The Catholic University of America
School of Engineering
Department of Electrical Engineering and Computer Science

CSC/ECE/DA 427/527 Fundamentals of Neural Networks

Project 2

Double-moon Classification using the Least Mean Squares Algorithm

Hoang Cao

Instructor: Dr. Hieu Bui

November 11th, 2020

Table of Content

1. Introduction

1.1 Autoregressive Model

In an AR model, the value of the outcome variable (Y) at some point t in time is — like "regular" linear regression — directly related to the predictor variable (X). Where simple linear regression and AR models differ is that Y is dependent on X and previous values for Y.

The AR process is an example of a stochastic process, which have degrees of uncertainty or randomness built in. The randomness means that you might be able to predict future trends pretty well with past data, but you're never going to get 100 percent accuracy. Usually, the process gets "close enough" for it to be useful in most scenarios.

An AR(p) model is an autoregressive model where specific lagged values of y_t are used as predictor variables. Lags are where results from one time period affect following periods. The value for "p" is called the order. For example, an AR(1) would be a "first order autoregressive process." The outcome variable in a first order AR process at some point in time t is related only to time periods that are one period apart (i.e. the value of the variable at t – 1). A second or third order AR process would be related to data two or three periods apart.

The AR(p) model is defined by the equation:

$y_t = \delta + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \ldots + \varphi_p y_{t-1} + A_t$

Where:

- $y_{t-1}, y_{t-2} \ldots y_{t-p}$ are the past series values (lags),
- A_t is white noise (i.e. randomness),
- and δ is defined by the following equation:

$$\delta = \left(1 - \sum_{i=1}^{p} \phi_i\right) \mu ,$$

where μ is the process mean

1.2 Least Mean Square Algorithm (LMS)

The Least Mean Squares (LMS) algorithm is a logical choice of subject to examine, because it combines the topics of linear algebra (obviously) and graphical models, the latter case because we can view it as the case of a single, continuous-valued node whose mean is a linear function of the value of its parents.

The High-Level Problem

We are going to be analyzing LMS in the context of linear regression, i.e., we will have some input features $x_n = (x_1, x_2, \ldots, x_k)^{(n)}$ along with their (scalar-valued) output y_n as our data, and the goal is to estimate a parameter vector θ such that $y_n = \theta^T x_n + \epsilon_n$, where the ϵ_n is admitting that we do not expect to exactly match y_n.

When we have ordinary linear regression, we often express the data all together in terms of matrices. Thus, we could have X be our $m \times n$ matrix of features, where there are m samples and n variables per sample, θ be the $n \times 1$ column vector of parameters, and the goal would be to find a solution θ to the problem $X\theta = Y$, where Y is the column vector of actual outputs. The problem is *linear* because the equations are linear in θ (X itself can actually be input to a more complicated function). In the rare case that X is square and that all columns are linearly independent, this immediately reduces to $\theta = X^{-1}Y$. Most of the time, though, X is a "tall" matrix and the data vector Y lies *outside* the column space of X.

The most natural solution, it seems, is to find the projection of Y onto the subspace of $Col(X)$, as that intuitively should minimize all our ϵ_n terms. Thus, we have the normal equations, the most important one in statistics: $(X^T X)\theta = X^T Y$. In most cases[1], we can invert $(X^T X)$. If not, we can use the *pseudo-inverse*[2].

There are several ways of getting to the normal equations. One can appeal geometrically, by using geometry and linear algebra. Alternatively, we can derive it based on the following cost function:

$$\begin{aligned} J(\theta) &= \frac{1}{2}\sum_{n=1}^{N} \epsilon_n^2 \\ &= \frac{1}{2}\sum_{n=1}^{N} (y_n - \theta^T x_n)^2 \\ &= \frac{1}{2}(y - X\theta)^T (y - X\theta) \\ &= \frac{1}{2}\|y - X\theta\|_2^2 \end{aligned}$$

2. Project Objective

2.1 Project Description

We will perform 4 tasks in this project:

1. We will build an Autoregressive (AR) model of order 1 to generate data and verify the statistical learning theory of the least-mean- square algorithm as discussed in Chapter 3, section 3.10. You will perform the computer experiment on linear prediction for the following learning-rate parameters: η = 0.001, η = 0.002, η = 0.01 and η = 0.02.

