# Neural Network and Deep Learning

# Assignment 2

**Submitted To:**

**Dr.Alioune Ngom**

**Submitted By:**

**Aayushi Navinchandra Patel (Student ID:110087817)**

**Aaditya Pradipbhai Parekh (Student ID:- 110084734)**

**Dhruvkumar Arvind Patel (Student ID:- 110055817)**

**Mrinal Walia (Student ID:- 110066886)**

**Date: 31-05-2022**

**GitHub Link: https://github.com/abhiwalia15/COMP8610_Assignment_2**

**Assignment_2**

**Question 1: In this question you are to create some simulated data sets and then use the Adaline neuron and the Sigmoid to perform some prediction. Use whatever programming language you want to use.**

**Generate 5000 synthetic data points (*x*, *y*) as follows:**

- Using the `rnorm()` function in R (or equivalent in Matlab or Python or etc), create a vector, *x*, containing 5000 observations drawn from a Gaussian distribution *N*(0, 1) [i.e., a normal distribution with mean 0 and variance 1]. This vector *x* represents your set of inputs *x*.

- Using the `rnorm()` function in R (or equivalent in Matlab or Python or etc), create a vector, *eps*, containing 5000 observation drawn from a *N*(0, 0.25) distribution; i.e., a normal distribution with mean 0 and variance 0.25.

- Using vectors *x* and *eps*, generate a vector *y* according to the model

$$y = -1 + 0.5x - 2x^2 + 0.3x^3 + eps.$$

Your 5000 data-points (*x*, *y*) are generated upon completion of this Part-c. Note that the true function is a cubic function with true weight vector being $w_{true}$ = (-1, +0.5, -2, +0.3).

- Implement the Adaline and Sigmoid neuron learning algorithms using (i) batch gradient descent [BGD] and (ii) stochastic gradient descent [SGD]. Using a cross-validation method of your choice (LOOCV or 10-fold-cv), test and compare their regression performances over the synthetic dataset created above. The initializations, the learning rate, the size of test set and training set, and the stopping criterion, and etc are left for you to explore. Think about the reasons why you use a particular strategy. Use your creativity and perform whatever experiments you want to test, and then tell me whatever story your experiments told you.

**Assignment_2**

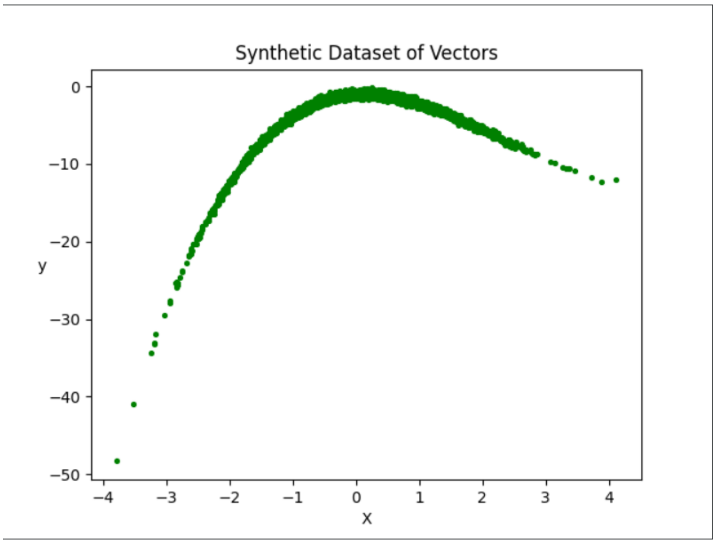
## Solution:

The below code solves the given question step by step with all required explanations:

1. Using function rnorm(), we first generate vector x and vector eps. Then we calculate vector y using vector x and vector eps.

```
# Required Libraries for Program
# Suggested running this program on PyCharm IDE
import matplotlib.pyplot as plt
from mlxtend.regressor import LinearRegression
from numpy import random
import numpy as np
from sklearn import metrics

# Q1- Generate random data for vector X and Vector ESP and using those Vector we generate Vector Y.
# Given linear equation for Y = -1 + (0.5 * X) - (2 * X)^2 + (0.3 * X)^3
x = random.normal(loc=0, scale=1, size=5000)
esp = random.normal(loc=0, scale=0.25, size=5000)
y = -1 + (0.5 * x) - (2 * (x ** 2)) + (0.3 * (x ** 3)) + esp
# true weights
true_weight_vector = [-1, 0.5, -2, 0.3]
# --------------------- END OF DATA GENERATION -----------------------

# Plotting graph using synthetic data of Vector X and Vector Y
plt.scatter(x, y, label="stars", color="green", marker=".", s=30)
plt.title("Synthetic Dataset of Vectors")
plt.xlabel("X")
plt.ylabel("y", rotation=0)
plt.show()
# -------------- END OF SYNTHETIC DATA PLOT -------------------------
```

**Output:** The output plots a gaussian distribution of x and y points.




2. In this part we implement Adaline Neuron Learning Algorithm using batch gradient descent (BGD) formula. The code for that can be checked below:

```python
# Convert Vector X into matrix for fitting/training model
X = np.asanyarray(x).reshape(-1, 1)

# Using LinearRegression model for implementing BATCH GRADIANT DECENT in ADALINE NEURAL NETWORK.
# method is sgd - stochastic gradient descent with Minibatch = 1 act as Batch Gradiant Decent
# ets = Learning Rate | epochs = Dataset read cycles
adaline_BGD1 = LinearRegression(method='sgd', eta=0.0001, epochs=20, random_seed=0, minibatches=1)
adaline_BGD2 = LinearRegression(method='sgd', eta=0.01, epochs=20, random_seed=0, minibatches=1)
# Training Model
adaline_BGD1.fit(X, y)
adaline_BGD2.fit(X, y)
# Making predictions
prediction1 = adaline_BGD1.predict(X)
prediction2 = adaline_BGD2.predict(X)
# calculating Mean Square Error
mean_sq_error = metrics.mean_squared_error(prediction1, y)
mean_sq_error2 = metrics.mean_squared_error(prediction2, y)

eta1 = 0.0001
eta2 = 0.01
```

Then we run and test the algorithm on our synthetic dataset we have created.

