OR 610-001: HW1 Solution

Abhishek Shambhu

1. As a result of medical examination, one of the tests revealed a serious illness in a person. This test has a high precision of 99% (the probability of a positive response in the presence of the disease is 99%, the probability of a negative response in the absence of the disease is also 99%). However, the detected disease is quite rare and occurs only in one person per 10,000. Calculate the probability that the person being examined does have an identified disease.

(1) The solution to the question can easily be calculated using Bayes Theorem:

$$P(A/B) = \frac{P(A) \times P(B/A)}{P(B)}$$

P(A) is the probability of event A
In our case, A is the event that you have this disease

P(B) is the probability of event B
In our case, B is the event that you test positive.

Here, $P(B/A) = 0.99$

$P(A) = \frac{1}{10000} = 0.0001$

$P(B) = P(B/A) \times P(A) + P(B/\text{not } A) \times P(\text{not } A)$

$= 0.99 \times \frac{1}{10000} + 0.01 \times \frac{9999}{10000}$

$= 0.000099 + 0.009999$

$= 0.010098$

$P(A/B) =$ Probability of a person being examined does have an identified disease.

$= \frac{P(B/A) \cdot P(A)}{P(B)}$

$= \frac{0.99 \times 0.0001}{0.010098}$

$\boxed{P(A/B) = 0.009803921568}$

2. Consider the following probabilistic model. The student does poorly in a class ($c = 1$) or well ($c = 0$) depending on the presence / absence of depression ($d = 1$ or $d = 0$) and weather he/she partied last night ($v = 1$ or $v = 0$) . Participation in the party can also lead to the fact that the student has a headache ($h = 1$). As a result of poor student's performance, the teacher gets upset ($t = 1$). The probabilities are given by:

$p(c = 1\|d, v)$	v	d
0.999	1	1
0.9	1	0
0.9	0	1
0.01	0	0

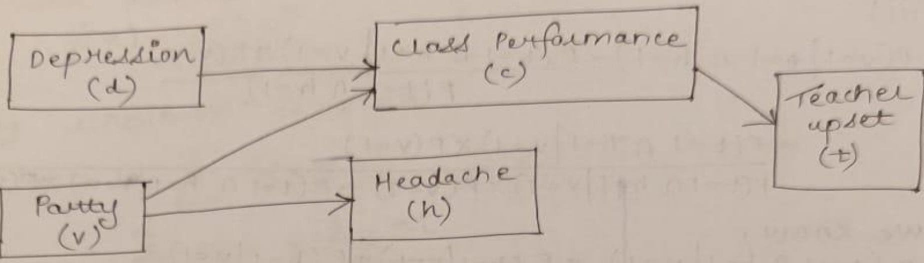
$p(h = 1\|v)$	v
0.9	1
0.1	0

$p(t = 1\|c)$	c
0.95	1
0.05	0

$p(v = 1) = 0.2$, and $p(d = 1) = 0.4$.

Draw the causal relationships in the model. Calculate $p(v = 1|h = 1)$, $p(v = 1|t = 1)$, $p(v = 1|t = 1, h = 1)$.

(2) Causal relationship model :



(i)

$P(h=1|v=1) = 0.9$

$P(v=1) = 0.2$

$P(h=1) = P(h=1|v=1) \times P(v=1) + P(h=1|v=0) \times P(v=0)$

$= 0.9 \times 0.2 + 0.1 \times 0.8$

$= 0.18 + 0.08$

$= 0.26$

$$P(v=1|h=1) = \frac{P(h=1|v=1) \times P(v=1) \cancel{+ P(h=1|v=0) \times P(v=0)}}{P(h=1)}$$

$$= \frac{0.9 \times 0.2}{0.26}$$

$\boxed{P(v=1|h=1) = 0.6923}$

(ii) $P(c=1) = P(c=1|v=1) \times P(v=1) + P(c=1|v=0) \times P(v=0)$

$= 0.480$

$P(c=0) = P(c=0|v=1) \times P(v=1) + P(c=0|v=0) \times P(v=0)$

$= 0.520$

$P(t=1) = P(t=1|c=1) \times P(c=1) + P(t=1|c=0) \times P(c=0)$

$= 0.482$

$P(t=1|v=1) = P(t=1|c=1) \times P(c=1|v=1) + P(t=1|c=0) \times P(c=0|v=1)$

$= 0.95 \times 0.939 + 0.05 \times 0.060$

$= 0.895$

$$P(v=1|t=1) = \frac{P(t=1|v=1) \times P(v=1)}{P(t=1)} = \boxed{0.371}$$

(iii)

$$P(v=1|t=1 \cap h=1) = \frac{P(t=1 \cap h=1| v=1) \times P(v=1)}{P(t=1 \cap h=1)}$$

$$= \frac{P(t=1 \cap h=1|v=1) \times P(v=1)}{P(t=1 \cap h=1|v=1) \times P(v=1) + P(t=1 \cap h=1|v=0) \times P(v=0)}$$