2. Perform the computer experiment on pattern classification for the distance of separation between the two moons of Figure 1 set at $d = 1$, $d = -4$, and $d = 0$ using the least- mean-square algorithm.
3. Compare the results of your experiments in Task 2 with the experiments on the perceptron and on the method of least squares.
4. Plot the pattern-classification learning curves of the least-mean-square algorithm applied to the double-moon configuration for the following values assigned to the distance of separation: $d = 1$, $d = -4$, $d = 0$. Compare the results of the experiment with the corresponding ones obtained using Rosenblatt's perceptron.

Figure 1.8 shows a pair of "moons" facing each other in an *asymmetrically* arranged manner. The moon labeled "Region *A*" is positioned symmetrically with respect to the *y*-axis, whereas the moon labeled "Region *B*" is displaced to the right of the *y*-axis by an amount equal to the radius *r* and below the *x*-axis by the distance *d.* The two moons have identical parameters:
radius of each moon, r = 10
width of each moon, $w = 6$
The vertical distance *d* separating the two moons is adjustable; it is measured with respect to the *x*-axis, as indicated in Fig.1:
- Increasingly positive values of d signify increased separation between the two moons;
- Increasingly negative values of d signify the two moons 'coming closer to each other.
The training sample t consists of 1,000 pairs of data points, with each pair consisting of one point picked from region *A* and another point picked from region *B*, both randomly. The test sample consists of 1,000 pairs of data points, again picked in a random manner.

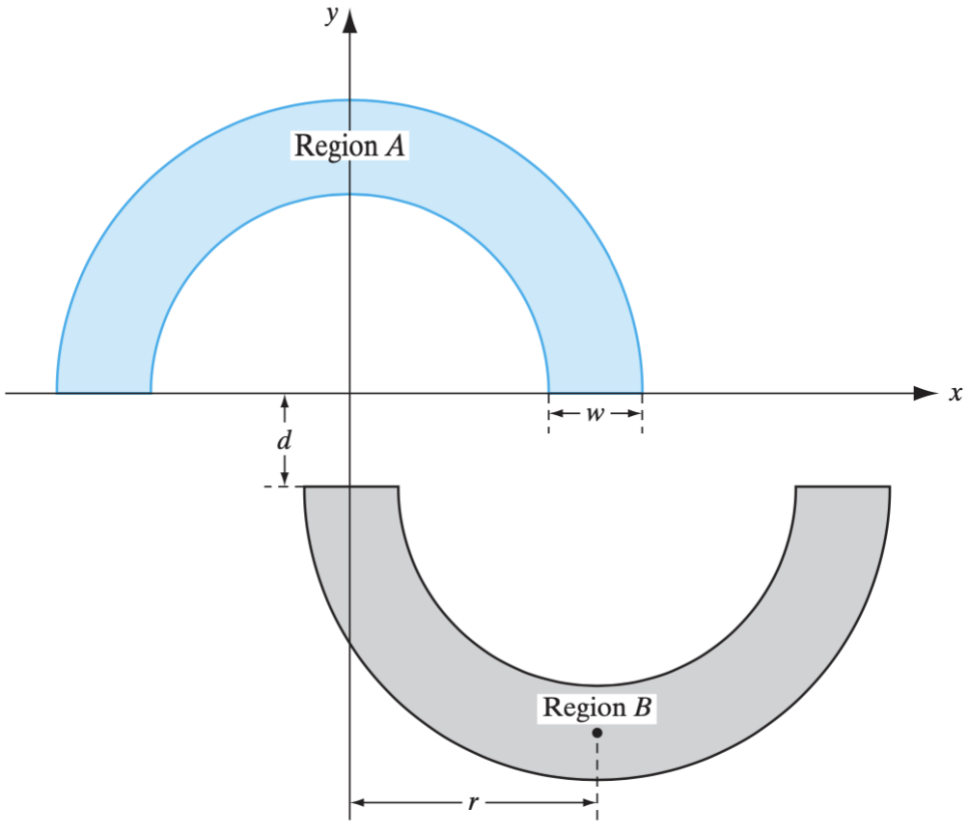

Figure.1: The double moon classification problem.