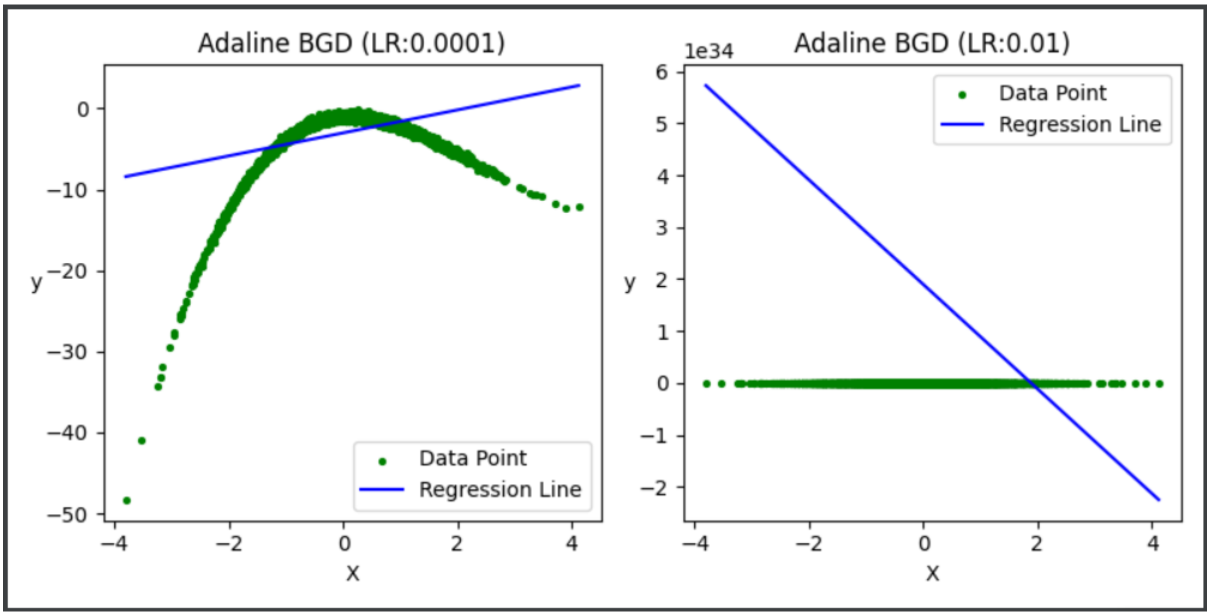
```
xp1 = np.linspace(x.min(), x.max(), 5000)
xp2 = np.linspace(x.min(), x.max(), 5000)

# Plotting graph and for both Learning Rate time for Adaline Neural network Structure.
# With Batch Gradiant Decent method

# Plotting Regression line on the graph side by side
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))
# Graph number 1
ax[0].scatter(X, y, color='green', label='Data Point', marker=".", s=30)
ax[0].plot(xp1, adaline_BGD1.predict(xp1.reshape(-1, 1)), color='blue', label="Regression Line")
ax[0].set_title("Adaline BGD (LR:" + str(eta1) + ")")
ax[0].set_xlabel("X")
ax[0].set_ylabel("y", rotation=0)
ax[0].legend()
# Graph number 2
ax[1].scatter(X, y, color='green', label="Data Point", marker=".", s=30)
ax[1].plot(xp2, adaline_BGD2.predict(xp2.reshape(-1, 1)), color='blue', label="Regression Line")
ax[1].set_title("Adaline BGD (LR:" + str(eta2) + ")")
ax[1].set_xlabel("X")
ax[1].set_ylabel("y", rotation=0)
ax[1].legend()
plt.tight_layout()
plt.show()
# ---------- Plotting Regression line on the graph side by side END --------------------
```