We Know,

$$P(t=1 \cap h=1|v=1) = P(t=1|v=1) \times P(h=1|v=1)$$

$$P(t=1 \cap h=1|v=0) = P(t=1|v=0) \times P(h=1|v=0)$$

Also,

$P(v=1) = 0.2$, $P(h=1|v=1) = 0.9$, $P(h=1|v=0) = 0.1$

$P(t=1|v=1) = 0.895$

$$P(t=1|v=0) = P(t=1|c=1) \times P(c=1|v=0) + P(t=1|c=0) \times P(c=0|v=0)$$

$$= 0.95 \times 0.366 + 0.05 \times 0.634$$

$$= 0.379$$

$$P(v=1|t=1 \cap h=1) = \frac{0.895 \times 0.9 \times 0.2}{0.895 \times 0.9 \times 0.2 + 0.379 \times 0.1 \times 0.8}$$

$$= \boxed{0.842}$$

3. Let $x_i \sim \text{Poss}(\lambda)$, $1 = 1, \ldots, N$. Find λ using MLE.

(3) Let $x_i \sim \text{Poss}(\lambda)$, $i=1 \ldots N$

Poisson function :

$$f(x) = \frac{e^{-\lambda}\lambda^{x}}{x!} \qquad x = 0, 1, 2, \ldots$$

log likelihood function

$$l(\lambda) = \ln \prod_{i=1}^{n} f(x_i, \lambda)$$

$$= \sum_{i=1}^{n} \ln \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}$$

$$= \sum_{i=1}^{n} \ln e^{-\lambda} + \sum_{i=1}^{n} x_i \ln \lambda$$

$$= -\sum_{i=1}^{n} \ln x_i!$$

To find the value of λ that maximizes log likelihood function by derivating with respect to λ

$$\frac{d}{d\lambda} l(\lambda) = 0$$

$$\frac{d}{d\lambda}\left\{\sum_{i=1}^{n} \ln e^{-\lambda} + \sum_{i=1}^{n} x_i \ln \lambda - \sum_{i=1}^{n} \ln x_i!\right\} = 0$$

$$\frac{d}{d\lambda}\left\{\sum_{i=1}^{n} (-\lambda) + \sum_{i=1}^{n} x_i \ln(\lambda) - \sum_{i=1}^{n} \ln x_i!\right\} = 0$$

$$-n + \frac{1}{\lambda}\sum_{i=1}^{n} x_i = 0$$

$$\boxed{\lambda = \frac{1}{n}\sum_{i=1}^{n} x_i}$$

∴ The maximum likelihood estimate for the parameter λ of a poission distribution is as shown above.

4. Let $x_i \sim \text{Poss}(\lambda)$, $1 = 1, \dots, N$. Find $p(\lambda|x_1, \dots x_N)$, assuming the Gamma prior $\lambda \sim \Gamma(\lambda|a, b)$.

(4)

We know,

likelihood: $f(x|\lambda) = \dfrac{e^{-N\lambda}(N\lambda)^{\alpha}}{\alpha!}$

Prior: $f(\lambda) = \dfrac{b^a}{P(a)} \lambda^{a-1} e^{-b\lambda}$

Suppose,

$\alpha_1, \alpha_2, \dots \alpha_n \sim \text{poss}(N\lambda)$

then

$f(\alpha_1, \dots \alpha_m|\lambda) = f(\alpha_1|\lambda) \dots \dots f(\alpha_m|\lambda)$

we can see that the observations are independent.

So,

$$f(\lambda|\alpha_1 \dots \alpha_m) = \frac{f(\alpha_1|\lambda) \dots \dots f(\alpha_m|\lambda) \cdot f(\lambda)}{f(\alpha_1, \dots \alpha_m)}$$

$$= \frac{\dfrac{e^{-N\lambda}(N\lambda)^{\alpha_1}}{\alpha_1!} \dots \dots \dfrac{e^{-N\lambda}(N\lambda)^{\alpha m}}{\alpha m!} \cdot \dfrac{b^a}{P(a)} \lambda^{a-1} e^{-b\lambda}}{f(\alpha_1, \alpha_2, \dots \alpha m)}$$

Taking derivatives,

$$= e^{-mN\lambda} \lambda^{\alpha_1 + \dots + \alpha m} \lambda^{a-1} e^{-b\lambda}$$

$$= e^{-(mN+b)\lambda} \lambda^{(\alpha_1 + \dots + \alpha m + a) - 1}$$

$\therefore \lambda|\alpha_1, \dots \alpha m \sim \Gamma(\alpha_1 + \dots + \alpha m + a, mN + b)$

5. Let $x_1, x_2, \ldots, x_N$ be an independent sample from the exponential distribution with density $p(x|\lambda) = \lambda \exp(-\lambda x)$, $x \geq 0$, $\lambda > 0$. Find the maximum likelihood estimate λ_{ML}. Choose the conjugate prior distribution $p(\lambda)$, and find the posterior distribution $p(\lambda|x_1, \ldots, x_N)$ and calculate the Bayesian estimate for λ as the expectation over the posterior.