3. Implementation of Algorithm

3.1 Generating Datapoint

We will be using datapoint function to generate data and create training and test dataset.

```
def datapoint(points,distance,radius, width):
    x1,x2,y1,y2 = moon(points,distance,radius, width)
    x3 = x1 + x2
    y3 = y1 + y2
    tr1_data = []
    tr2_data = []
    train_data = []
    label = []
    # Since the data have no label, we need to assign for them.
    # We aasign 1 for the first half moon and 0 for the other half
    for i in range(len(x1)):
      a = [x1[i],y1[i], 1]
      tr1_data.append(a)
    for i in range(len(x2)):
      b = [x2[i],y2[i], 0]
      tr2_data.append(b)
    pre_data = np.concatenate((tr1_data,tr2_data))
    np.random.shuffle(pre_data)
    # Extract train_data and label from labeled data
    for i in range(len(pre_data)):
        x = pre_data[i][0]
        y = pre_data[i][1]
        output = pre_data[i][2]
        train_data.append([x, y])
        label.append(output)
    X1 = np.array(train_data)
    Y1 = np.array(label)
    return X1,Y1
```

3.2 Normalize Data

After receiving the data from generating moon function and datapoint function, we normalize the dataset in order for classification task using Least Mean Square algorithm to obtain the best result.

```
def normalize(X,y):
    X_norm = np.ndarray.copy(X)
    y_norm = np.ndarray.copy(y)
    X_mu = np.mean(X_norm,axis=0)
    X_dev = np.std(X_norm, axis = 0)
    X_norm = X_norm - X_mu
    X_dev[X_dev ==0] = 1
    X_norm = X_norm / X_dev
    y_mu = np.mean(y_norm,axis=0)
    y_norm = y_norm - y_mu
    X_norm = np.hstack((np.ones(X_norm.shape[0])[np.newaxis].T,X_norm))
    return X_norm,y_norm
```

3.3 Initialization

We may express the evolution of the weight vector in the LMS algorithm as:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)]$$
$$= [\mathbf{I} - \eta\mathbf{x}(n)\mathbf{x}^T(n)]\hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)d(n)$$

where $\mathbf{I}$ is the identity matrix. In using the LMS algorithm, we recognize that

$$\hat{\mathbf{w}}(n) = z^{-1}[\hat{\mathbf{w}}(n+1)]$$

We set the weight vector = 0, learning rate = 0.01.

3.4 Fit Function to Train Data using LMS algorithm

The function will be trained with 1000 data points generated and normalized from double half-moon function with distance = 0, distance = 1, and distance = -4

```
def fit(X,y,epochs,lr):
    X, y = normalize(X,y)
    w = np.zeros(X[0].shape)
    MSE = []
    square_e = 0
    for _ in range(epochs):
        square_e = 0
        for i,target in zip(X, y):
            yhat = np.dot(i, w)
            errors= target - yhat
            mse = target - predict(i[1:],w)
            w = w + lr* i * errors
            square_e += (mse ** 2)
        MSE.append(square_e/len(X))
        return w, MSE
```

3.5 Decision Boundary

The decision boundary will be plotted in a 2-D space

```
def draw_line(w, xx):
    '''
    Draw decision boundary
    w0 + w1x + w2y = 0 => y = -(w0 + w1x)/w2
    '''
    x = np.linspace(np.amin(xx),np.amax(xx),100)
    y = -(w[0]+x*w[1])/w[2]

    plt.plot(x, y, '--k',label="DB")
```

4.

5. Perform Pattern Classification using LMS Algorithm and Compare the results of Least Mean Square Algorithm with the Rosenblatt ‘s Perceptron and the Method of Least Squares (Task2 + Task3).

