CSE353 Assignment 4 Report

Chengzhi Dong
ID 112890166

Due November 16 2021

1 Introduction

For assignment 4, I implemented Linear Regression algorithm, Perceptron Learning algorithm, Logistic Regression algorithm, and Logistic Regression algorithm with Stochastic Gradient Descent on a data set with 40 training samples. After, I obtained the $w_{LinearRegression}$, $w_{LogisticRegression}$, and $w_{SGDLogisticRegression}$ from these algorithms, I apply them to the data set and check how well they classify the data by compute their error rates. Also, I try to run these algorithm under different conditions, such as use $w_{LinearRegression}$ and zero vector as w_0 to initialize the algorithms, try different learning rates in the Logistic Regression, and change different the number of samples used in the stochastic gradient computing. I observed and compared the behaviors of these algorithms under different conditions in terms of the computational cost (number of iterations and running time) and accuracy (error rate).

2 Method

In the python code, I imported the numpy, matplotlib.pyplot, random, and time libraries. Then, I loaded the data from "X.txt" and "Y.txt" into two matrix named x_{data} and y_data. Also, I find the sample size and the demensions of each x data.

I defined a Sign function, a Sigmoid function, and count error function. The sign funcion takes an input s and returns -1 if s is negative, 1 if s is positive, 0 otherwise. The sig function takes an input s, calculate and return $\frac{1}{1+e^{-s}}$. The count error function takes a decision boundary w, a data set x_data, and the labels y_data. It loops through the x_data and compare the $\text{sign}(w^T x_n)$ with corresponding y_n to determine if a error exist. Using $sign(w^T x_n) \neq y_n$ to check for error always work because $w_{LinearRegression}$, $w_{LogisticRegression}$, and $w_{SGDLogisticRegression}$ can be apply to binary classification by modify the hypothesis function to:
$h(x) = sign(w^T_{LinearRegression} x)$, $h(x) = sign(w^T_{LogisticRegression} x)$, and $h(x) = sign(w^T_{SGDLogisticRegression} x)$

Also, I defined functions for Linear Regression algorithm, PLA algorithm, Logistic Regression algorithm, SGD Logistic Regression algorithm, and visualization. Similar to Assignment3, the visualization function separate the data with $+1$ and -1 labels and plot them with blue and red color dots. Also, it find the min and max x_1, use them to find the x_2 on the decision boundary, and draw the line of decision boundary. The visualization also works for Linear Regression and Logistic Regression because $w_{LinearRegression}$, $w_{LogisticRegression}$, and $w_{SGDLogisticRegression}$ can be apply to binary classification by modify the hypothesis function to:
$h(x) = sign(w^T_{LinearRegression}x)$, $h(x) = sign(w^T_{LogisticRegression}x)$, and $h(x) = sign(w^T_{SGDLogisticRegression}x)$

Then the following code is the actual implementation, training and testing part for Part 1, 2, 3 of Assignment 4.

For Part 1, I mainly implemented the Linear Regression algorithm and Perceptron Learning algorithm (PLA) to compute the decision boundary. For part (a), I defined a function for Linear Regression algorithm, which takes data set x_data and the ground truth label y_data. The function calculates and returns the closed form solution for the Linear Regression, which is $w^* = (X^TX)^{-1}X^TY$. I called the function to calculate the $w_{LinearReg}$. In part (b), I called the visualization function to plot the data and $w_{LinearReg}$. Then I called the count_error function to count the mistakes that $w_{LinearReg}$ has over the data set. For part (c), I defined a function for PLA, which takes an initial decision boundary w_0, x_data and y_data. Start with w_0, PLA will loop through the data in a naive order, and if it found a mistake by $sign(w^Tx_n) \neq y_n$, then it will fix w_{PLA} by $w_{PLA} + y_nx_n$ and begin a new iteration. The loop ends when no mistake is found. The function also record the total number of iteration it take to find the optimal decision boundary w_{PLA}. The function return both the optimal decision boundary (w_{PLA}) and the number of iteration ($time$) as a dictionary. I call the PLA function with $w_{LinearReg}$ as the initialization on w_{PLA} and with zero vector as the initialization on w_{PLA}. Then I call the visualization function and count error function to plot the result and compare the error rate and number of iterations.

