 2: Implementation of OR Logic

COMPLEMENT

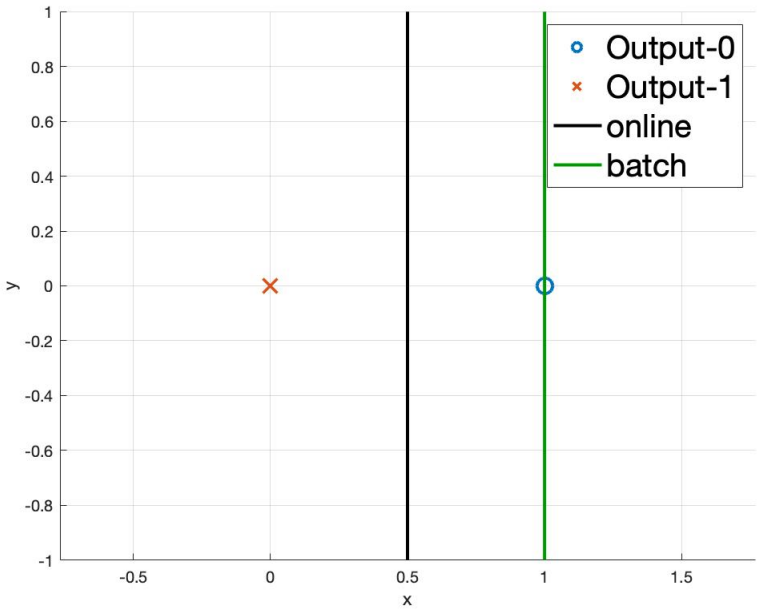

Figure 3: Implementation of COMPLEMENT Logic

(b) The Exclusive OR logic can't be implemented because it is not linearly separable.

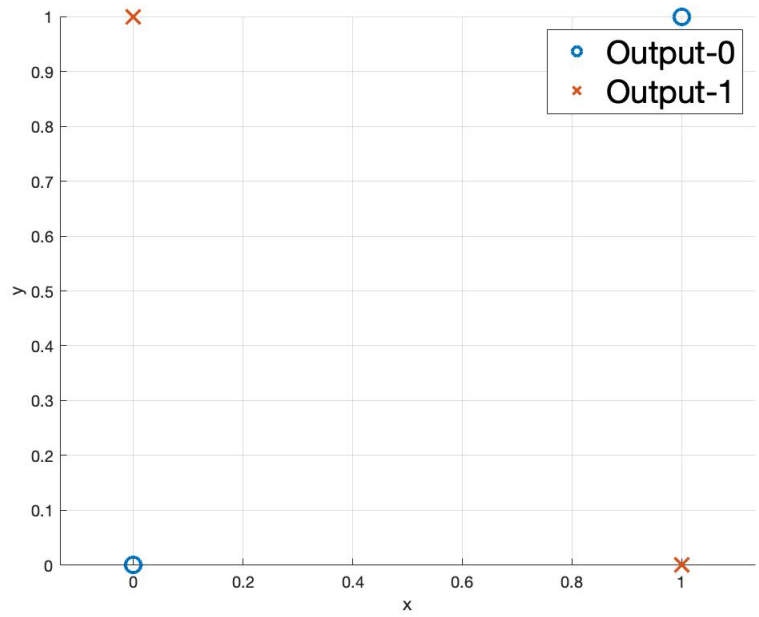

Figure 4: XOR Logic

Problem-5

(a) Name of functions used - genPoints
Radius of the innermost circle fixed for experiments = 10