5.1 Distance = 1

The parameters of the double moon will be trained with 1000 data points, distance = 1, radius = 10, width = 6, learning rate = 0.01, and test with the same amount of data. After training and testing, we obtain the result and compare with Rosenblatt's Perceptron and Least Squares Methods:

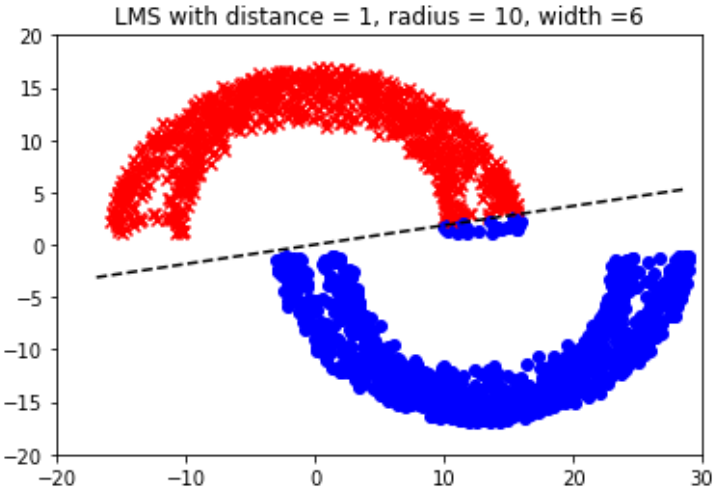

Figure. 2: Classification using LMS with d=1, radius = 10, width = 6

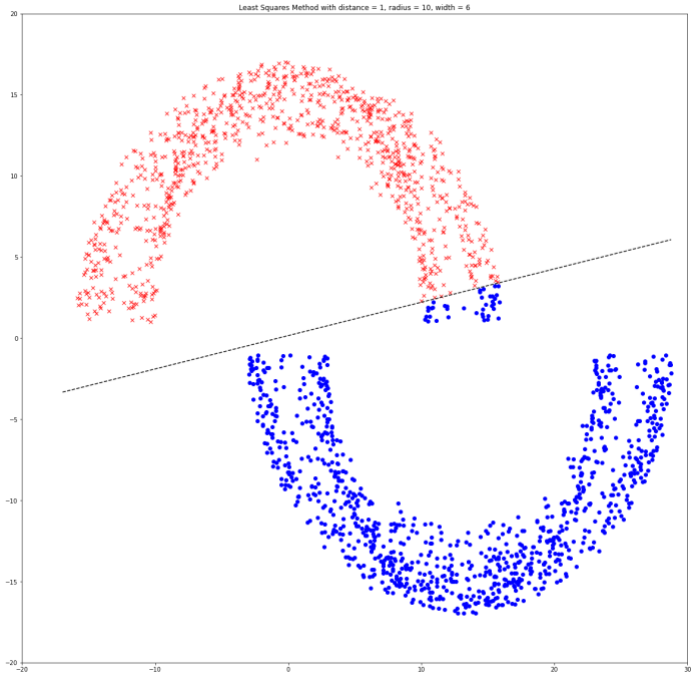

Figure. 3: Classification using Least Squares with d = 1, radius = 10, width = 6

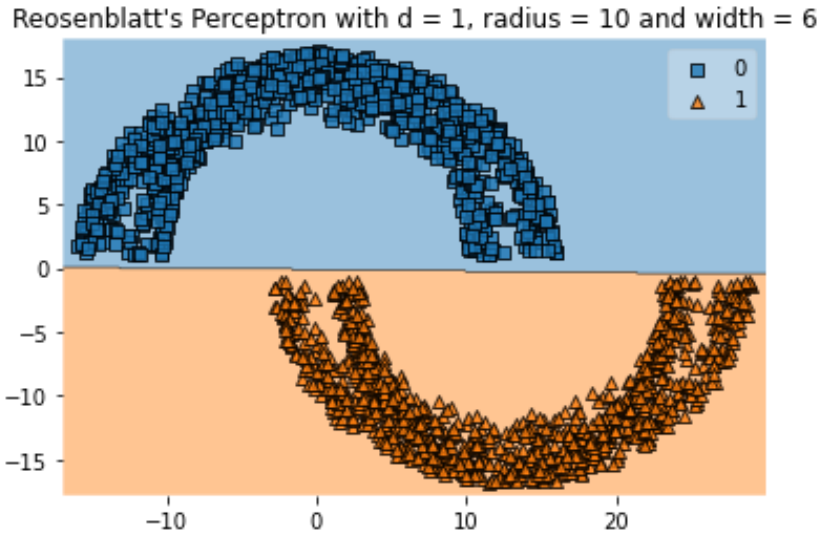

Figure. 4: Classification using Perceptron with d = 1, radius = 10, width = 6

On this position where two half-moons have distance between each other, Perceptron algorithm perform well on linear separation these data points comparing with Least Squares and LMS which have a similar result.