For Part 2, I mainly implemented the Logistic Regression algorithm. I defined a function for the Logistic Regression algorithm, which takes initial w_0, leaning rate α (step), x_data, y_data, and max_iteration as inputs. The Logistic Regression algorithm will loop for at most max_iteration. For each iteration, it will use gradient descent to find the optimal $w_{LogisticReg}$: $w_{t+1} = w_t + \alpha\frac{1}{N}\sum_{n=1}^{N} sig(-y_nw_t^Tx_n)(y_nx_n)$ where the Gradient of Logistic Regression Error $\nabla E(w_t) = \frac{1}{N}\sum_{n=1}^{N} sig(-y_nw_t^Tx_n)(y_nx_n)$. The algorithm end either when it reaches max iteration or $\|\nabla E(w_t)\|_2^2 \leq \varepsilon$ which in this case $\varepsilon = 0.00001$ and $\|\nabla E(w_t)\|_2^2$ is also the dot product of $\nabla E(w_t)$ with itself. The function also

record the number of iteration it takes, and it returns both $w_{LogisticReg}$ and the number of iteration as a dictionary. I called the Logistic Regression algorithm function with $w_0 = [0, 0, 0]$ and $\alpha = 5$ in part(a) and obtained the $w_{LogisticReg}$ and the number of iterations. Also, I use the *time* library to record and compute the running time of the algorithm. In part(b), I called the visualization function to plot the $w_{LogisticReg}$ and the data. I use the count error function to compute the error rate. In part(c), I call the Logistic Regression algorithm function with $w_0 = w_{LinearReg}$ that I obtained from Part1 and $\alpha = 5$. Then I use visualization, count error function, time library to compare the result with part(a and b). For part(d), I called the Logistic Regression algorithm function with $w_0 = [0, 0, 0]$ and $w_0 = w_{LinearReg}$ and different learning rates $\alpha = 0.5, 50, 500$. Then I compare the result with the error rate, number of iteration, running time in part(a, b and c).

For Part 3, I mainly implemented the Logistic Regression algorithm with Stochastic Gradient Descent. I defined a function for the SGD Logistic Regression algorithm, which takes initial w_0, leaning rate α (step), number of randomly selected samples k, x_data, y_data, and max_iteration as inputs. The SGD Logistic Regression algorithm work similar to the Logistic Regression algorithm, but instead of using all samples in Gradient Descent, it only randomly choose k samples for each iteration of Stochastic Gradient Descent. The function create a list of order of data. In each iteration, it use the random library to shuffle the order and select k order, then it uses Stochastic Gradient Descent with these k samples to find the optimal $w_{SGDLosgisticReg}$ by $w_{t+1} = w_t + \alpha \frac{1}{k} \sum_{n=1}^{k} sig(-y_n w_t^T x_n)(y_n x_n)$. The Error of Stochastic Gradient Descent is $\nabla E(w_t) = \frac{1}{k} \sum_{n=1}^{k} sig(-y_n w_t^T x_n)(y_n x_n)$. The algorithm ends the loop either when it reaches max iteration or $\|\nabla E(w_t)\|_2^2 \leq \varepsilon$ which in this case $\varepsilon = 0.00001$ and $\|\nabla E(w_t)\|_2^2$ is also the dot product of $\nabla E(w_t)$ with itself. The function also record the number of iteration it takes, and it returns both $w_{SGDLogisticReg}$ and the number of iteration as a dictionary. In part(a), I obtained the $w_SGDLogisticReg$ and the number of iterations by calling the SGD Logistic Regression algorithm function with w_0= zero vector, $\alpha = 5, k = 15$. Also, I used the time library to record and compute the running time of the algorithm. I compared the computational cost with the logistic regression without SGD. In part(b), I used the visualization and count error function to plot the $w_SGDLogisticReg$ with the data and to compute the error rate. In part(c), I try to use the SGD Logistic Regression algorithm function with different $k = 5, 25, 35$ and observe the change in terms of error rate, number of iteration and running time.