(5) $p(x|\lambda) = \lambda \exp(-\lambda x)$ $\quad x \geq 0,\ \lambda > 0$

The log likelihood is

$$l(\lambda) = \sum_i \log \lambda - \lambda x_i = n \log \lambda - \lambda \sum_i x_i$$

Set the derivative to 0:

$$n/\lambda - \sum_i x_i = 0 \implies \lambda = \frac{1}{\bar{x}}$$

Posterior distribution = likelihood × prior

$$P(\lambda|x_i) = \frac{P(\lambda) . P(x_i|\lambda)}{\int_0^{\infty} P(x_1|\lambda) . P(\lambda)}$$

$\lambda \sim \text{Gamma}(\alpha, \beta)$

$$P(\lambda|x_i) = \frac{P(x_i|\lambda) . P(\lambda)}{\int_0^{\infty} P(x_i|\lambda) . P(\lambda)}$$

$$= \frac{\left(\frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda}\right)\left(\lambda^{\sum_{i=1}^n \delta_i} e^{-\lambda\left(\sum_{i=1}^n x_i\right)}\right)}{\int_0^{\infty} \left(\frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda}\right)\left(\lambda^{\sum_{i=1}^n \delta_i} e^{-\lambda\left(\sum_{i=1}^n x_i\right)}\right) d\lambda}$$

$$= \frac{\frac{b^a}{\Gamma(a)} \lambda^{\sum_{i=1}^n \delta_i + a - 1} e^{-\lambda\left(\sum_{i=1}^n x_i + b\right)}}{\frac{b^a}{\Gamma(a)} \Gamma\left(\sum_{i=1}^n \delta_i + \alpha\right)\left(\frac{1}{\sum_{i=1}^n x_i + b}\right)^{\sum_{i=1}^n \delta_i + a}}$$

$$= \frac{\lambda^{\sum_{i=1}^n \delta_i + a - 1} e^{-\lambda\left(\sum_{i=1}^n x_i + b\right)}}{\frac{b^a}{\Gamma(a)} \Gamma\left(\sum_{i=1}^n \delta_i + a\right)\left(\frac{1}{\sum_{i=1}^n x_i + b}\right)^{\sum_{i=1}^n \delta_i + a}}$$

$$= \frac{\left(\sum_{i=1}^{n} x_i + b\right)^{\sum_{i=1}^{n} \delta_i + a}}{\Gamma\left(\sum_{i=1}^{n} \delta_i + a\right.} \lambda^{\sum_{i=1}^{n} \delta_i + a - 1} e^{-\lambda\left(\sum_{i=1}^{n} x_i + b\right)}$$

$$= \text{Gamma}\left(\sum_{i=1}^{n} \delta_i + a, \sum_{i=1}^{n} x_i + b\right)$$

Bayesian estimation of λ as expectation over posterior

$$E\left[e^{-c\lambda}\right] = \int_0^{\infty} e^{-c\lambda} p(\lambda | x_i)\, d\lambda$$

$$= \int_0^{\infty} e^{-c\lambda} \frac{\left[\sum_{i=1}^{n} x_i + b\right]^{\sum_{i=1}^{n} \delta_i + a} \times \lambda^{\sum_{i=1}^{n} \delta_i + a - 1} \times e^{-\lambda\left(\sum_{i=1}^{n} x_i + b\right)}}{\Gamma\left(\sum_{i=1}^{n} \delta_i + a\right.} d\lambda$$

$$= \frac{\left(\sum_{i=1}^{n} x_i + b\right)^{\sum_{i=1}^{n} \delta_i + a}}{\Gamma\left(\sum_{i=1}^{n} \delta_i + a\right.} \int_0^{\infty} \lambda^{\sum_{i=1}^{n} \delta_i + a - 1} e^{-\lambda\left(\sum_{i=1}^{n} x_i + b + c\right)} d\lambda$$

$$= \frac{\left(\sum_{i=1}^{n} x_i + b\right)^{\sum_{i=1}^{n} \delta_i + a} \Gamma\left(\sum_{i=1}^{n} \delta_i + a\right)}{\left(\sum_{i=1}^{n} x_i + b + c\right)^{\sum_{i=1}^{n} \delta_i + a} \Gamma\left(\sum_{i=1}^{n} \delta_i + a\right)}$$

$$= \left(\frac{\sum_{i=1}^{n} x_i + b}{\sum_{i=1}^{n} x_i + b + c}\right)^{\sum_{i=1}^{n} \delta_i + a}$$

$$\bar{\lambda} = -\frac{1}{c} \ln\left[E\left(e^{-c\lambda}\right)\right]$$

$$\hat{\lambda}_{BC} = -\frac{1}{c} \ln\left[\left(\frac{\sum_{i=1}^{n} x_i + b}{\sum_{i=1}^{n} x_i + b + c}\right)^{\sum_{i=1}^{n} \delta_i + a}\right]$$

6. SGD for Ridge Regression Solve the ℓ_2 regularized logistic regression. The objective function is

$$f(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right) + \frac{\lambda}{2m}||\theta||_2^2$$

where

$$h_\theta(x) = g(\theta^T x), \qquad g(z) = \frac{1}{1+e^{-z}}$$

There are three parameters in the model $\theta = (\theta_1, \theta_2, \theta_3)$, where θ_1 corresponds to intersect, so you need to add column of ones to the data. Fit the logistic regression to the `reg-lr-data` dataset. Write down the derivative for the objective function.