5.2 Distance = -4

The parameters of the double moon will be trained with 1000 data points, distance = -4 , radius = 10, width = 6, learning rate = 0.01, and test with the same amount of data. After training and testing, we obtain the result and compare with Rosenblatt's Perceptron and Least Squares Methods:

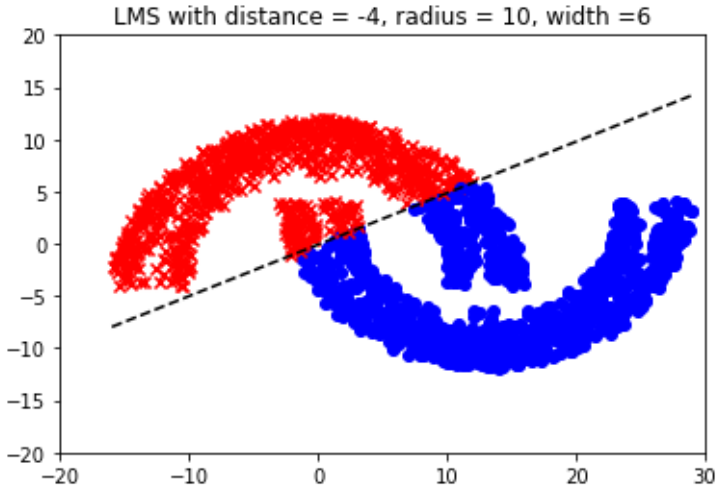

Figure. 5: Classification using LMS with d = -4, radius = 10, width = 6

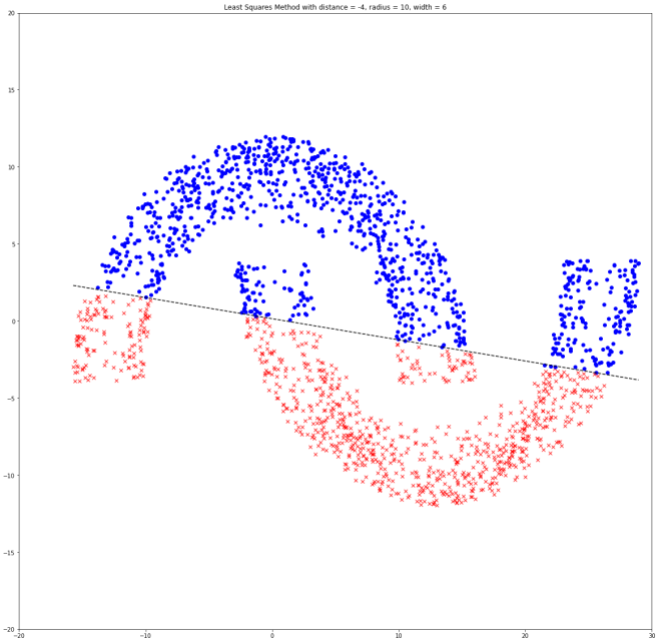

Figure. 6: Classification using Least Squares with d = -4, radius = 10, width = 6