3 Experiment

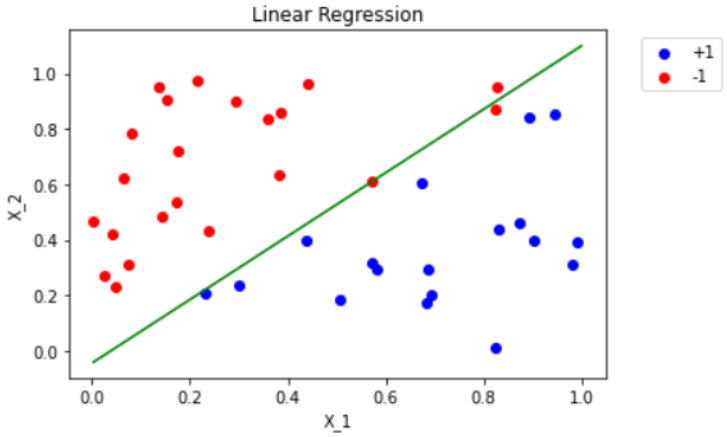
Part 1 (a):
Below are my results for part 1 (a).

```
Part1(a):
w_LinearRegression:[-0.07882655  2.04885018 -1.79061198]
```

Since there is a closed form solution for Linear Regression, no loops are needed, so the algorithm computed $w_{LinearRegression}$ very fast.

Part 1 (b):
Below are my results for part 1 (b).

```
Part1(b):
```

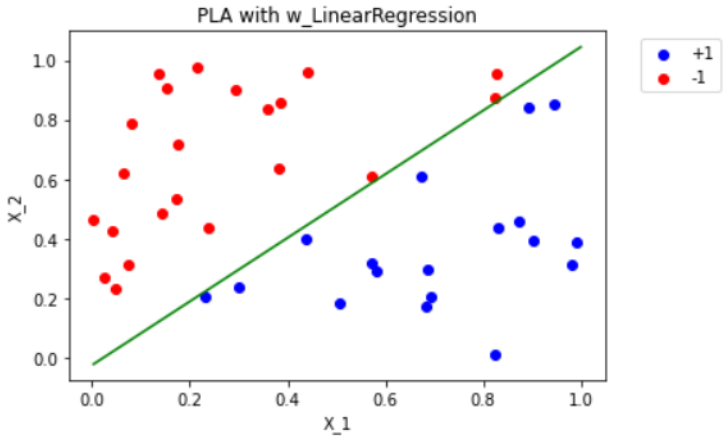



```
Number of Error for w_LinearRegression: 1
Error Rate for w_LinearRegression: 0.025
```

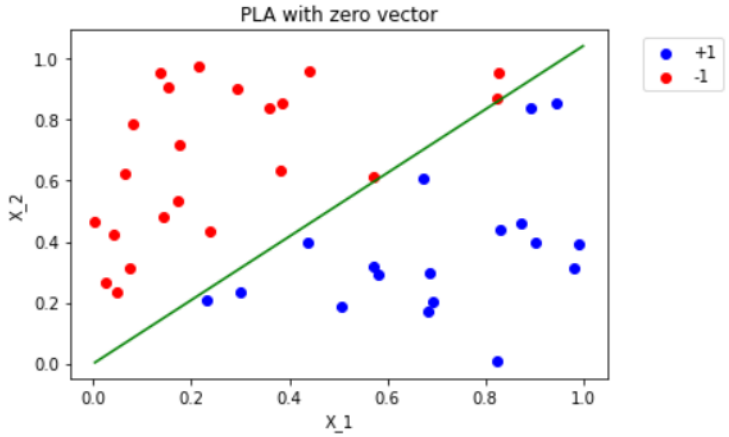
The result of apply $w_{LinearRegression}$ in binary classification is decent. Although there is one error and the error rate is 0.025, $w_{LinearRegression}$ is a good start point for binary classification because it require a small amount of time to compute and has a low error rate.