(a) Solve using gradient descent method

(b) Solve using stochastic gradient descent.

Run the algorithms with different step sizes and different values of λ. Plot data and the regression line as well as convergence plot (iteration vs sopping criteria) for several runs.

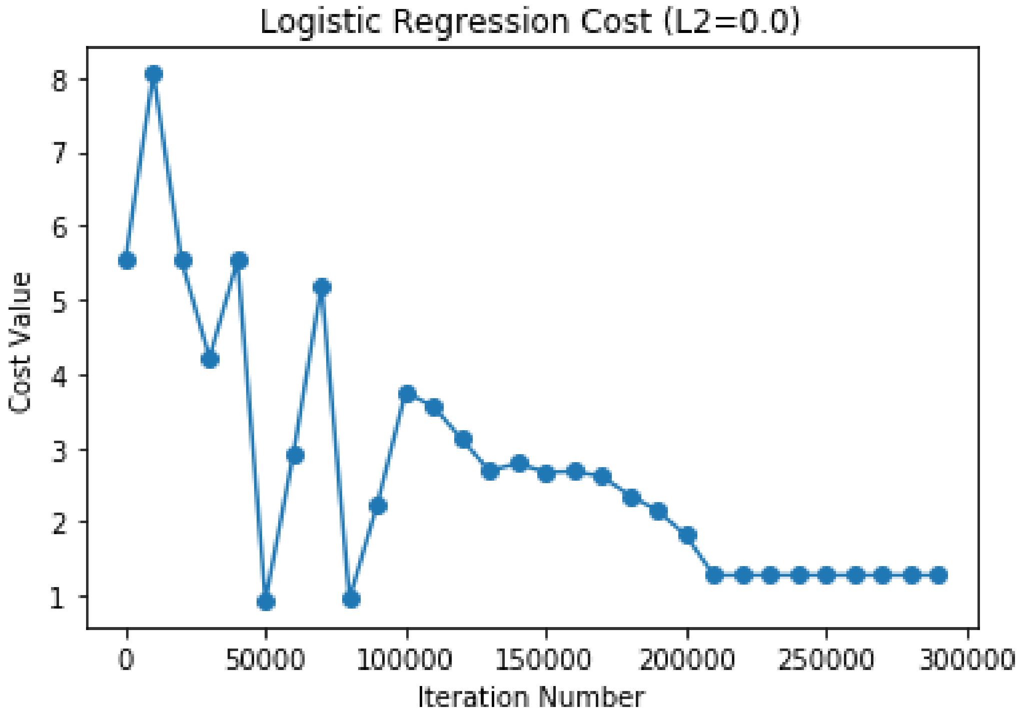
Q6) Code with Outputs as seen in Spyder (Anaconda Python Console):

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 15 20:37:31 2019

@author: Abhishek Shambhu
"""
# Building a Logistic Regression model and fitting it on the reg-lr-data dataset
# Importing packages and reading the data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from textwrap import wrap
data = pd.read_csv(r'D:\sem3\or610\reg-lr-data.csv')

# splitting the data into the x values and y values as numpy arrays
x = data.iloc[:,:3].values
y = data['y'].values

# Logistic Regression Model using Gradient Descent Method
class Log_Reg_L2:
    """ Defining Logistic Regression with L2 regularization
    Parameters are:
        l2: lambda value for l2 regularization
        n: number of iterations over the dataset
        l_rate: learning rate/step value
    """

    def __init__(self, l2=0.0, n=1000, l_rate=0.05):
        self.l2 = l2
        self.n = n
        self.l_rate = l_rate