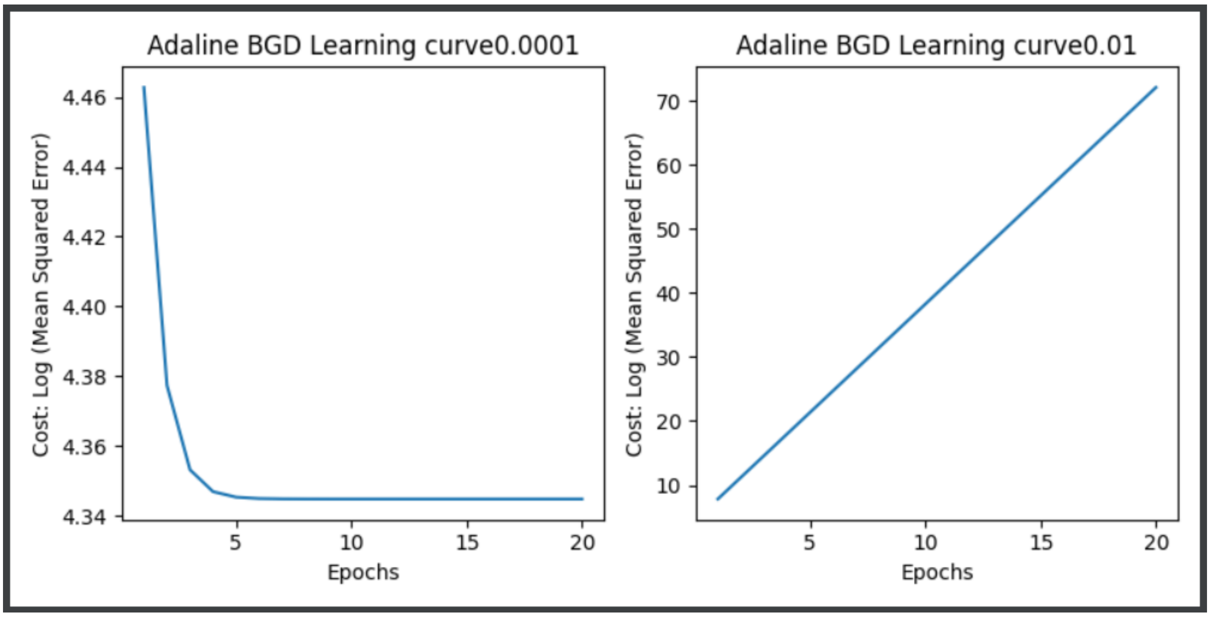
## **Output:**

In the output, you can see the predicted weights, learning rate, intercept, Slope, and mean-squared-error values respectively:

```
Adaline Batch Gradient Decent
---------------------------------------------------
    Learning Rate:  0.0001
    Intercept: 1.41
    Slope: -3.01
    MSE:  8.197191249772326

    Learning Rate:  0.01
    Intercept: -9691902210295116437873257372188672.00
    Slope: 19008377071277409457734620748447744.00
    MSE:  4.554987507602271e+68
```

The plots helps us visualize our hyper-parameters in more details and easily.




3. In this part, we are implementing Adaline Learning Algorithm using stochastic gradient descent [SGD] formula.

```python
# STOCHASTIC GRADIANT DECENT in ADALINE NEURAL NETWORK.
# Using LinearRegression model
# method is sgd - stochastic gradient descent with Minibatch = 1 act as Batch Gradiant Decent
# ets = Learning Rate | epochs = Dataset read cycles
adaline_SGD1 = LinearRegression(method='sgd', eta=0.0001, epochs=20, random_seed=0, minibatches=len(y))
adaline_SGD2 = LinearRegression(method='sgd', eta=0.01, epochs=20, random_seed=0, minibatches=len(y))
# Training Model
adaline_SGD1.fit(X, y)
adaline_SGD2.fit(X, y)
# Making predictions
prediction1 = adaline_SGD1.predict(X)
prediction2 = adaline_SGD2.predict(X)
# calculating Mean Square Error
mean_sq_error = metrics.mean_squared_error(prediction1, y)
mean_sq_error2 = metrics.mean_squared_error(prediction2, y)

eta1 = 0.0001
eta2 = 0.01
```

```python
# Printing numeric output
print("\nAdaline Stochastic Gradient Decent")
print("-----------------------------------------------------")

print("\tLearning Rate: ", eta1, end='\n')
print('\tIntercept: %.2f' % adaline_SGD1.w_, end='\n')
print('\tSlope: %.2f' % adaline_SGD1.b_, end='\n')
print('\tMSE: ', mean_sq_error, end='\n')

print("\n\tLearning Rate: ", eta2, end='\n')
print('\tIntercept: %.2f' % adaline_SGD2.w_, end='\n')
print('\tSlope: %.2f' % adaline_SGD2.b_, end='\n')
print('\tMSE: ', mean_sq_error2, end='\n')

xp1 = np.linspace(x.min(), x.max(), 5000)
xp2 = np.linspace(x.min(), x.max(), 5000)
```

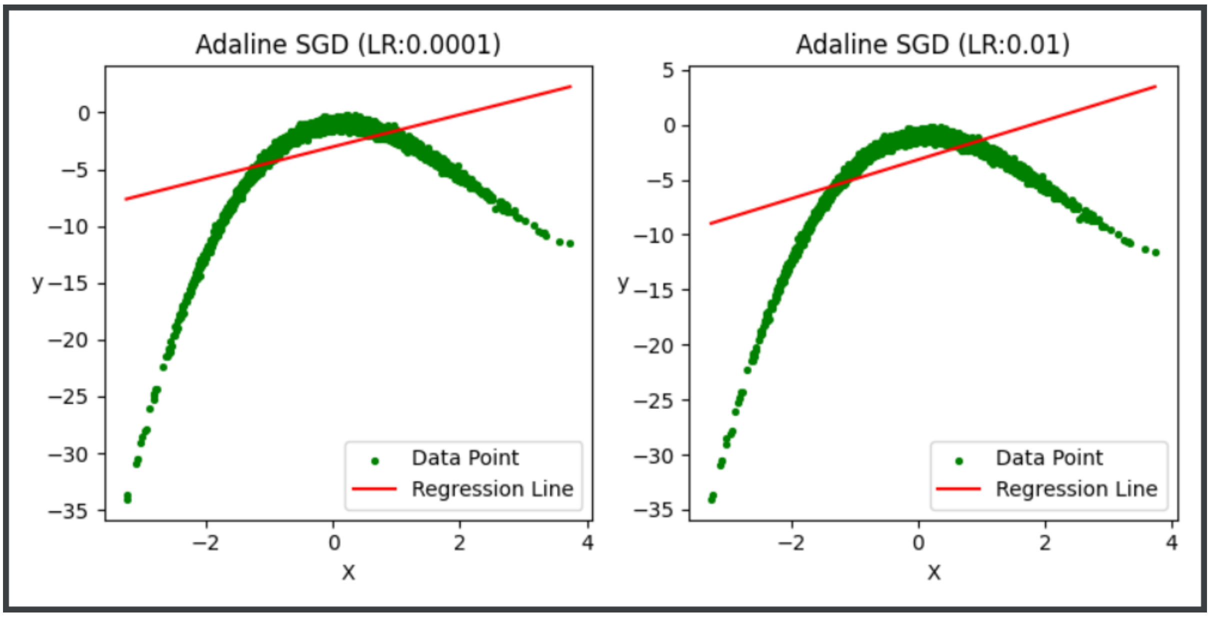
Now, we run the algorithm on our synthetic dataset.