Part 1 (c):
Below are my results for part 1 (c).

```
Part1(c):
Initialize w with w_LinearRegression:
```




```
w_PLA:[-0.07882655  3.35325018 -3.13411198]
Error Rate: 0.0
Iterations: 13
============================================================
Initialize w with zero vector:
```




```
w_PLA:[ 0.      3.8812 -3.7239]
Error Rate: 0.0
Iterations: 31
```

Comparing initialization of w_{PLA} with $w_{LinearRegression}$ and a zero vector, PLA that initialized with $w_{LinearRegression}$ obtained a w_{PLA} that has the same error rate of 0 with w_{PLA} that obtained by PLA that initialized with a zero vector. Therefore, the final w_{PLA} might be different but the accuracy will be the same no matter what w the PLA initialized with. However, PLA that initialized with

$w_{LinearRegression}$ only takes 13 iteration to obtained the optimal w_{PLA} while PLA that initialized with zero vector takes 31 iterations to obtained the optimal w_{PLA}. It is better to start with $w_{LinearRegression}$ because it causes the PLA to iterate less time which means faster.

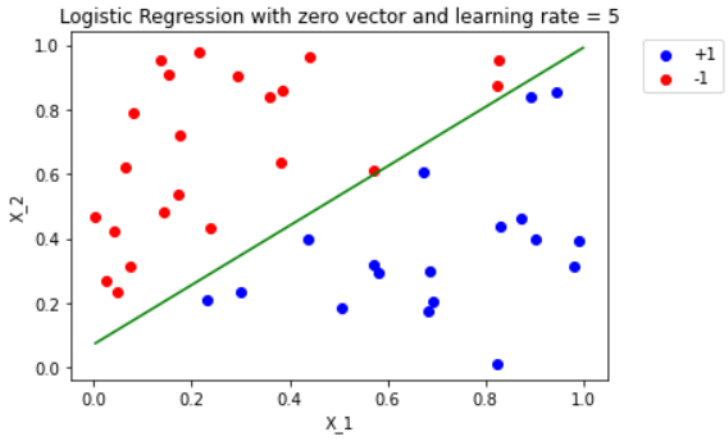
Part 2 (a):
Below are my results for part 2 (a).

```
Part2(a)
Logistic Regression that initialize with zero vector and has a learning rate of 5
w_LogisticRegression: [  1.2891256   16.56967867 -18.01727107]
============================================================
```

This the $w_{LogisticRegression}$ the algorithm computed with zero vector as the initialization on $w_{LogisticRegression}$ and $\alpha = 5$.

Part 2 (b):
Below are my results for part 2 (b).

```
Part2(b)
```

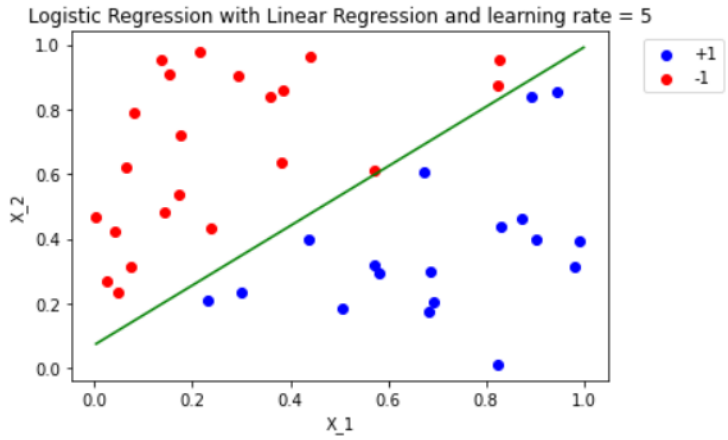



```
Error Rate: 0.0
Iterations: 586
Runtime: 0.19448184967041016 seconds
============================================================
```

The $w_{LogisticRegression}$ from Part2 (a) has error rate of 0. The algorithm iterated 586 times and ran 0.1945 seconds. The $w_{LogisticRegression}$ did a well job on classify the data set because it has no error.