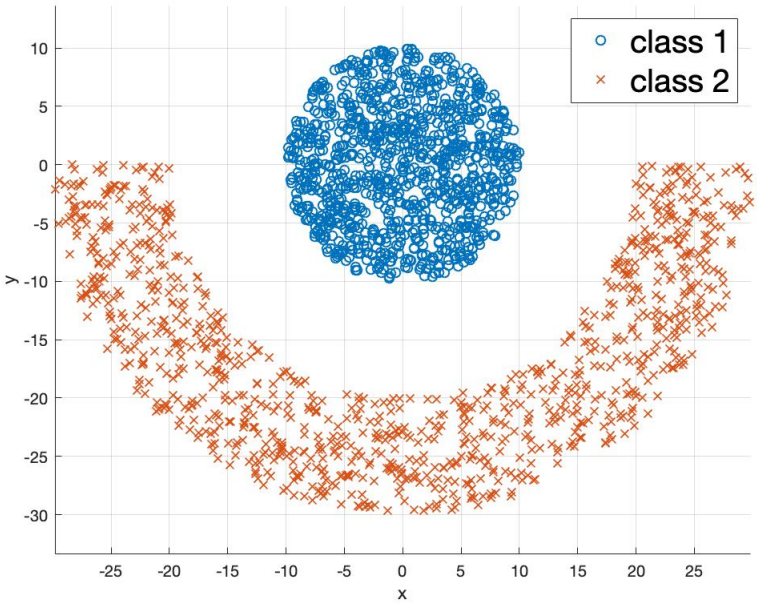

Figure 5: Input Data

(b) Online Learning Functions used - onlinePerceptron

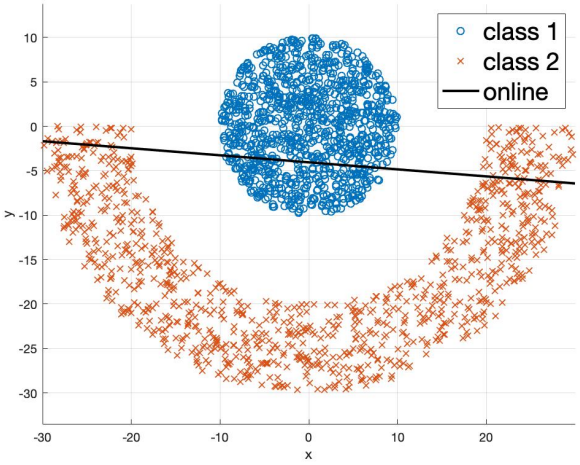

Figure 6: $D = 0$

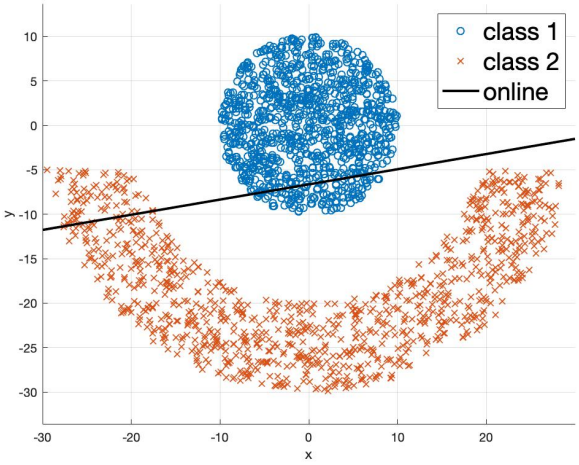

Figure 7: $D = 5$

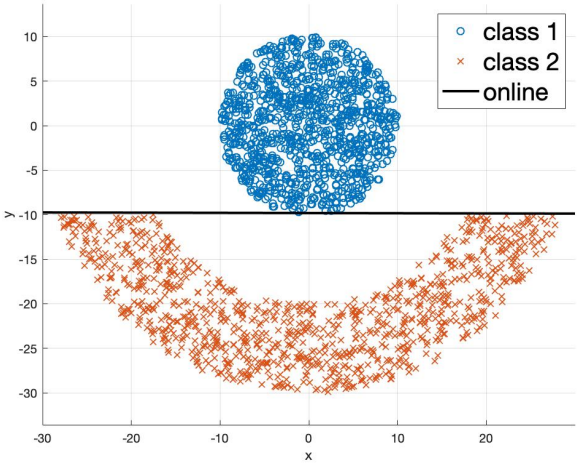

Figure 8: $D = 10$

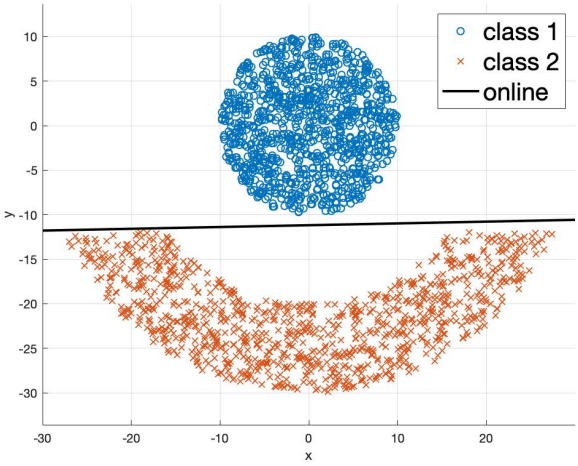

Figure 9: $D = 12$

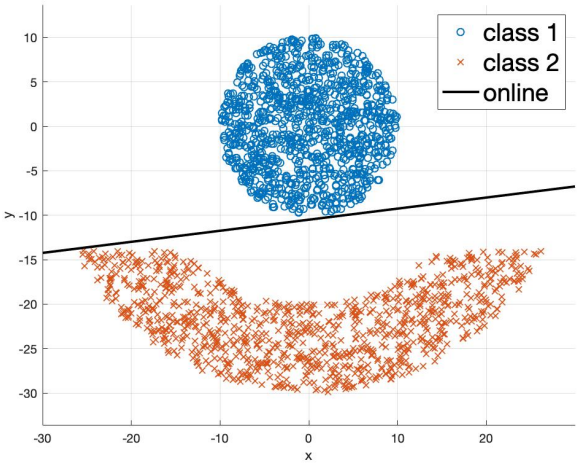

Figure 10: $D = 14$

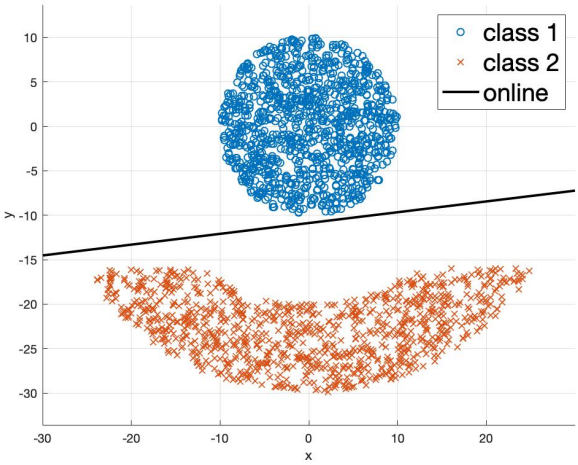

Figure 11: $D = 16$

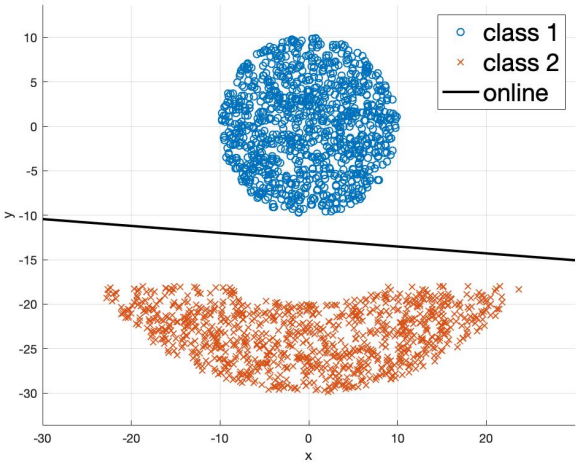

Figure 12: $D = 18$

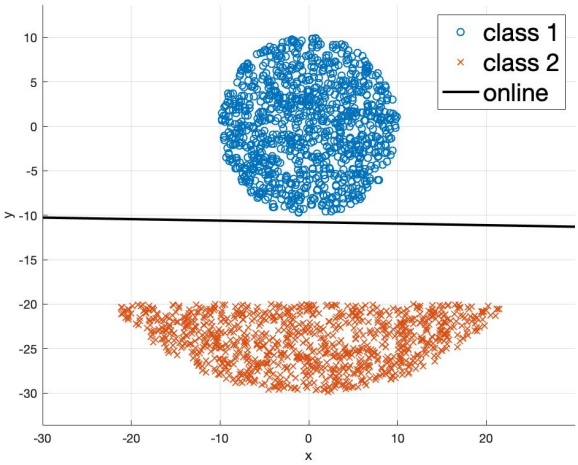

Figure 13: $D = 20$

Batch Learning Functions used - batchPerceptron

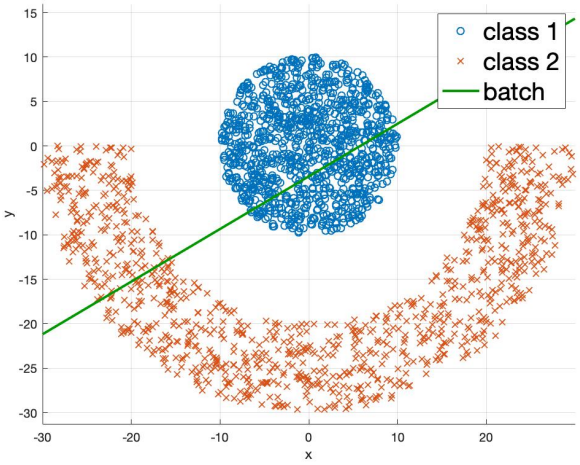

Figure 14: $D = 0$

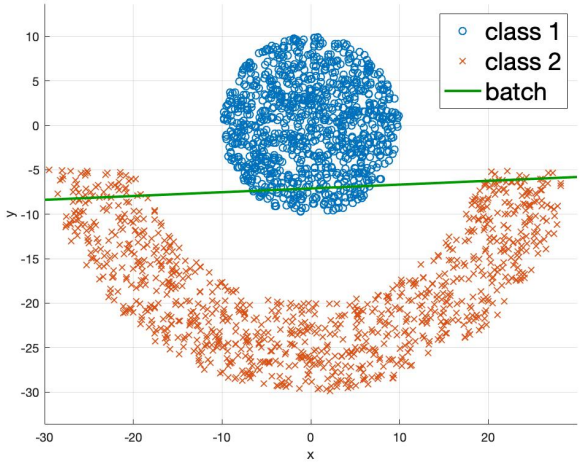

Figure 15: $D = 5$