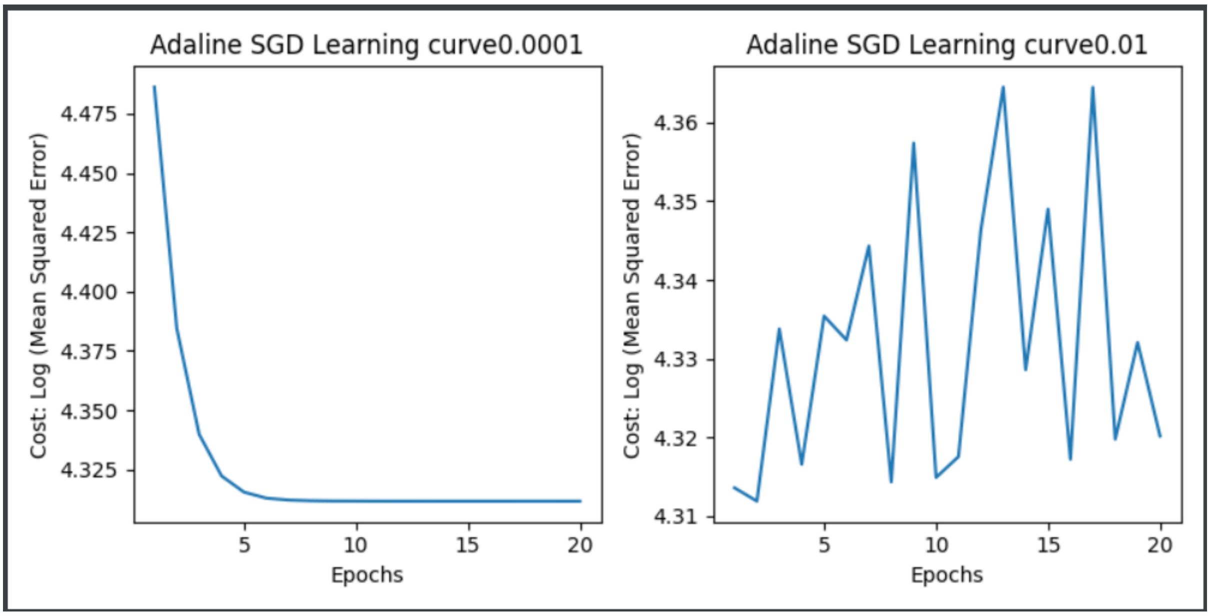
```python
# Plotting graph and for both Learning Rate time for Adaline Neural network Structure.
# With Batch Gradiant Decent method

# Plotting Regression line on the graph side by side
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))
# Graph number 1
ax[0].scatter(X, y, color='green', label='Data Point', marker=".", s=30)
ax[0].plot(xp1, adaline_SGD1.predict(xp1.reshape(-1, 1)), color='red', label="Regression Line")
ax[0].set_title("Adaline SGD (LR:" + str(eta1) + ")")
ax[0].set_xlabel("X")
ax[0].set_ylabel("y", rotation=0)
ax[0].legend()
# Graph number 2
ax[1].scatter(X, y, color='green', label="Data Point", marker=".", s=30)
ax[1].plot(xp2, adaline_SGD2.predict(xp2.reshape(-1, 1)), color='red', label="Regression Line")
ax[1].set_title("Adaline SGD (LR:" + str(eta2) + ")")
ax[1].set_xlabel("X")
ax[1].set_ylabel("y", rotation=0)
ax[1].legend()
plt.tight_layout()
plt.show()
# ----------- Plotting Regression line on the graph side by side END --------------------
```

```python
# ----------- Plotting Regression line on the graph side by side END --------------
 
# Plotting Learning rate graphs side by side
fig, ax1 = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))
# Graph number 1
ax1[0].plot(range(1, adaline_SGD1.epochs + 1), np.log10(adaline_SGD1.cost_))
ax1[0].set_xlabel('Epochs')
ax1[0].set_ylabel('Cost: Log (Mean Squared Error)')
ax1[0].set_title('Adaline SGD Learning curve' + str(eta1))
ax1[1].plot(range(1, adaline_SGD2.epochs + 1), np.log10(adaline_SGD2.cost_))
# Graph number 2
ax1[1].set_xlabel('Epochs')
ax1[1].set_ylabel('Cost: Log (Mean Squared Error)')
ax1[1].set_title('Adaline SGD Learning curve' + str(eta2))
plt.tight_layout()
plt.show()
# ---------- Plotting Learning rate graphs side by side END ---------------------
# ----------- STOCHASTIC BATCH GRADIANT DECENT END _______________________________
```

## Output:

As you can see our predicted weights, learning rate and MSE values are shown below in a comparative plot.