    def sigmoid(self, z):
        # This is the sigmoid function of z
```

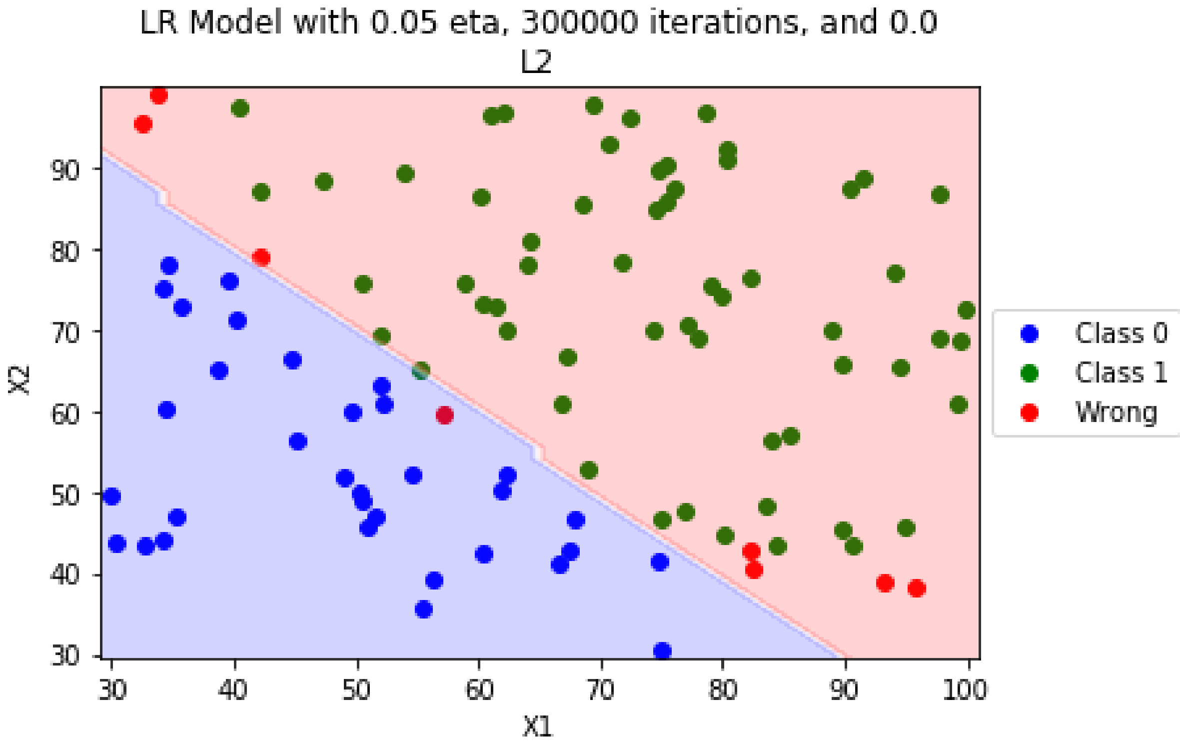
```
        return 1/(1+ np.exp(-z))

    def fit(self, x, y):
        # fit the training data

        y = y.reshape(-1,1)
        # initialize the values of the weights to zero
        self.theta = np.zeros((x.shape[1],1))
        m = y.shape[0]
        # adding in padding value so that we never take the log of 0
        pad = 1e-6
        self.cost_values = []
        for i in range(self.n):
            z = self.sigmoid(np.dot(x, self.theta))
            # calculating the gradient with the derived formula
            gradient = x.T.dot(z-y)/m + (self.l2/m*self.theta)
            self.theta -= self.l_rate * gradient
            # implementing the cost (objective) function given
            cost = np.average(-y*np.log(z+pad) - ((1-y)*np.log(1-z+pad)))
            l2_cost = cost + (self.l2/(2*m) * np.linalg.norm(self.theta[1:])**2)  # we don't regularize the
intersect
            self.cost_values.append(l2_cost)
        return self

    def predict(self, x, threshold=0.5):
        # return the predicted values in (0,1) format
        return np.where(self.sigmoid(x.dot(self.theta)) >= threshold,1,0)

    def predict_prob(self, x):
        # return the predicted values in percentage format
        return self.sigmoid(x.dot(self.theta))

# Logistic Regression using Stochastic Gradient Descent
class Log_Reg_L2_SGD:
    """ Logistic Regression with L2 regularization and Stochastic Gradient Descent
    The parameters are:
        l2: lambda value for l2 regularization
        n: number of iterations over the dataset
        l_rate: learning rate
        batch_size: size of each batch (SGD=1 and full batch = len(x))
    """

    def __init__(self, l2=0.0, n=1000, l_rate=0.05, batch_size=1):
        self.l2 = l2
        self.n = n
        self.l_rate = l_rate
        self.batch_size = batch_size

    def sigmoid(self, z):
        # This is the sigmoid function of z
        return 1/(1+ np.exp(-z))

    def fit(self, x, y):
        # fit the training data
```

```
        y = y.reshape(-1,1)
        # initialize the values of the weights to zero
        self.theta = np.zeros((x.shape[1],1))
        m = y.shape[0]
        pad = 1e-6
        self.cost_values = []
        for i in range(self.n):
            # shuffling each iteration as to prevent overfitting
            shuffled_values = np.random.permutation(m)
            X_shuffled = x[shuffled_values]
            y_shuffled = y[shuffled_values]
            # iterating over each batch
            for batch in range(0, m, self.batch_size):
                x_batch = X_shuffled[batch:batch+self.batch_size]
                y_batch = y_shuffled[batch:batch+self.batch_size]
                z = self.sigmoid(np.dot(x_batch, self.theta))
                # calculating the gradient with the derived formula
                gradient = x_batch.T.dot(z-y_batch)/m + (self.l2/m*self.theta)
                self.theta -= self.l_rate * gradient
                # implementing the cost (objective) function given
                cost = np.average(-y_batch*np.log(z+pad) - ((1-y_batch)*np.log(1-z+pad)))
                l2_cost = cost + (self.l2/(2*m) * np.linalg.norm(self.theta[1:])**2)  # we don't regularize the
intersect
                self.cost_values.append(l2_cost)
        return self

    def predict(self, x, threshold=0.5):
        # return the predicted values in (0,1) format
        return np.where(self.sigmoid(x.dot(self.theta)) >= threshold,1,0)

    def predict_prob(self, x):
        # return the predicted values in percentage format
        return self.sigmoid(x.dot(self.theta))