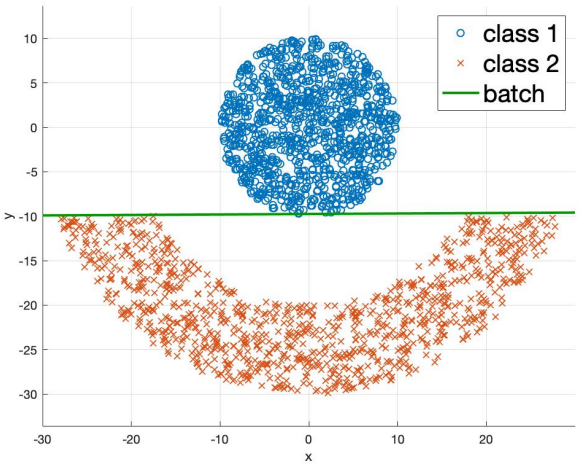

Figure 16: $D = 10$

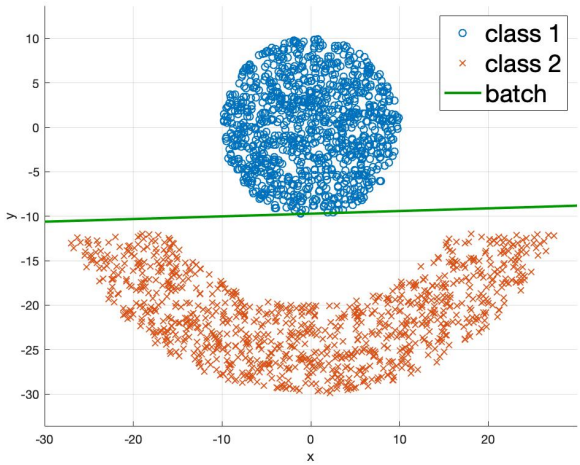

Figure 17: $D = 12$

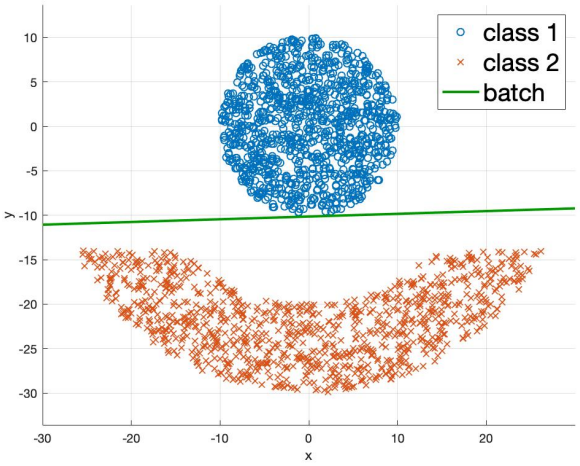

Figure 18: $D = 14$

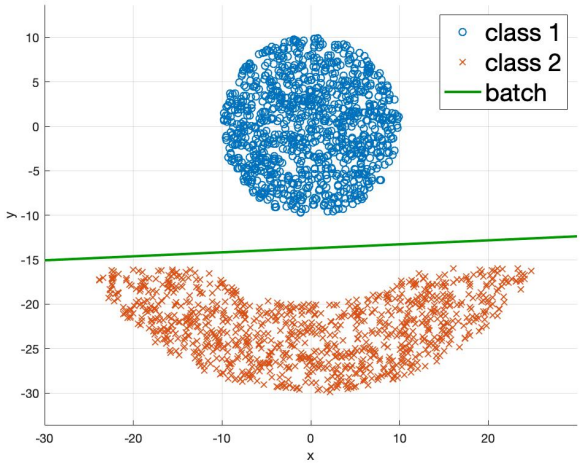

Figure 19: $D = 16$

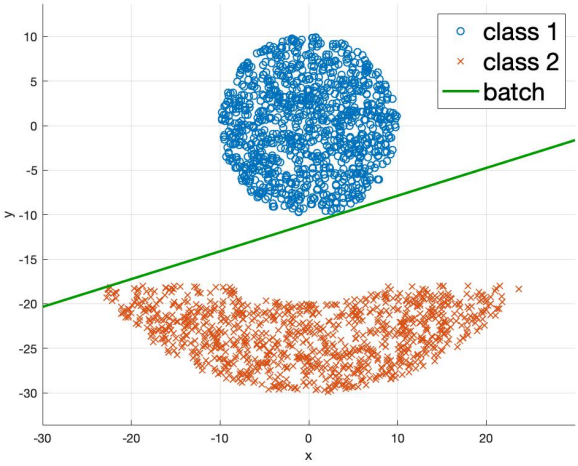

Figure 20: $D = 18$

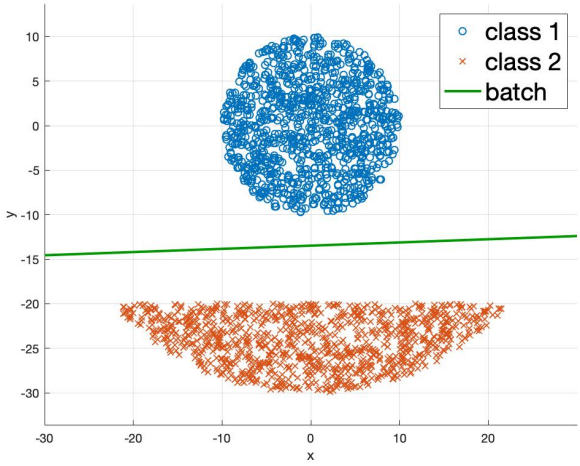

Figure 21: $D = 20$

Observations - for D equal to 0 and 5, the classes are not linearly separable, and hence the Perceptron doesn't given a correct classification. For every other case, the Perceptron correctly classifies the given data points.

(c) Functions used - batchPerceptron, onlinePerceptron
Value of D fixed = 14.
Let the weight vector be denoted by $[b, w_1, w_2]^T$, where b is the bias.
Online Learning