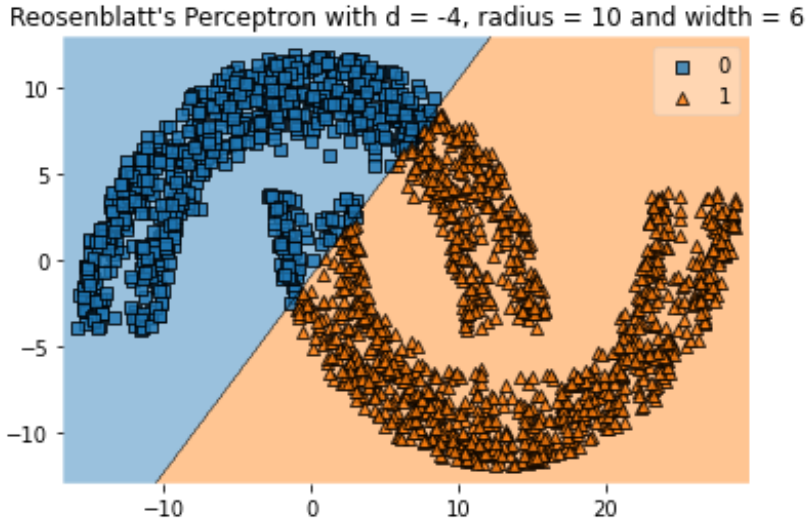

Figure. 7: Classification using Perceptron with d = -4, radius = 10, width = 6

In this case where two-half moons are overlapped on each other, all three classifiers gave similar results, however, LMS and Least Squares algorithm did a better job on drawing decision boundary than Perceptron.

5.3 Distance = 0

The parameters of the double moon will be trained with 1000 data points, distance = 0 , radius = 10, width = 6, learning rate = 0.01, and test with the same amount of data. After training and testing, we obtain the result and compare with Rosenblatt's Perceptron and Least Squares Methods:

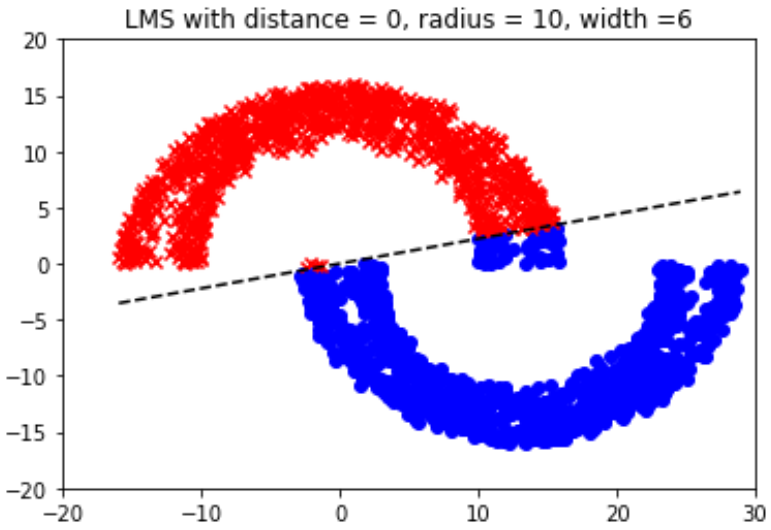

Figure. 8: Classification using LMS with d = 0, radius = 10, width = 6

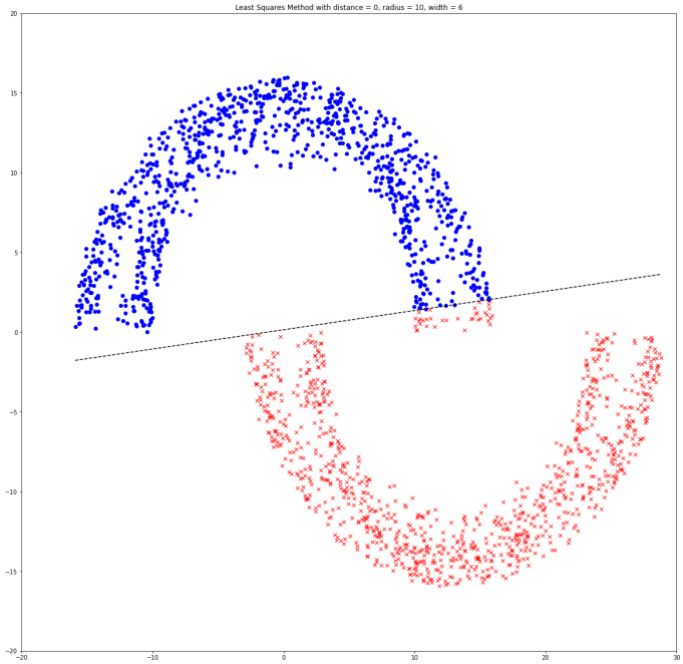

Figure. 9: Classification using Least Squares with d = 0, radius = 10, width = 6

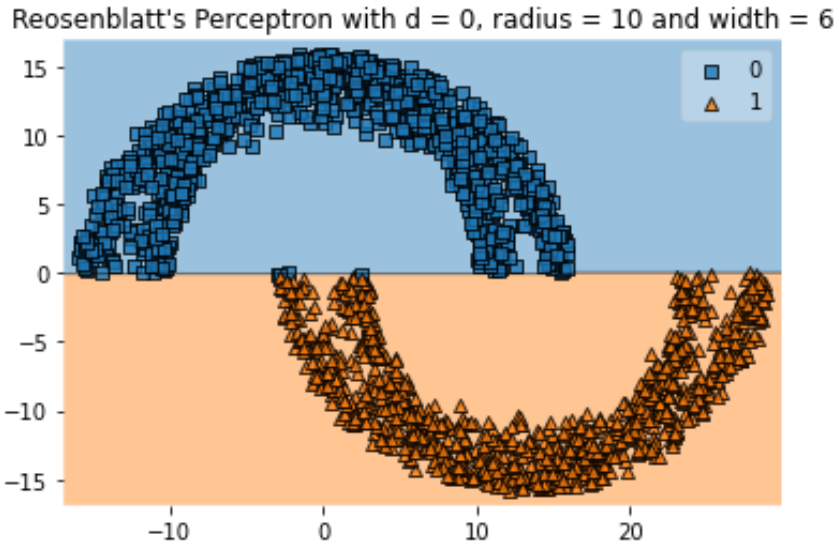

Figure. 10: Classification using Perceptron with d = 0, radius = 10, width = 6

Results show that Rosenblatt's Perceptron Algorithm perform a better classification than LMS and Least Squares Algorithms when the distance between two half-moons equals 0.

6. **Compare Pattern-Classification Learning Curves of the Least Mean Square Algorithm with Rosenblatt's Perceptron Learning Curve (Task4)**

   In this task, we will compare pattern-classification between LMS and Rosenblatt's perceptron using learning curve graphs.