# Function to Plot the Cost Values
def plot_cost(trained_model, printed_values = 30, is_sgd=False):
    # printed values determines how many values are printed to the chart
    # this prevents the chart from becoming too cluttered
    if is_sgd:
        # averaging the values over each iteration
        batch_avg = [np.mean(trained_model.cost_values[i:i+4]) for i in range(1,
len(trained_model.cost_values), int(x.shape[0]/trained_model.batch_size))]
        model_plot = [batch_avg[i] for i in range(1, len(batch_avg), int(trained_model.n/printed_values))]
        plt.plot(range(1, len(batch_avg),int(trained_model.n/printed_values)), model_plot, marker='o')
        plt.xlabel('Iteration Number')
        plt.ylabel('Cost Value')
        plt.title('Logistic Regression Cost (L2={})'.format(trained_model.l2))
    else:
        model_plot = [trained_model.cost_values[i] for i in range(1, len(trained_model.cost_values),
int(trained_model.n/printed_values))]
        plt.plot(range(1, len(trained_model.cost_values)+1,int(trained_model.n/printed_values)), model_plot,
marker='o')
        plt.xlabel('Iteration Number')
        plt.ylabel('Cost Value')
        plt.title('Logistic Regression Cost (L2={})'.format(trained_model.l2))
```

```
# Function to Plot the Decision Boundary
def plot_decision_boundary(trained_model, x, y, is_sgd=False):
    fig, ax = plt.subplots()
    predictions = model.predict(x)
    #plotting class = 0 correct
    ax.scatter(x[(predictions.flatten() == y) & (y==0)][:,1], x[(predictions.flatten() == y) & (y==0)][:,2],
color='b', label="Class 0")
    # plotting class = 1 correct
    ax.scatter(x[(predictions.flatten() == y) & (y==1)][:,1], x[(predictions.flatten() == y) & (y==1)][:,2],
color='g', label="Class 1")
    # plotting incorrect classifications
    ax.scatter(x[predictions.flatten() !=y][:,1], x[predictions.flatten() != y][:,2], color='r', label="Wrong")
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

    x1_min, x1_max = x[:,1].min()-1, x[:,1].max()+1
    x2_min, x2_max = x[:,2].min()-1, x[:,2].max()+1
    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_max))

    graph_predictions = model.predict(np.array([np.ones((2500,1)).ravel(), xx1.ravel(), xx2.ravel()]).T)
    graph_predictions = graph_predictions.reshape(xx1.shape)
    ax.contourf(xx1, xx2, graph_predictions,alpha=0.2, cmap='bwr')
    ax.set_xlim(xx1.min(), xx1.max())
    ax.set_ylim(xx2.min(), xx2.max())
    if is_sgd:
        ax.set_title('\n'.join(wrap("SGD LR Model with {} batch size, {} eta, {} iterations, and {}
L2".format(trained_model.batch_size,
                    trained_model.l_rate, trained_model.n, trained_model.l2),50)), fontsize=12)
    else:
        ax.set_title('\n'.join(wrap("LR Model with {} eta, {} iterations, and {} L2".format(
                    trained_model.l_rate, trained_model.n, trained_model.l2),50)),fontsize=12)
    plt.show()

#Part 1)
#Part 1A) Logistic Regression Gradient Descent: l_rate=0.05, n=300,000, L2=0.0
model = Log_Reg_L2(l2=0.0, n=300000,l_rate=0.05)
model.fit(x, y)
predictions = model.predict(x)
print("1A) Accuracy: {:.0f}%".format(sum(predictions.flatten() == y)/len(y)*100))
#1A) Accuracy: 92%