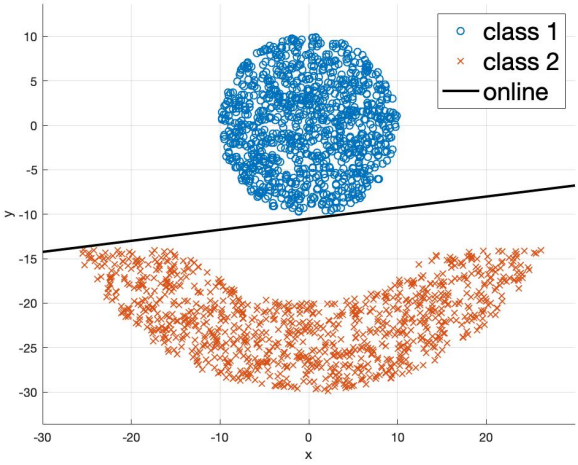

Figure 22: Initial weight vector $[0, 0, 0]^T$

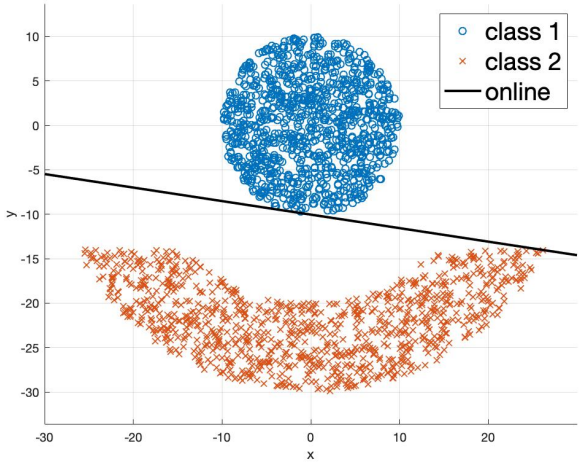

Figure 23: Initial weight vector $[0, 0, 1]^T$

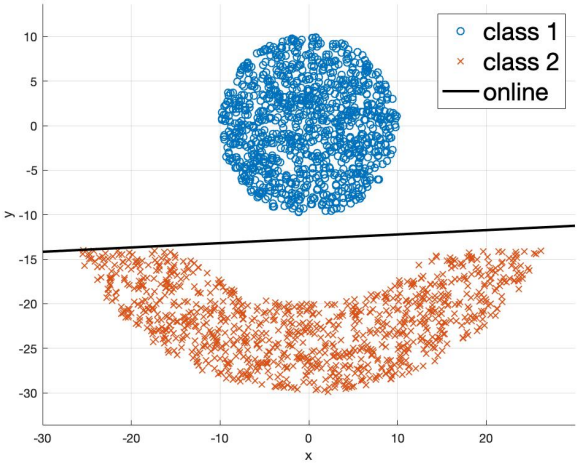

Figure 24: Initial weight vector $[0, 1, 0]^T$

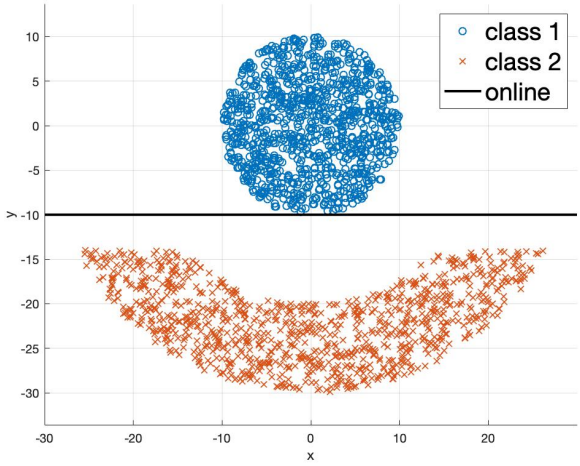

Figure 25: Initial weight vector $[10, 0, 1]^T$

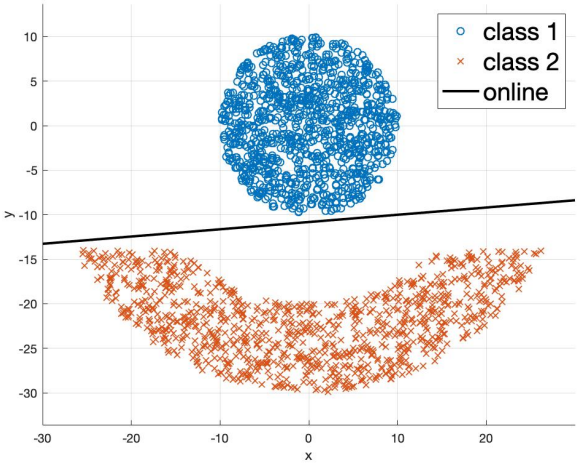

Figure 26: Initial weight vector $[-10, 1, 0]^T$

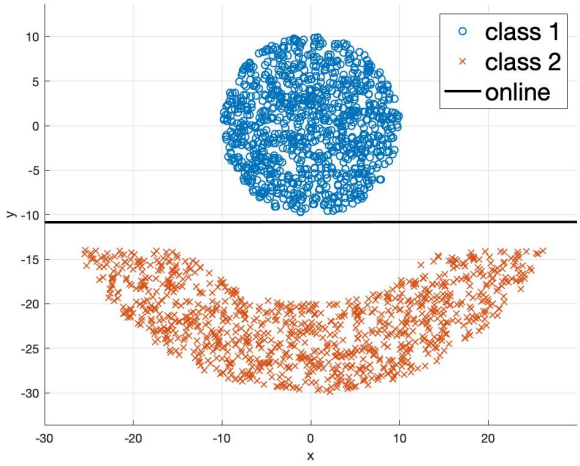

Figure 27: Initial weight vector $[1, 1, 1]^T$

Batch Learning

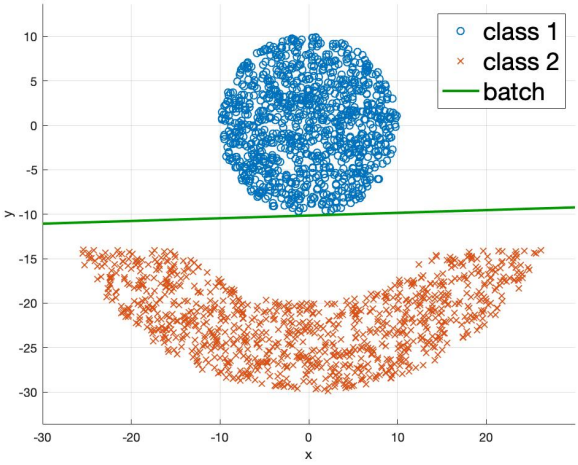

Figure 28: Initial weight vector $[0, 0, 0]^T$

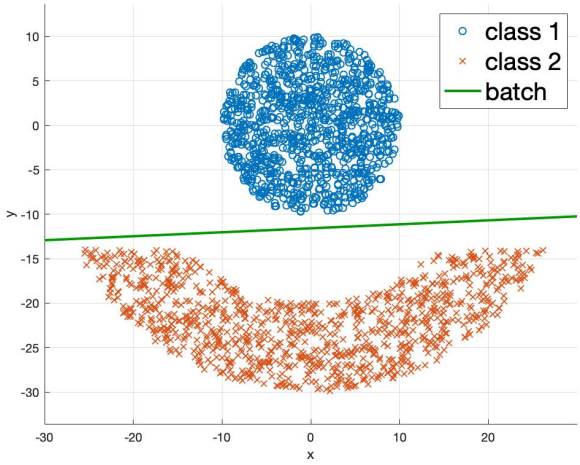

Figure 29: Initial weight vector $[0, 0, 1]^T$

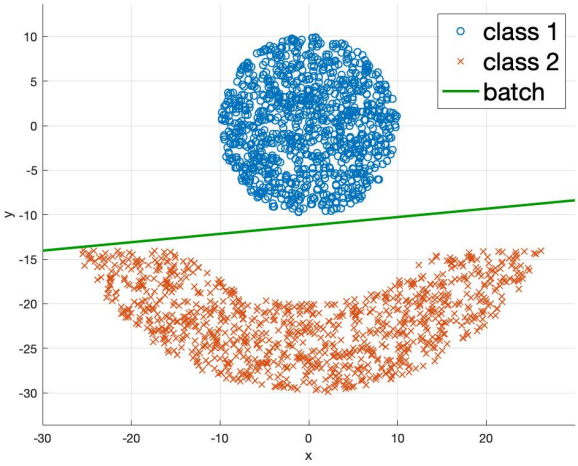

Figure 30: Initial weight vector $[0, 1, 0]^T$

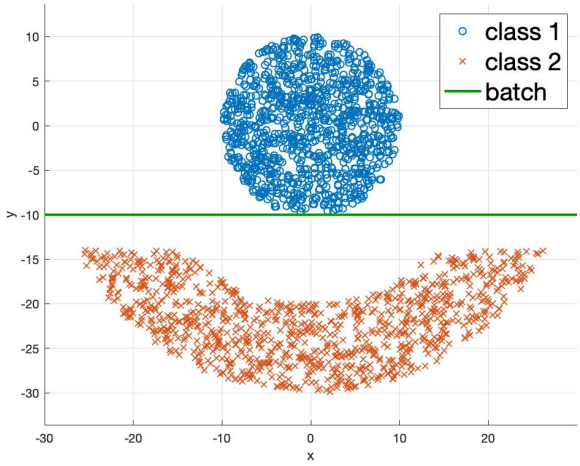

Figure 31: Initial weight vector $[10, 0, 1]^T$

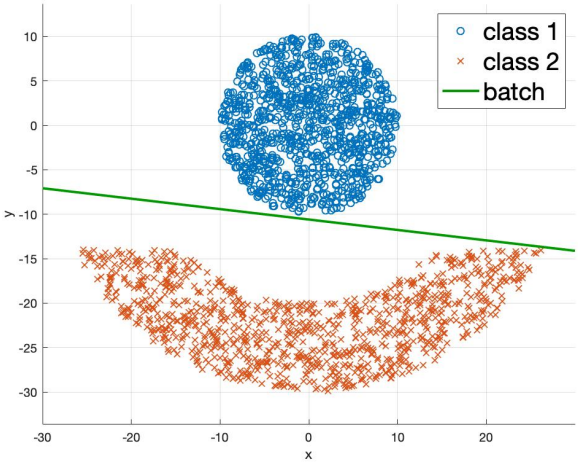

Figure 32: Initial weight vector $[-10, 1, 0]^T$

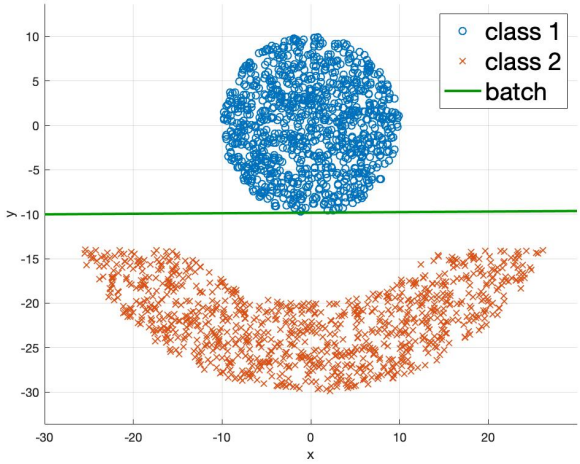

Figure 33: Initial weight vector $[1, 1, 1]^T$

Observations The Decision Boundaries corresponding to different initial conditions for the weight vector are different. This is because in the perceptron algorithm, the weights are modified only when an input is misclassified, and this update also depends on the values of the weights themselves. For different initial condition, the value of the weights that will be updated changes, as well as the inputs that are initially misclassfied change. Because of this, the final decision boundary that we get is different.

(d) Functions used - onlinePerceptron, onlinePerceptronR
We compare the results of the normal online mode, and online mode after randomizing the sequence of inputs (denoted as 'online' and 'random' in the plots respectively).