4. In this part we are going to Implement a Sigmoid Neural Learning Algorithm with a tanh activation function. Below is the code for that:

```python
# This program is into GOOGLE COLAB due to some GPU and Libray Loading issues in Local machine
# --------------------------------------------------------------------------------------------
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn import metrics

# Saperatly againg generating 5000 synthetic data points for this istance of code
# because it is on GOOGLE COLAB
vector_x = random.normal(loc=0, scale=1, size=5000)
vector_esp = random.normal(loc=0, scale=0.25, size=5000)
vector_y = -1 + 0.5 * vector_x - 2 * (vector_x ** 2) + 0.3 * (vector_x ** 3) + vector_esp
```

Now, running the algorithm on our synthetic dataset.

```python
# reshaping dataset to fit into model
X = vector_x.reshape(-1, 1)
Y = vector_y.reshape(-1, 1)

# initiating model with help of TensorFlow Keras
model = tf.keras.Sequential([tf.keras.layers.Dense(1, input_shape=(1,), activation='tanh')])
model.compile(optimizer=tf.keras.optimizers.SGD(0.01), loss='mean_squared_error', metrics=['mean_squared_error'])

# fitting/traning model on synthetic data points
model.fit(X, Y, epochs=30, batch_size=512)
prediction = model.predict(X)

# printing numeric data of our sigmod function
print(" \n Sigmoid Nueron using tanh activation function: \n ")
print("----------------------------------------------------------")
mean_sq_error = metrics.mean_squared_error(vector_y, prediction)
print(" \n Result:")
print("------------")
print("Mean Square Error: ", mean_sq_error)
new_data = np.linspace(vector_x.min(), vector_x.max())

# plotting graph of our synthetic data points used in sigmod function
plt.scatter(X, vector_y, color='green', label='Data Points', marker='.')
plt.plot(new_data, model.predict(new_data.reshape(-1)), color='blue', label='Regression Line')
plt.title("Sigmoid Nueron - tanh Activation")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

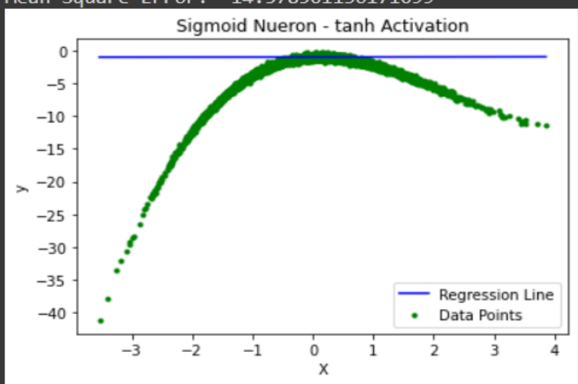
## Output:

The predicted values of weight and MSE is shown in the plot below.

## Note:

As seen, the sigmoid neuron does not give the correct output for our created dataset as the MSE value is very high.

```
 Sigmoid Nueron using tanh activation function:

-------------------------------------------------------------

 Result:
------------
Mean Square Error:  14.578501136171099
```



5. In this part, we are performing cross validation(10-fold-CV) of Batch gradient descent algorithm and the stochastic gradient descent algorithm.

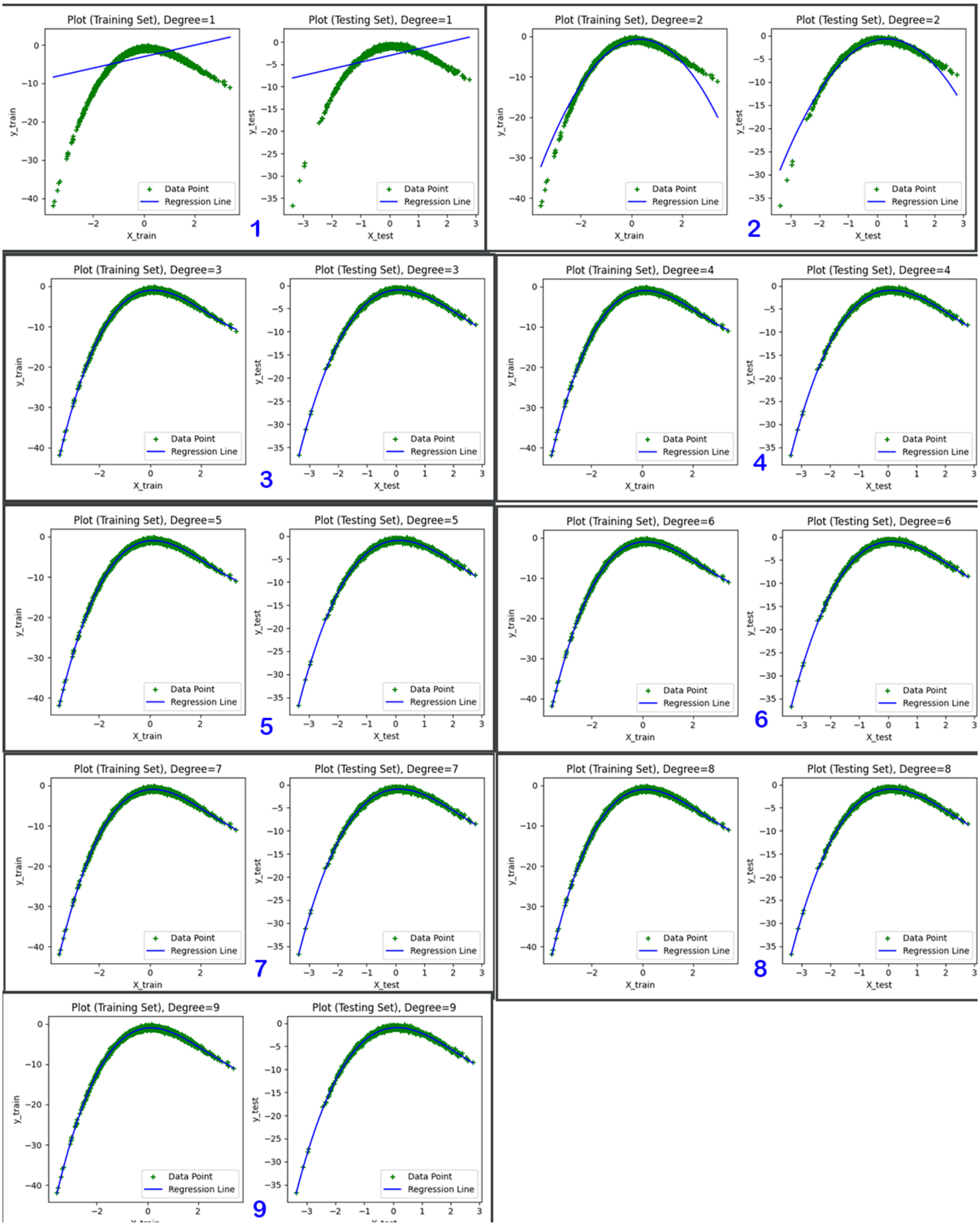
```python
# CROSS VALIDATION
# importing extra Libraries here to avoid Conflict between import names
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures

# Splitting dataset into TRAINING( size = 70% ) AND TESTING DATASET( size = 30% )
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
lm = LinearRegression()

# using 10-fold cross validation method on our modelS
cross_val = KFold(n_splits=10, shuffle=True)

min_mse = 23432142134.2343
min_degree = 1
```

```python
# loop fit and transform our split sets. also calculating, comparing and testing regression performances
for itr in range(1, 11):
    poly = PolynomialFeatures(degree=itr)
    coss_val_model = poly.fit_transform(x_train)
    poly.fit(coss_val_model, y_train)
    model = lm.fit(coss_val_model, y_train)
    scores = cross_val_score(model, coss_val_model, y_train, scoring="neg_mean_squared_error", cv=cross_val, n_jobs=1)
    mean_sq_err = np.mean(np.abs(scores))
    print("Degree: " + str(itr) + ", \nPolynomial MSE: " + str(mean_sq_err) + ", STD: " + str(np.std(scores)))
    if (min_mse > mean_sq_err):
        min_mse = mean_sq_err
        min_degree = itr

    # converting our dataset into array using numpy "asarray" and then reshaping it into matrix
    x_train_array = np.asarray(x_train).reshape(-1)
    y_train_array = np.asarray(y_train).reshape(-1)
    x_test_array = np.asarray(x_test).reshape(-1)
    y_test_array = np.asarray(y_test).reshape(-1)
    weights = np.polyfit(x_train_array, y_train_array, itr)

    # generating model with the given weights
    model = np.poly1d(weights)
    new_train = np.linspace(x_train_array.min(), x_train_array.max())
    new_test = np.linspace(x_test_array.min(), x_test_array.max(), 70)
    predict_plot_train = model(new_train)
    predict_plot_test = model(new_test)
```

```python
    # printing the weight vectors
    print("Weights:")
    for j in range(0, len(weights)):
        print("w" + str(j) + " = " + str(weights[j]))

    # plotting graphs for the regeneration performance and degree

    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

    ax[0].scatter(x_train, y_train, color='green', label='Data Point', marker='+')
    ax[0].plot(new_train, predict_plot_train, color='blue', label='Regression Line')
    ax[0].set_title("Plot (Training Set), Degree=" + str(itr))
    ax[0].set_xlabel("X_train")
    ax[0].set_ylabel("y_train")
    ax[0].legend()

    ax[1].scatter(x_test, y_test, color='green', label='Data Point', marker='+')
    ax[1].plot(new_test, predict_plot_test, color='blue', label='Regression Line')
    ax[1].set_title("Plot (Testing Set), Degree=" + str(itr))
    ax[1].set_xlabel("X_test")
    ax[1].set_ylabel("y_test")
    ax[1].legend()

    plt.tight_layout()
    plt.show()

print("Best Values:   ")
print("Mean Square Error: " + str(min_mse))
print("Best Degree: " + str(min_degree))
# ----------------- CROSS VALIDATION ENDS -----------------------
```

## Output:

```
Best Values:
Mean Square Error: 0.062340247318043704
Best Degree: 4
```




## Observation:

During the experiment, we tried different values of learning rates and epochs to check their effects on the performance of the given algorithms. With many attempts of trail and errors, we concluded the optimal values for our experiment.

We implemented BGD it is found that learning rate is too small in this method that is 0.01. Before implementation of cross validation on Adaline and sigmoid the mean square error is high. After implementation of 10-fold cross validation method the mean square error is deduced to 0.06234024731. So, it can be said that, the solution is more optimized after implementing cross validation.

**Question: 2 In this question you are to create some simulated data sets and then use the Perceptron neuron to perform some classification.**

- **Randomly create 2500 data-points (x, y)'s of class -1 to lie one side of the function f above and 2500 data-points (x, y)'s of class +1 to lie on the other side of the function. Indeed, here, you are not required to create your data using the function f above; you can use any function you want, as long as it is a simple linearly separable function of your choice to be used to separate 5000 data points into two classes (I have mentioned the function above simply because you have it already).**

- **Implement the Perceptron learning algorithm and run it on your synthetic data set. Obtain the best Perceptron model via any cross-validation method of your choice. Use your creativity to tell me anything about your Perceptron: for example, how does the performance (speed and accuracy) vary when changing the learning rate, or when varying the size of the size of the training and test sets?**

**Solution**

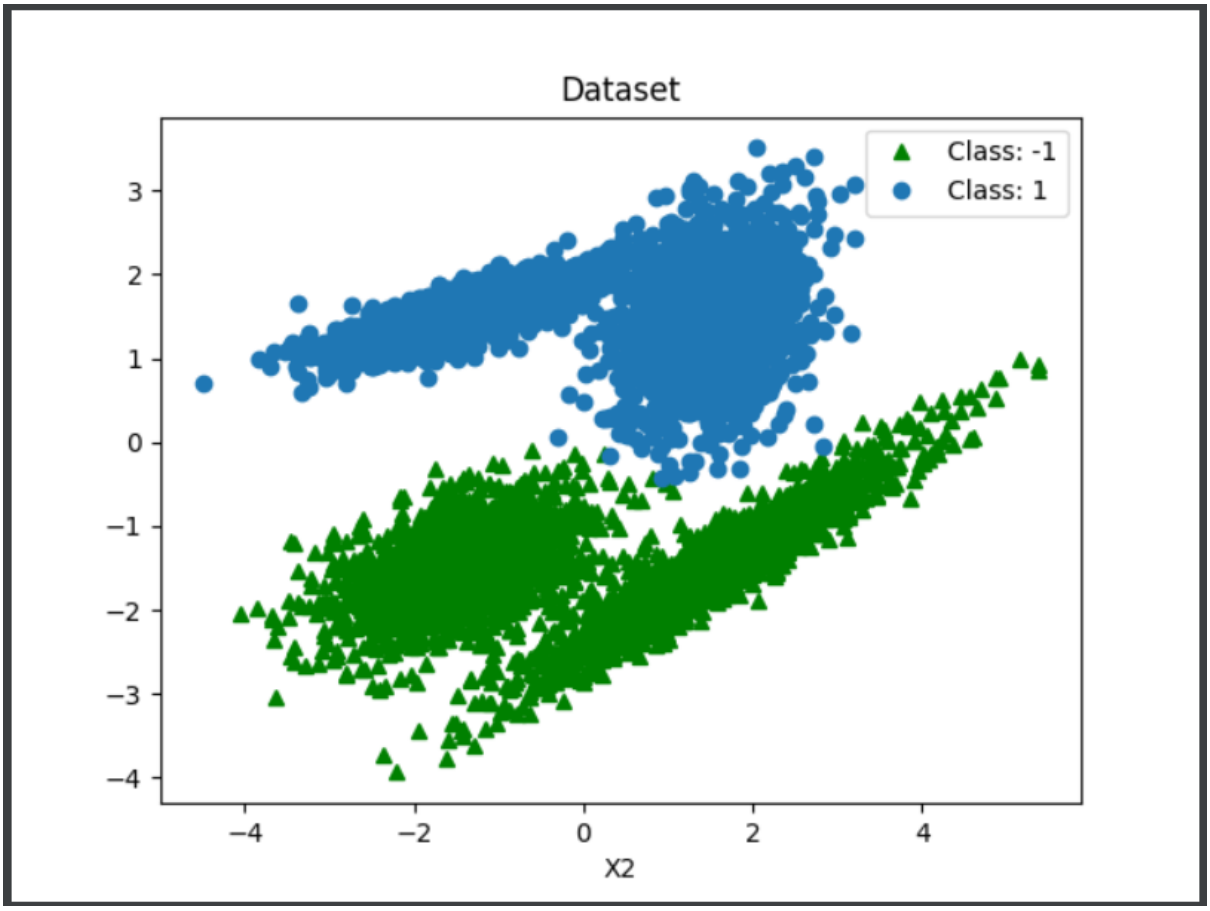
1. Import all the required libraries and randomly create 2500 data-points (x, y)'s of class -1 to lie one side of the function f above and 2500 data-points (x, y)'s of class +1 to lie on the other side of the function.

```python
#  Required Libraries for Program
# Suggested running this program on PyCharm IDE
from sklearn.datasets import make_classification
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plot
import numpy as num
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import StandardScaler
from sklearn import metrics

# crate 5000 data points into two classes 2500 each class are represented by 0 and 1 in arra
# class 0 = 2500 data points | class 1 = 2500 data points
X, y = make_classification(
    n_samples=5000, n_features=2,
    n_redundant=0, n_informative=2,
    n_clusters_per_class=2, class_sep=1.5,
    flip_y=0, random_state=0, shuffle=False)

# converting Class 0 into class -1 as required by the question
for itr, j in enumerate(num.asarray(y)):
    if j == 0:
        y[itr] = -1

# counting and separating class 1 and class -1
elements, elements_counts = num.unique(y, return_counts=True)
print("\nFrequency of unique class of the array:\n")
print(num.asarray((elements, elements_counts)))

```