plot_cost(model, printed_values=30)
```

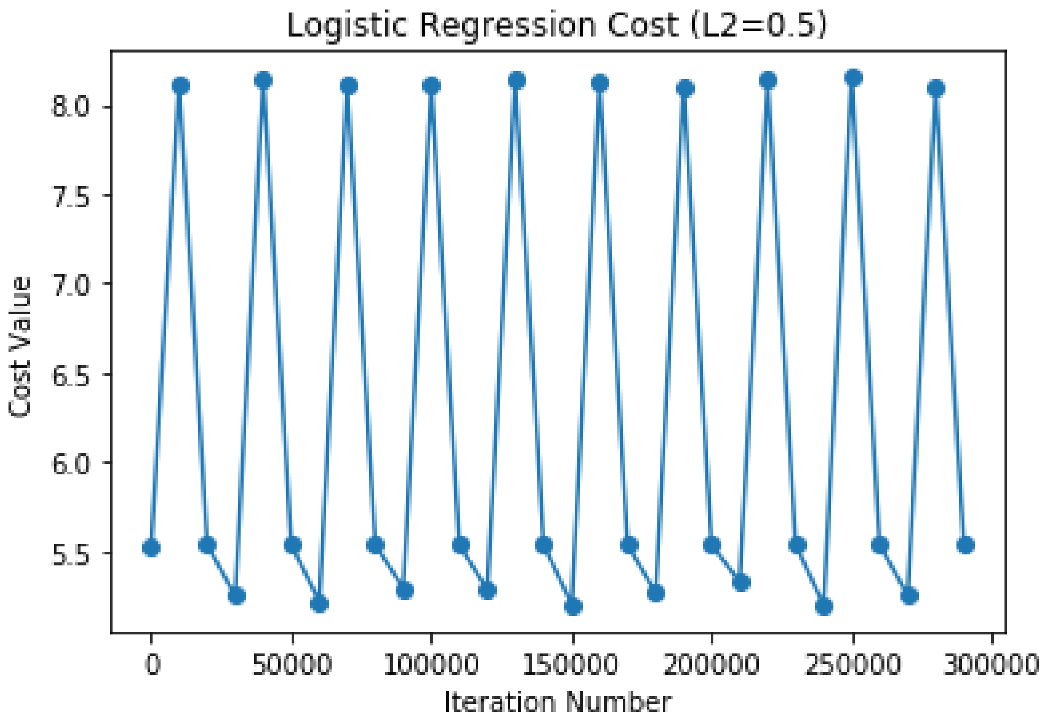


plot_decision_boundary(model, x, y)

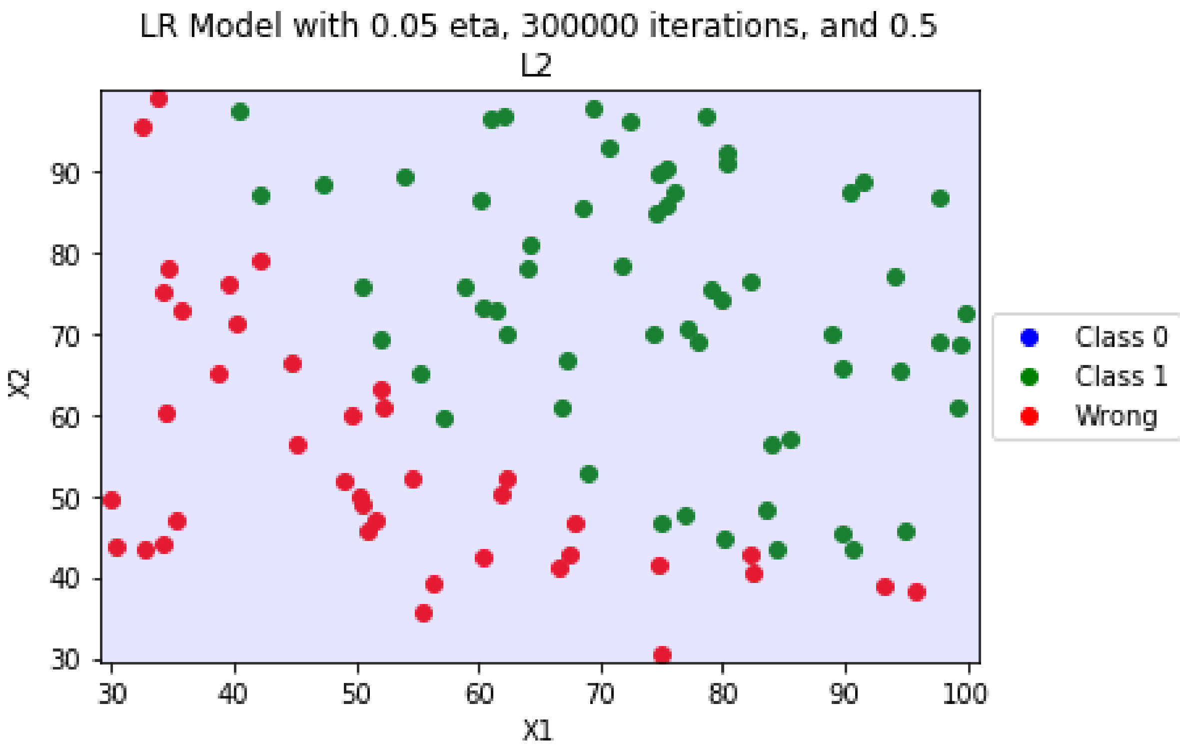


#Part 1B) Logistic Regression Gradient Descent: l_rate=0.05, n=300,000, L2=0.5
model = Log_Reg_L2(l2=0.5, n=300000,l_rate=0.05)
model.fit(x, y)
predictions = model.predict(x)