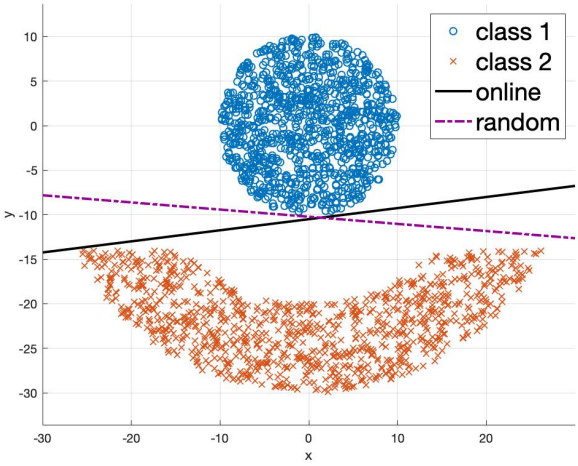

Figure 34: Online vs. Random

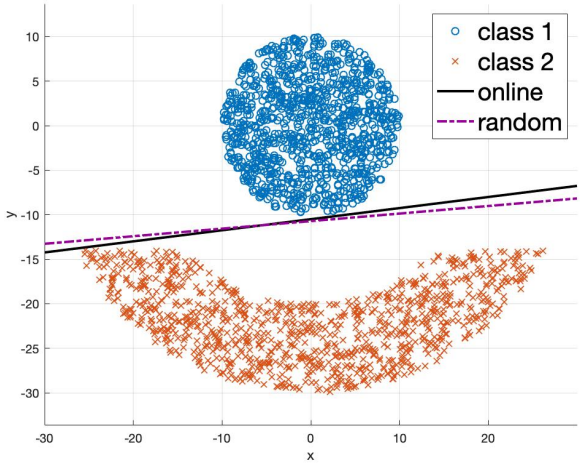

Figure 35: Online vs. Random

Observations The Decision Boundaries when the input sequence is randomized is coming out to be different, which is as expected. This is because the update to the weight vector is dictated by which inputs are misclassified. Let us say that one time the perceptron is shown input-p followed by input-q, both of which are misclassified, and hence contribute to the update of the weight vector. In a different run, the input-q is shown first to the perceptron. The input-p may no longer be misclassified to contribute to the weight vector.

(e) Functions used - addNoise, onlinePerceptron, batchPerceptron, accuracy.
The stopping criterion for learning is the maximum number of epochs. The learning is stopped after the maximum number of allowable epochs is reached, which was kept at 100 for these experiments.
In case the run time of the Perceptron is a problem, and every epoch is computationally expensive, then the stopping criterion can be - $norm(w(n+1) - w(n)) < \epsilon$, where ϵ is a small number.