Part 2 (c):
Below are my results for part 2 (c).

```
Part2(c)
```

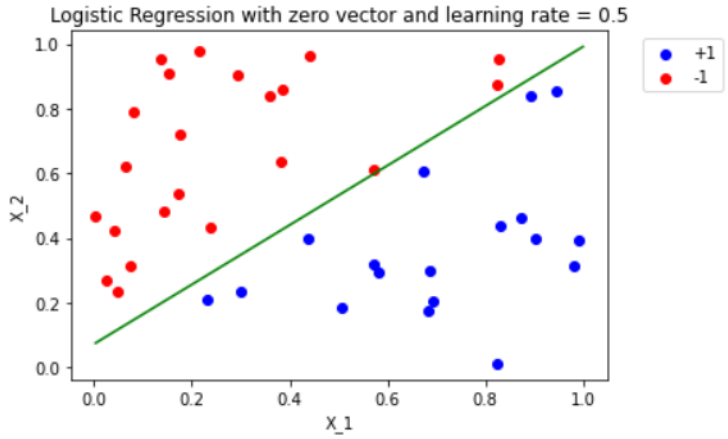



```
w_LogisticRegression:[  1.28855179  16.56460268 -18.01140722]
Error Rate: 0.0
Iterations: 580
Runtime: 0.21452069282531738 seconds
============================================================
```

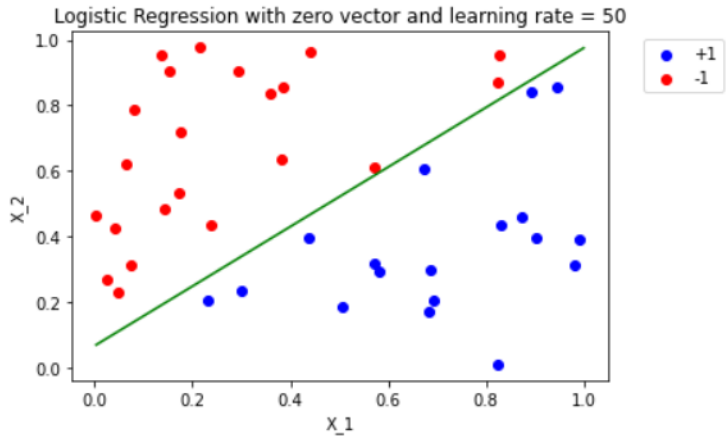
When use $w_{LinearRegression}$ from Part1 as the initialization on $w_{LogisticRegression}$, the error rate is still 0, but the computational cost is slightly better. The algorithm has 480 iteration which is 4 iterations less, and the running time is 0.2145 seconds which is about 0.02 seconds faster.

Part 2 (d):
Below are my results for part 2 (d).

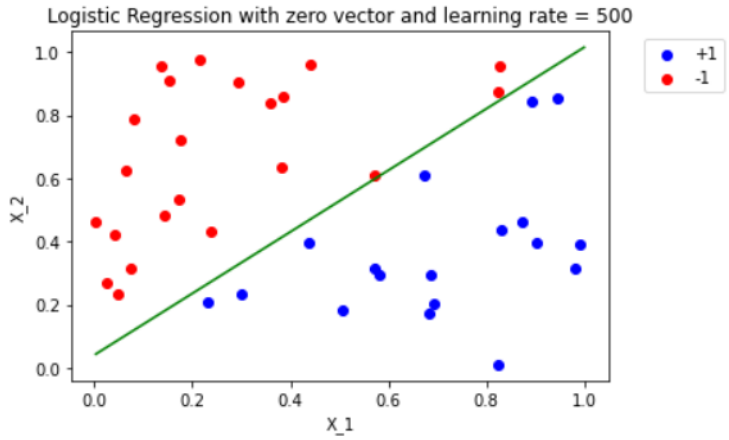
```
Part2(d)
Initialize w_0 with zero vector and different learning rates
```




```
w_LogisticRegression:[  1.2884465   16.56378952 -18.01044708]
Error Rate: 0.0
Iterations: 5862
Runtime: 1.8960630893707275 seconds
============================================================
```