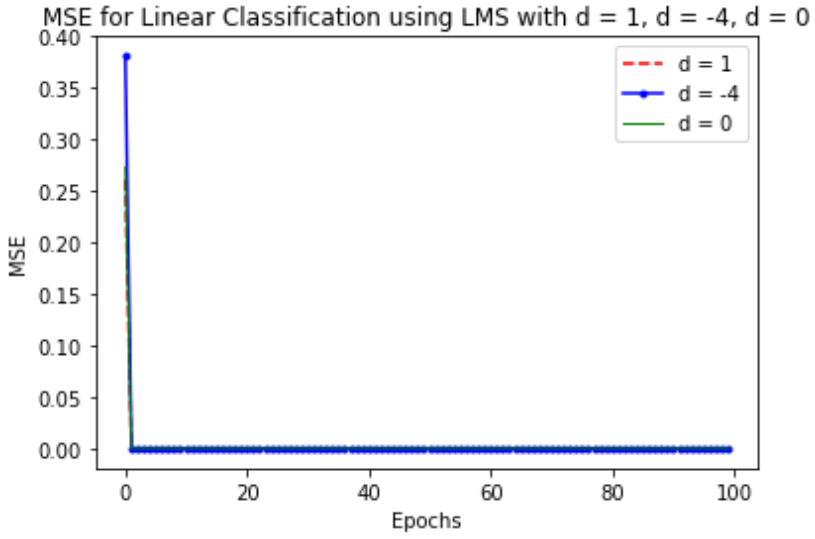

Figure. 11: Learning Curve of LMS with d = 0, d = -4, d = 1

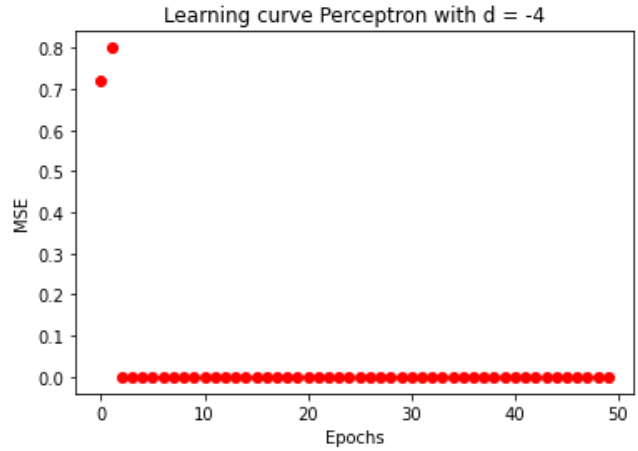

Figure. 12: Learning Curve of Perceptron with d = -4

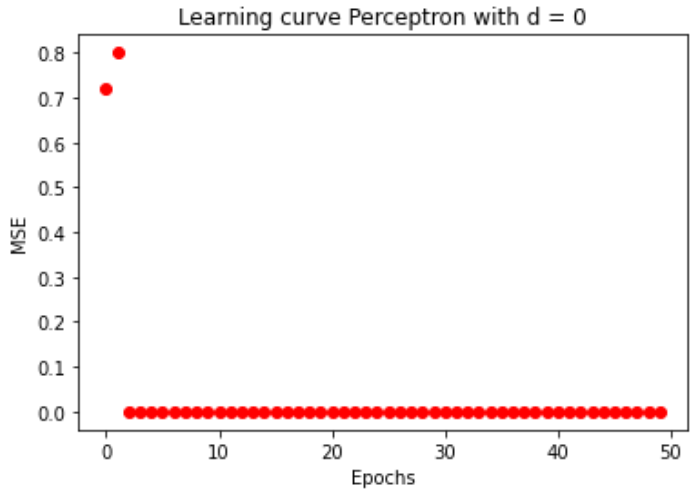

Figure. 13: Learning Curve of Perceptron with d = 0

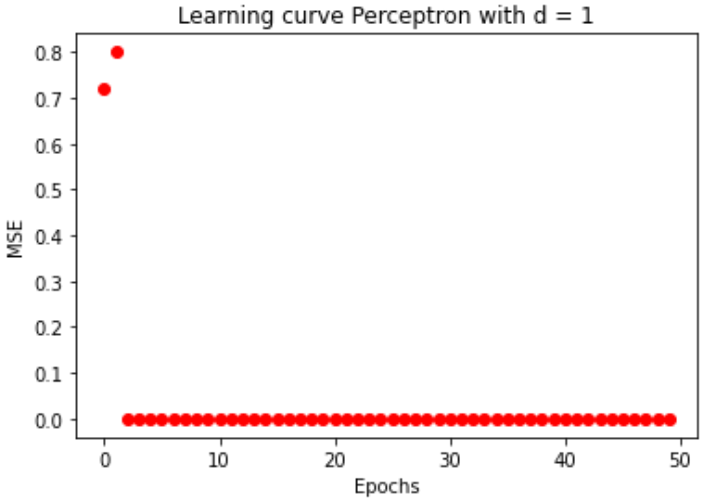

Figure. 14: Learning Curve of Perceptron with d = 1

Based on the learning curve graphs of LMS and Perceptron through different distance d, we believe that Perceptron and LMS's performances are identical. Hence, they are both good at linear classification.

7. Conclusion

Throughout this project, we gain a certain understanding about Least Mean Square Algorithm and how it is comparing with Least Squares Methods and Rosenblatt's Perceptron. We can conclude that Perceptron perform better than LMS and Least Squares when there is a distance data points between two half-moons but not efficient when perform non-linear classification task. LMS demonstrate a slightly better performance when it comes to non-linear classification. Overall, they perform well on linear classification task

My github: https://github.com/caoh96/CSC-527

8. References

https://www.statisticshowto.com/autoregressive-model/

https://danieltakeshi.github.io/2015-07-29-the-least-mean-squares-algorithm/