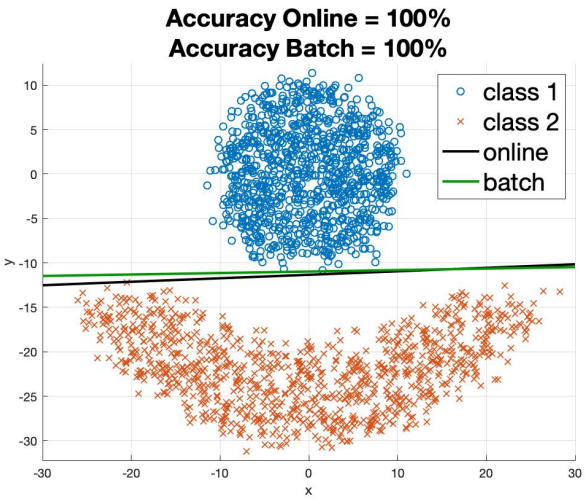

Figure 36: Variance=1

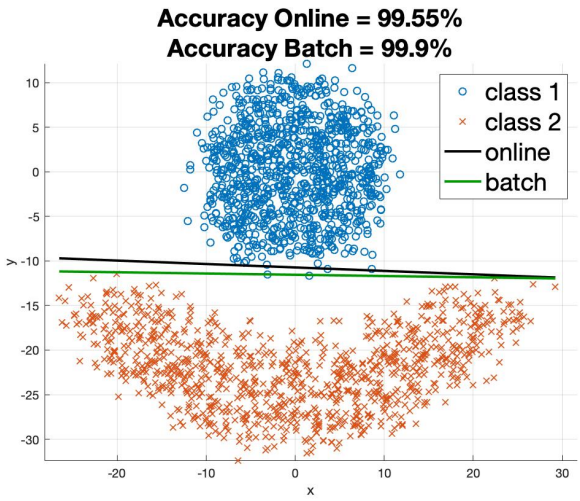

Figure 37: Variance=2

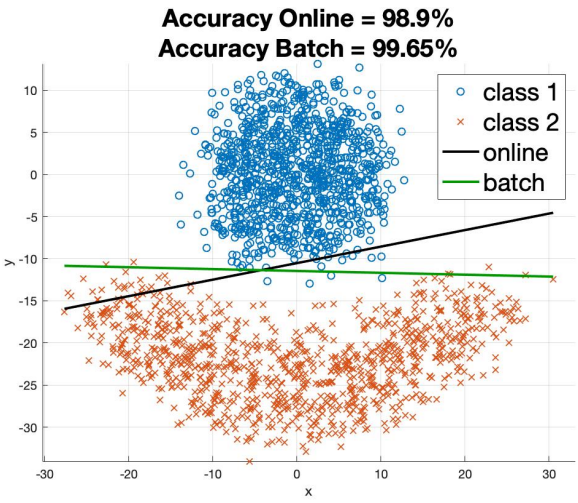

Figure 38: Variance=4

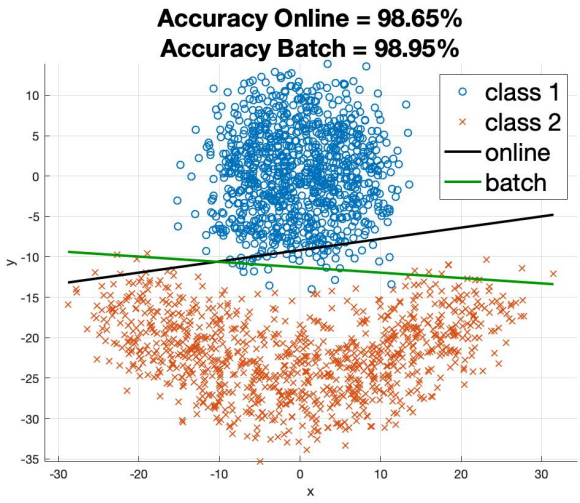

Figure 39: Variance=6

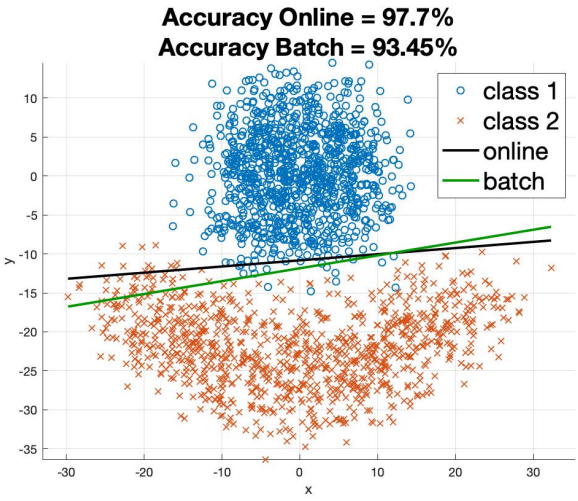

Figure 40: Variance=8

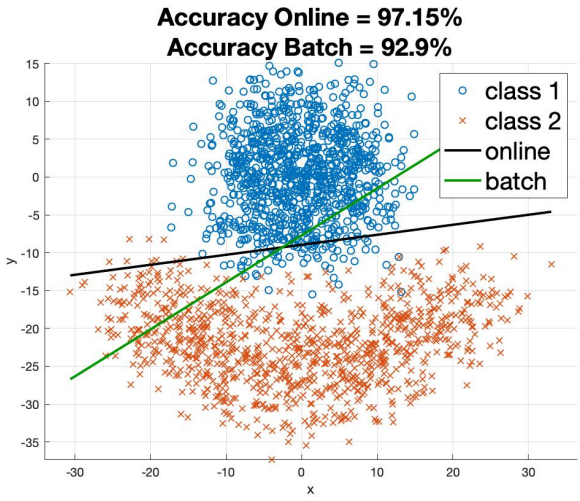

Figure 41: Variance=10

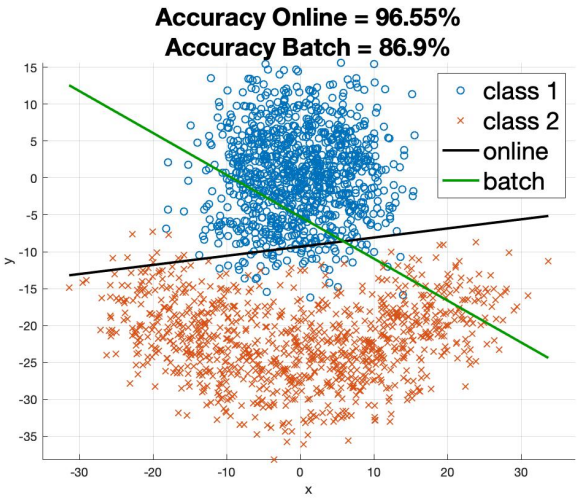

Figure 42: Variance=12

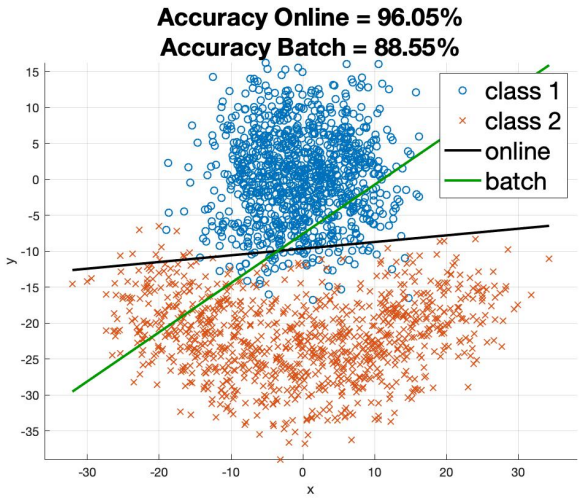

Figure 43: Variance=14

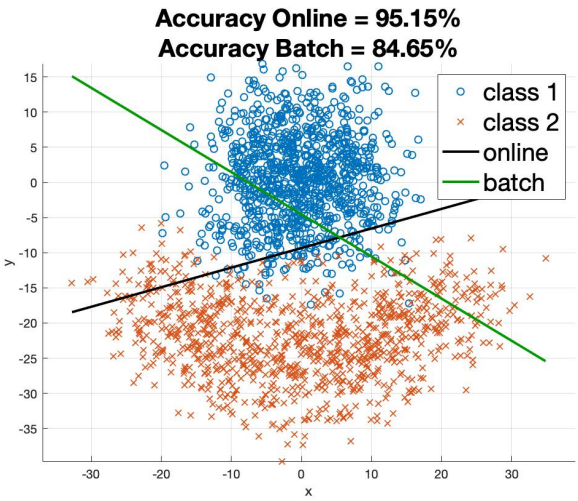

Figure 44: Variance=16

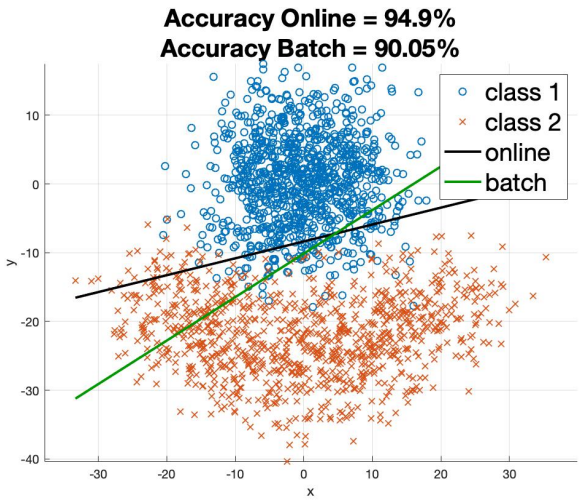

Figure 45: Variance=18

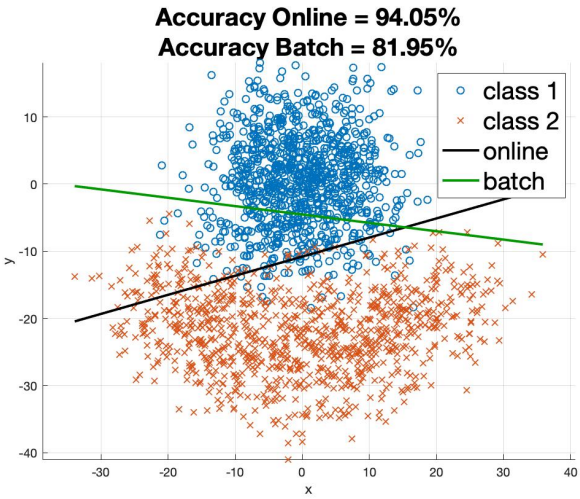

Figure 46: Variance=20

Observations The accuracy was calculated as the number of correctly classified points divided by the total number of points. As we can see, the accuracy is decreasing as the variance of the noise added is increasing, which is as expected. This is because as the variance of the noise that is added is increasing, the distance between the classes decreases. The classes no longer remain linearly separable.

(f) Value of D fixed is 14. Initial weight vector fixed is $[0, 0, 1]^T$.
For batch learning, n_{max} is the number of epochs taken to converge.
For online learning, if e is the number of epochs taken to converge, N is the total number of training samples, and n is the number of steps taken in the last epoch before the algorithm converges, then, n_{max} is $eN + n$.