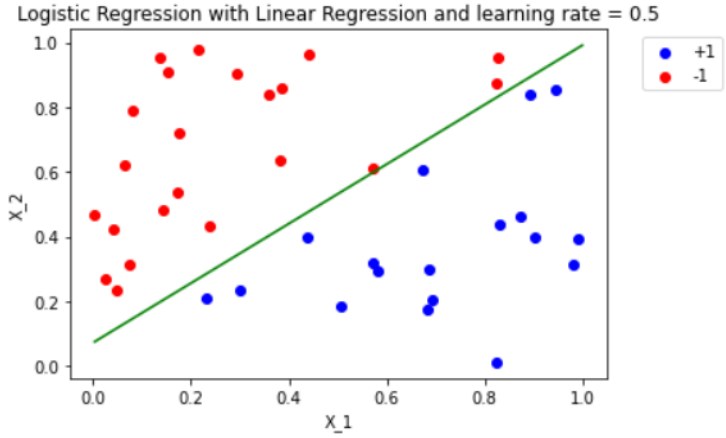



```
w_LogisticRegression:[  2.47049619  33.19997855 -36.57641007]
Error Rate: 0.0
Iterations: 70
Runtime: 0.02916693687438965 seconds
============================================================
```

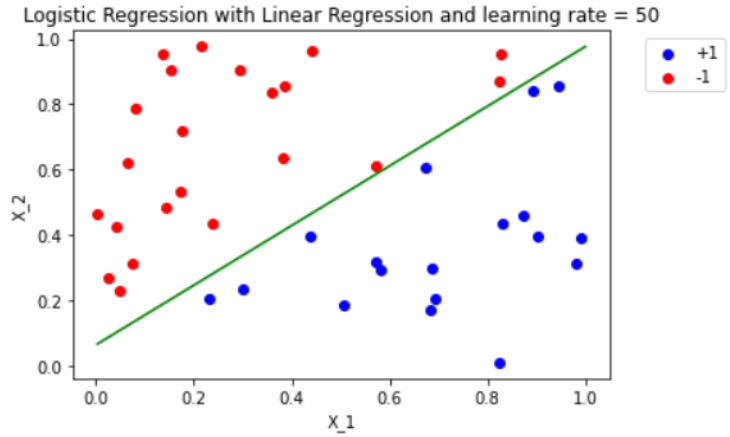



```
w_LogisticRegression:[  14.00050038  329.92017438 -339.13805362]
Error Rate: 0.0
Iterations: 32
Runtime: 0.017064809799194336 seconds
================================================================
```

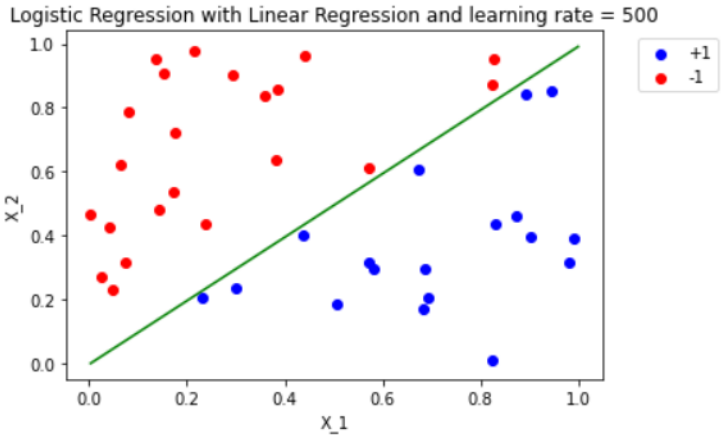
Part2(d)
Initialize w_0 with Linear Regression and different learning rates




```
w_LogisticRegression:[  1.28842569  16.56360537 -18.01023436]
Error Rate: 0.0
Iterations: 5804
Runtime: 1.842651128768921 seconds
================================================================
```




```
w_LogisticRegression:[  2.37533852  33.38999873 -36.63735223]
Error Rate: 0.0
Iterations: 54
Runtime: 0.02534174919128418 seconds
============================================================
```




```
w_LogisticRegression:[  -1.39422886  336.39908171 -338.07775853]
Error Rate: 0.0
Iterations: 20
Runtime: 0.008976936340332031 seconds
============================================================
```