```python

# plotting graph for the class -1 and class 1
print("\nDataset of Class 1 and Class -1:\n ")
plot.plot(X[:, 0][y == -1], X[:, 1][y == -1], 'g^', label='Class: -1')
plot.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'o', label="Class: 1")
plot.title("Dataset")
plot.xlabel("X1")
plot.xlabel("X2")
plot.legend()
plot.margins()
plot.show()
# ---------------- Classification END ---------------------------------------

```

**<u>Output:</u>**

We can see a graph of separate data points of blue and green colour.




2. In this part, we are implementing Perceptron learning algorithm to obtain the best Perceptron model via cross-validation method

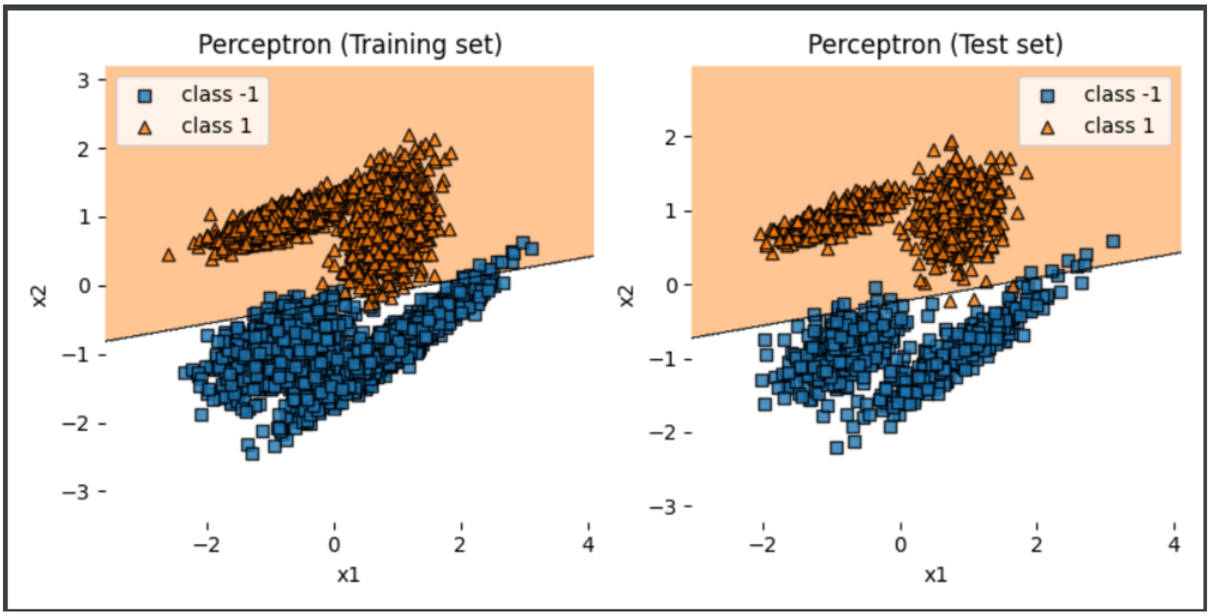
```
# ---------------------------------------------------------------------------------
# Implementing the perceptron on the data set created above
# Splitting class into train test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Scaling, fitting and transforming our data set into matrix for our model to reduce shape issues
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training perceptron model using the training set we have creates
model = Perceptron(random_state=42, alpha=0.01, eta0=0.2, max_iter=100)
model.fit(X_train, y_train)

# calculating performance of perceptron on training set
accuracy1 = cross_val_score(estimator=model, X=X_train, y=y_train, cv=10)
print("\nAccuracy and Variance on the Training dataset of Perceptron :\n")
accuracy2 = accuracy1.mean() * 100
print('Mean Accuracy: %.2f' % accuracy2, '%')
print("Standard Deviation: ", accuracy1.std())

print("\nResult of Test dataset: \n")
# Predicting the Test set results of perceptron
prediction = model.predict(X_test)
```

```
# Creating table to present the performance of a classification model
# confusion matrix
con_matrix = metrics.confusion_matrix(y_test, prediction)
print("Confusion Matrix:\n ", con_matrix)
print("{0}".format(metrics.classification_report(y_test, prediction)))
testing_accuracy = metrics.accuracy_score(y_test, prediction) * 100
print('Accuracy:%.2f' % testing_accuracy, "%\n")

# plotting graph for the perception's result
fig, axes = plot.subplots(nrows=1, ncols=2, figsize=(8, 4))
fig1 = plot_decision_regions(X_train, y_train, clf=model, ax=axes[0], legend=0)
fig2 = plot_decision_regions(X_test, y_test, clf=model, ax=axes[1], legend=0)

axes[0].set_title('Perceptron (Training set)')
axes[0].set_xlabel('x1')
axes[0].set_ylabel('x2')
axes[1].set_title('Perceptron (Test set)')
axes[1].set_xlabel('x1')
axes[1].set_ylabel('x2')

holder, labels = fig1.get_legend_handles_labels()
fig1.legend(holder, ['class -1', 'class 1'])
fig2.legend(holder, ['class -1', 'class 1'])
plot.tight_layout()
plot.show()
# -------------------------------- PERCEPTRON END ------------------------------
```

## Output:

As seen below, the frequency of our unique class of the array is shown below. Also, we have achieved an accuracy of 99.01% with a standard deviation of 0.004 value. The results on the test datasets along with a confusion matrix is given below:

```
Python Console

Frequency of unique class of the array:

[[  -1    1]
 [2500 2500]]

Dataset of Class 1 and Class -1:


Accuracy and Variance on the Training dataset of Perceptron :

Mean Accuracy: 99.01 %
Standard Deviation:  0.004470147897130724

Result of Test dataset:

Confusion Matrix:
  [[601  15]
 [  3 631]]
              precision    recall  f1-score   support

          -1       1.00      0.98      0.99       616
           1       0.98      1.00      0.99       634

    accuracy                           0.99      1250
>>>
```




By using the perceptron learning algorithm datapoint has been separated in -1 and 1. After splitting the dataset in train and test data and implementing the CV method the data set become more accurate. As shown in output above the data point is plotted below and above the slope more accurately i.e. 99.01%.

Thank You