Online Learning

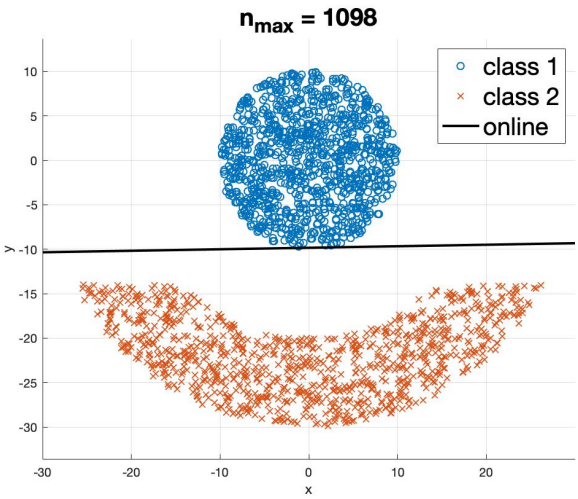

Figure 47: $\eta = 0.1$

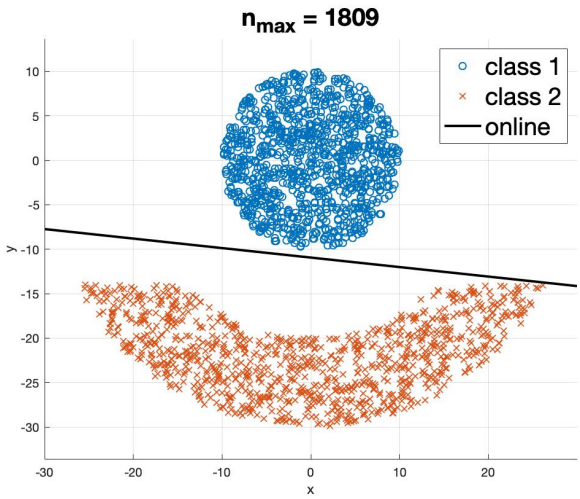

Figure 48: $\eta = 0.2$

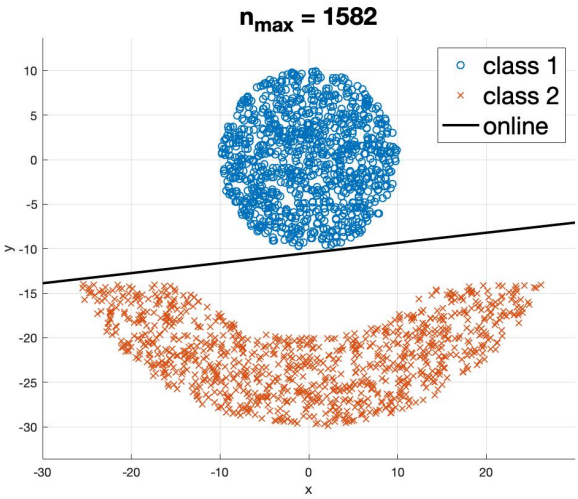

Figure 49: $\eta = 0.4$

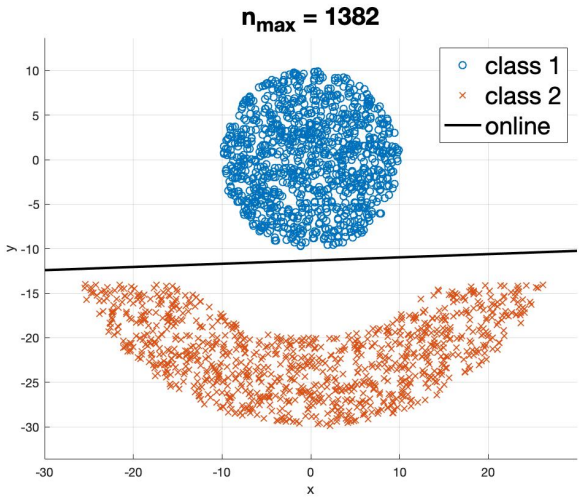

Figure 50: $\eta = 0.6$

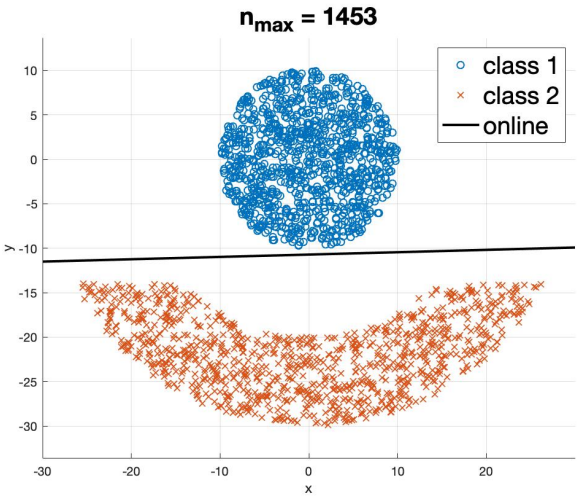

Figure 51: $\eta = 0.8$

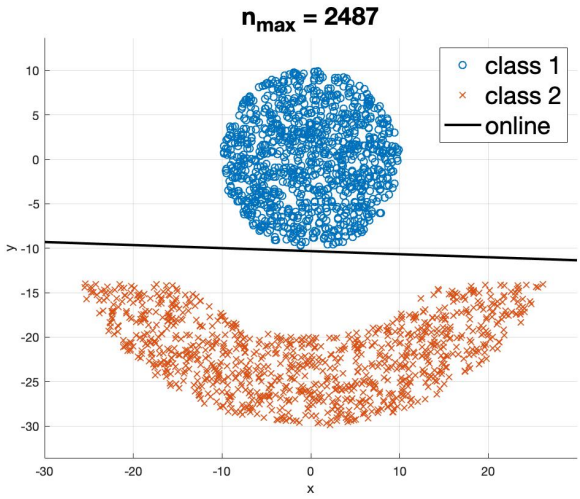

Figure 52: $\eta = 1$

Batch Learning

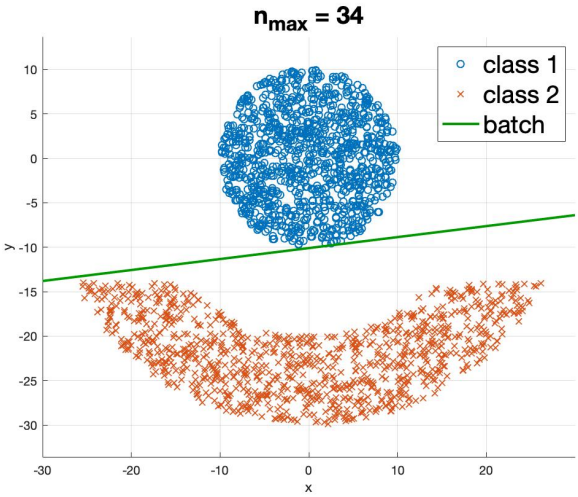

Figure 53: $\eta = 0.1$

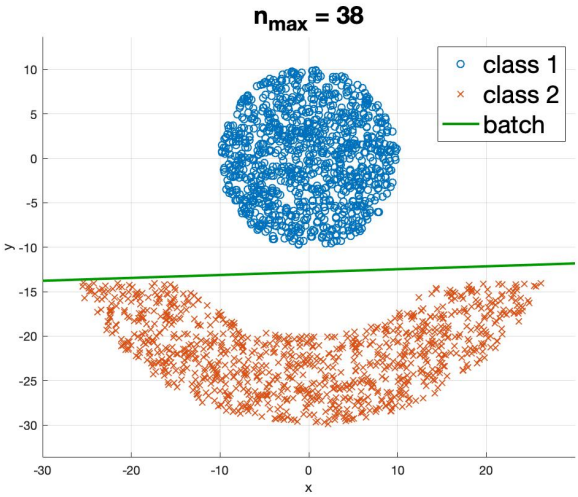

Figure 54: $\eta = 0.2$

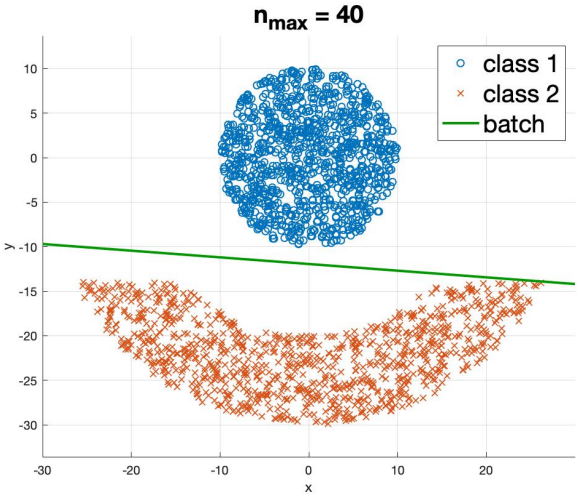

Figure 55: $\eta = 0.4$

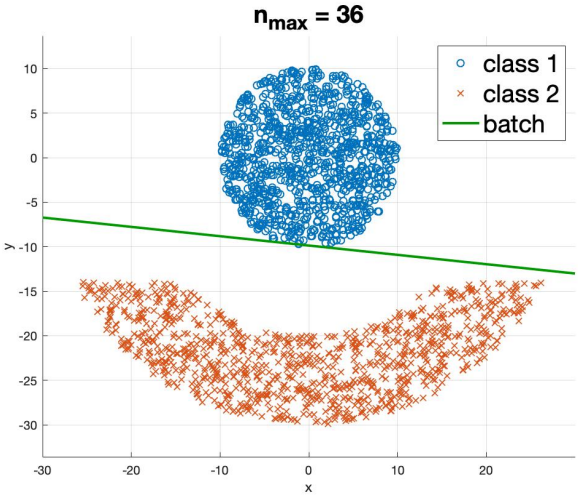

Figure 56: $\eta = 0.6$

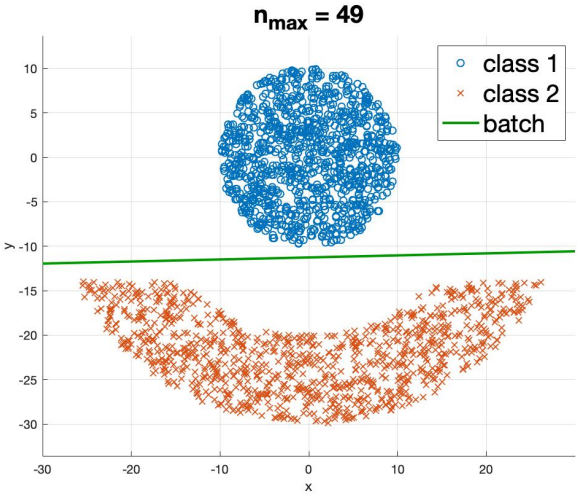

Figure 57: $\eta = 0.8$

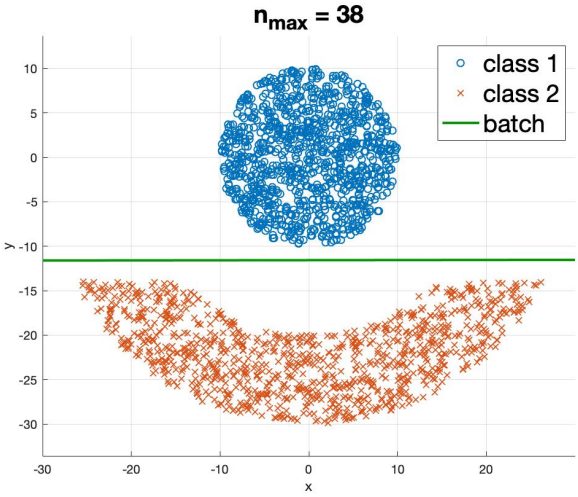

Figure 58: $\eta = 1$

(g) The learning rate η affects the convergence of the Perceptron. A small η will result in a slower convergence, but will give a smoother trajectory to the optimal weights. On the other hand, a larger η will give faster convergence, requiring a smaller number of steps, but at every step, a large adjustment will be made to the weights. This may make the network unstable, and the convergence oscillatory in nature, increasing the number of steps required.
The number of steps that is actually taken for the Perceptron to converge depends on an interplay between these two factors.
There is no monotonic pattern in the variation of n_{max} with the number of steps required. This suggests that our learning is becoming oscillatory before convergence.

Appendix

```
%{
function to generate the data points.
it takes as input R the radius of innermost circle, D is the given Distance
N is the total number of points to be generated
The coordinates of the points are stored in inputdata
and the desired output stored in desired.
%}
function [inputdata,desired] = genPoints(R,D,N)
rng(0); %to set the seed so that the input points are same in all the experiments
i = 1;
    while(i≤N/2)
        inputdata(i,1) = 2*R*rand-R;
        inputdata(i,2) = 2*R*rand-R;
        desired(i,1) = 1;
        if((inputdata(i,1)^2 + inputdata(i,2)^2) < R^2)
            i = i + 1;
        end
    end
Rin = 2*R;
Rout = 1.5*Rin;
    while(i≤N)
        inputdata(i,1) = 2*Rout*rand-Rout;
        inputdata(i,2) = 2*Rout*rand-Rout;
        desired(i,1) = -1;
        if((inputdata(i,1)^2 + inputdata(i,2)^2) < ((Rout)^2) && (inputdata(i,1)^2 ...
            + inputdata(i,2)^2) > ((Rin)^2))
            if(inputdata(i,2) ≤ D)
                i = i + 1;
            end
        end
    end
end
```

```
%{
This the Perceptron programmed in online mode with sequential inputs.
The inputs are -
N - total number of data points
data - values of inputs
desired - the desired output of the samples
eta - learning rate
wstart - the starting weight
epochMax - the maximum number of epochs allowed
The outputs are -
w - the final weights
acc - accuracy of the classification
emax - the number of epochs in which the our solution converged
nmax - the number of steps in which our perceptron converged. If the number
of epochs taken to converge is e, and the number of steps required in the
last epoch after which no update is made to the weight vector is n, then
the number of steps nmax is eN+n.
%}
function [w,acc,emax,nmax] = onlinePerceptron(N,data,desired,eta,wstart,epochMax)
    emax = inf;
    nmax = inf;
    w = wstart;
```

```
    flag = 0;
    for epoch = 1:epochMax
        wold = w;
        for n = 1:N
            woldin = w;
            v(n) = w'*[1 data(n,:)]';
            if(v(n)>0)
                y(n) = 1;
            else
                y(n) = -1;
            end
            w = w + eta*(desired(n)-y(n))*[1 data(n,:)]';
            if(norm(w-woldin)==0)
                if(flag==1)
                    if(nmax>(n+(epoch-1)*N))
                        nmax = n+(epoch-1)*N;
                    end
                else
                    nmax = n+(epoch-1)*N;
                end
                flag = 1;
            else
                flag = 0;
            end
        end