In Part3 (d), I have try various learning rate(α): 0.5, 50, and 500 with both initialization with zero vector and $w_{LinearRegression}$. Overall, as the learning rate α increase from 0.5 to 500, the number of iterations the algorithm takes decrease from around 5800 iterations to around 20 iterations. And the run time decreased from about 1.88 seconds to about 0.01 seconds. the learning rate α increase, the algorithm is exponentially faster.

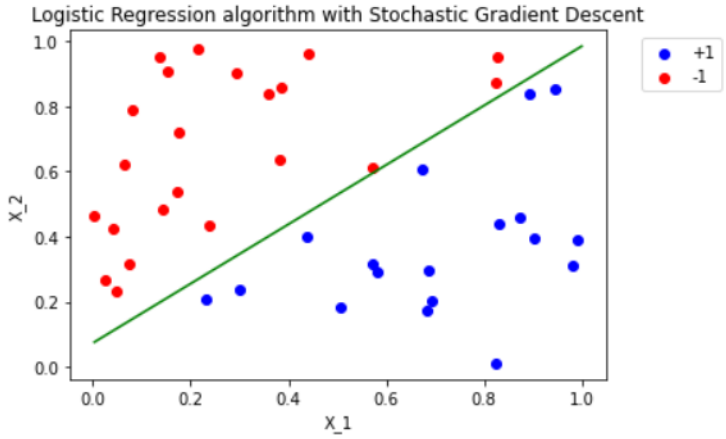
Part 3 (a):
Below are my results for part 3 (a).

```
Part3(a)
w_0 = [0, 0, 0] ; learning rate = 5 ; k = 15
w_SGDLogisticRegression:[  1.27166365  15.84239606 -17.37126448]
Iterations: 504
Runtime: 0.08427572250366211 seconds
============================================================
```

Using Logistic Regression SGD algorithm with $w_0 = [0, 0, 0]$, learning rate $\alpha = 5$, and $k = 15$. The algorithm takes 504 iterations and 0.0843 seconds. Compared to Logistic Regression without SGD which takes 586 iterations and 0.1944 seconds, Logistic Regression SGD is faster.

Part 3 (b):
Below are my results for part 3 (b).

```
Part3(b)
```

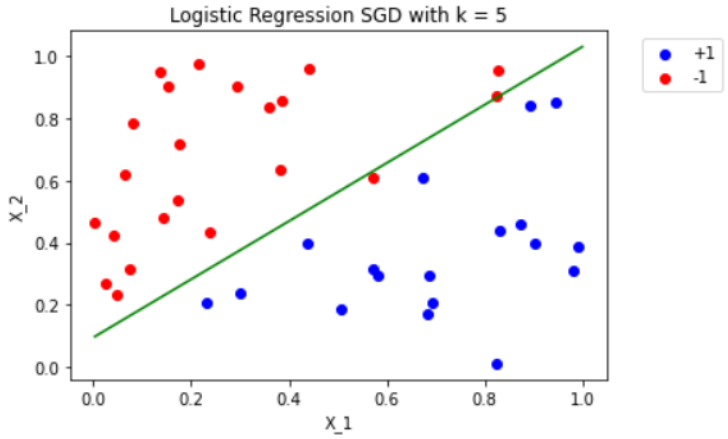



```
Error Rate: 0.0
============================================================
```