        if(norm(w-wold)==0)
            if(emax>epoch)
                emax = epoch;
            end
        end
    end
    acc = accuracy(y,desired,N);
    if(norm(w-wold)≠0)
        disp('online mode did not converge');
    end
end
```

```
%{
This the Perceptron programmed in batch mode.
The inputs are -
N - total number of data points
data - values of inputs
desired - the desired output of the samples
eta - learning rate
wstart - the starting weight
epochMax - the maximum number of epochs allowed
The outputs are -
w - the final weights
acc - accuracy of the classification
emax - the number of epochs in which the our solution converged
%}
function [w,acc,emax] = batchPerceptron(N,data,desired,eta,wstart,epochMax)
    w = wstart;
    emax = inf;
    for epoch = 1:epochMax
        wold = w;
        sumMisclassified = 0;
        for n = 1:N
```

```
            v(n) = w'*[1 data(n,:)]';
            if(v(n)>0)
                y(n) = 1;
            else
                y(n) = -1;
            end
            if(y(n)~=desired(n))
                sumMisclassified = sumMisclassified + desired(n)*[1 data(n,:)]';
            end
        end
        w = w + eta*sumMisclassified;
        if(norm(w-wold)==0)
            if(emax>epoch)
                emax = epoch;
            end
        end
    end
    acc = accuracy(y,desired,N);
    if(norm(w-wold)~=0)
        disp('batch mode did not converge');
    end
end
```

```
%{
This the Perceptron programmed in online mode with randomised inputs.
The inputs are -
N - total number of data points
data - values of inputs
desired - the desired output of the samples
eta - learning rate
wstart - the starting weight
epochMax - the maximum number of epochs allowed
The outputs are -
w - the final weights
acc - accuracy of the classification
emax - the number of epochs in which the our solution converged
nmax - the number of steps in which our perceptron converged. If the number
of epochs taken to converge is e, and the number of steps required in the
last epoch after which no update is made to the weight vector is n, then
the number of steps nmax is eN+n.
%}
function [w,acc,emax,nmax] = onlinePerceptronR(N,data,desired,eta,wstart,epochMax)
    emax = inf;
    nmax = inf;
    flag = 0;
    w = wstart;
    rng('shuffle');
    for epoch = 1:epochMax
        wold = w;
        p = randperm(N);
        for n = 1:N
            woldin = w;
            v(n) = w'*[1 data(p(n),:)]';
            if(v(n)>0)
                y(p(n)) = 1;
            else
                y(p(n)) = -1;
            end
```

```
            w = w + eta*(desired(p(n))-y(p(n)))*[1 data(p(n),:)]';
            if(norm(w-woldin)==0)
                if(flag==1)
                    if(nmax>(n+(epoch-1)*N))
                        nmax = n+(epoch-1)*N;
                    end
                else
                    nmax = n+(epoch-1)*N;
                end
                flag = 1;
            else
                flag = 0;
            end
        end
        if(norm(w-wold)==0)
            if(emax>epoch)
                emax = epoch;
            end
        end
    end
    acc = accuracy(y,desired,N);
    if(norm(w-wold)~=0)
        disp('online mode did not converge');
    end
end
```

```
%{
function to calculate accuracy of the perceptron
Take as input the calculated and desired responses, and the total number of points
%}
function ans = accuracy(calculated, desired, N)
ans = 0;
    for i = 1:N
        if(calculated(i)~=desired(i))
            ans = ans+1;
        end
    end
ans = (N-ans)/N*100;
end
```

```
%{
function to add noise to the input data.
this takes as input the data which has to be made noisy, the no. of samples N, the ...
   mean mu and the standard deviation sigma of the gaussian noise to be added it ...
   gives as output the modified data set
%}
function dataModified = addNoise(inputData,N,mu,sigma)
rng(1); %sets the seed for random number generation
    for i = 1:N
        r1 = sigma*randn + mu;
        r2 = sigma*randn + mu;
        dataModified(i,1) = inputData(i,1) + r1;
        dataModified(i,2) = inputData(i,2) + r2;
    end
end
```