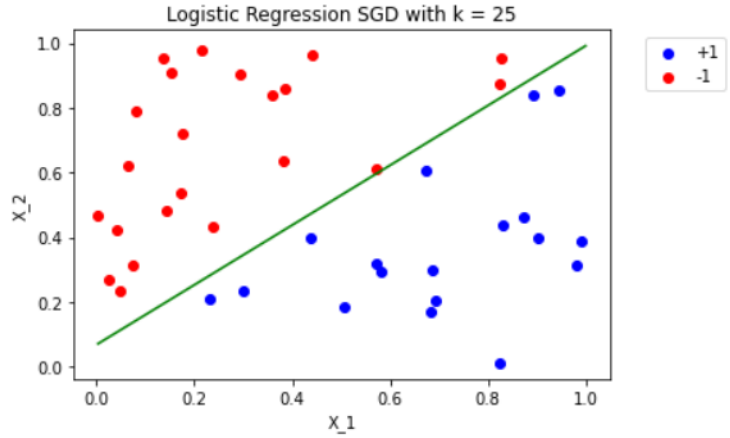
With sample size of $k = 15$, apply $w_{LogisticRegressionSGD}$ on classification of the data set results an error rate of 0, which is same as the Logistic Regression without SGD. In this case the Logistic Regression SGD algorithm is better because it computes faster and gives same accurate result.

Part 3 (c):
Below are my results for part 3 (c).

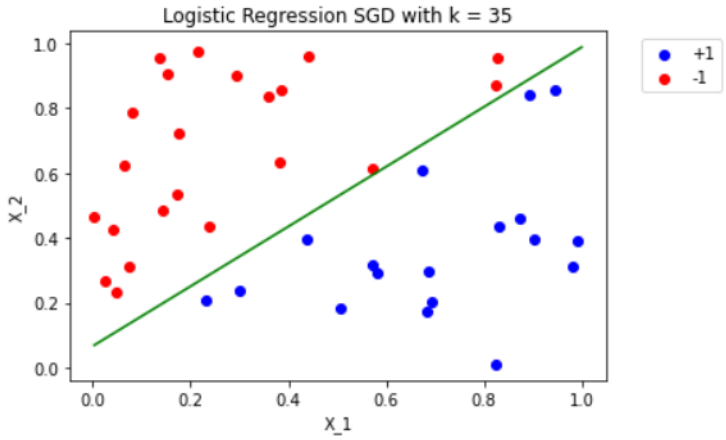
Part3(c)




```
w_SGDLogisticRegression:[  1.13371892  11.19215487 -11.94608797]
Error Rate: 0.025
Iterations: 182
Runtime: 0.014992952346801758 seconds
============================================================
```




```
w_SGDLogisticRegression:[  1.17126745  16.01943573 -17.33683534]
Error Rate: 0.0
Iterations: 529
Runtime: 0.12052607536315918 seconds
============================================================
```




```
w_SGDLogisticRegression:[  1.20705557  16.60364241 -18.03148468]
Error Rate: 0.0
Iterations: 587
Runtime: 0.1776256561279297 seconds
============================================================
```

In Part3 (c), I have try various $k = 5, 25, 35$ with same $w_0 = [0, 0, 0]$ and $\alpha = 5$. As I observed, if the k is too small like 5, then the algorithm will run faster but the result is not very accurate. When $k = 5$, the error rate of $w_{LogisticRegressionSGD}$ is 0.025, the algorithm only takes 182 iteration and 0.015 seconds of run time. When $k = 35$, the error rate is 0, the algorithm takes 587 iterations and 0.178 seconds of run time. Overall, as the k increases, the accuracy and computational cost will also increase.

4 Discussion

For the loop order of PLA, I initially use the random order to check the mistake of w on data. However, when I compare the results in term of computational cost, between initialization with $w_{LinearRegression}$ and zero vector. The number of iteration and algorithm running time is not stable due to random order of correcting mistakes. Therefore, I changed the order to naive order to obtain a more stable results.

For ε in $\|\nabla E(w_t)\|_2^2 \leq \varepsilon$, I have tried number like 0.01, which is not enough because it only takes the algorithm a few iteration to obtain the w but the w is not accurate, and did a horrible job on classifying the data. Therefore, I changed the ε to 0.00001. Also, for the max iteration, I tried small number like 100 and large number like 10000, then after several experiments with the code I found that 1000 is the suitable max iteration.