print("1B) Accuracy: {:.0f}%".format(sum(predictions.flatten() == y)/len(y)*100))
#1B) Accuracy: 60%

plot_cost(model, printed_values=30)

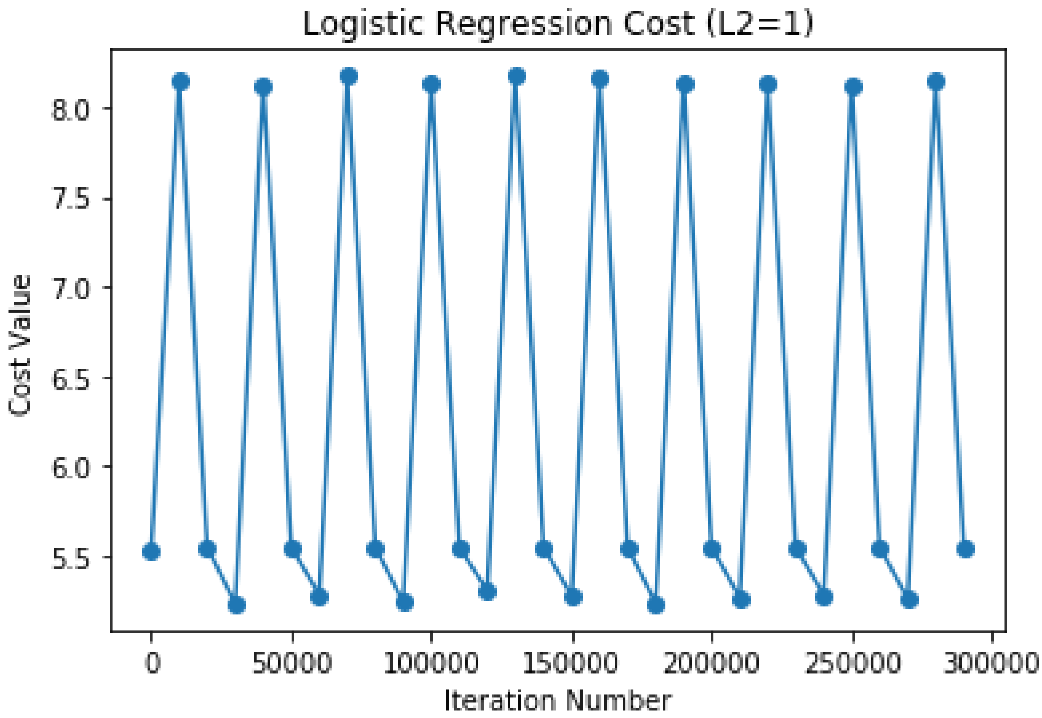


plot_decision_boundary(model, x, y)

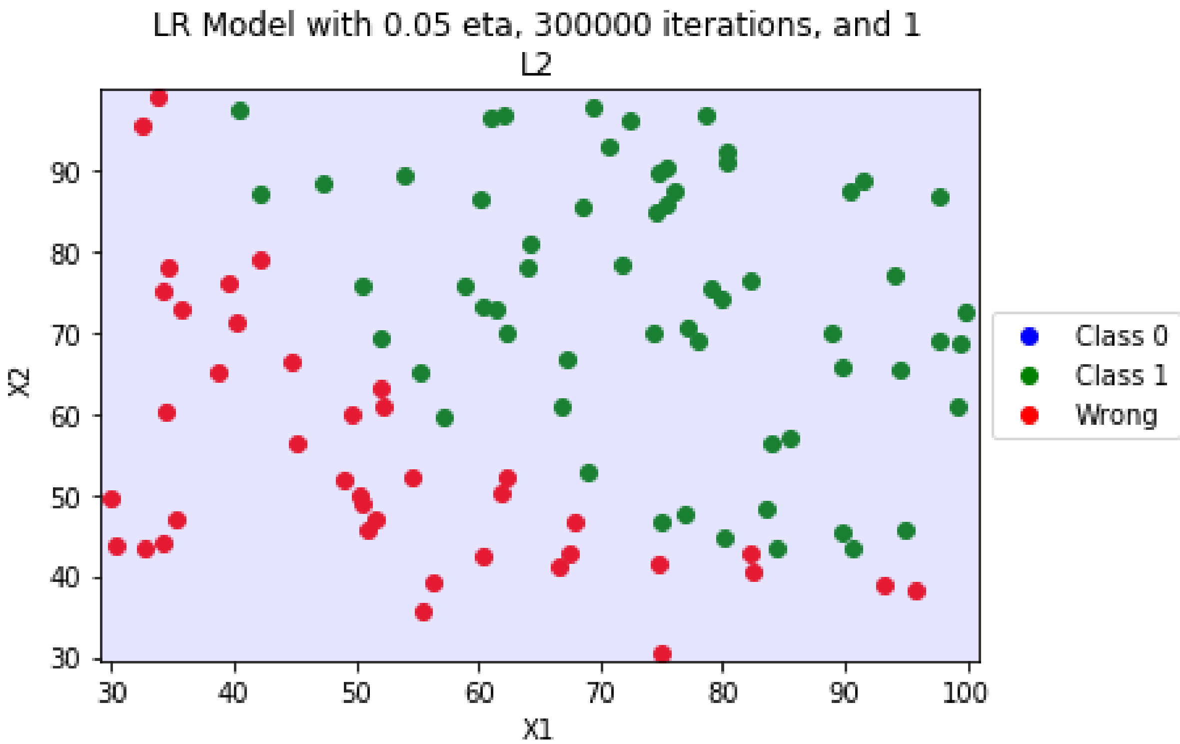


#Part 1C) Logistic Regression Gradient Descent: l_rate=0.05, n=300,000, L2=1
model = Log_Reg_L2(l2=1, n=300000, l_rate=0.05)
model.fit(x, y)

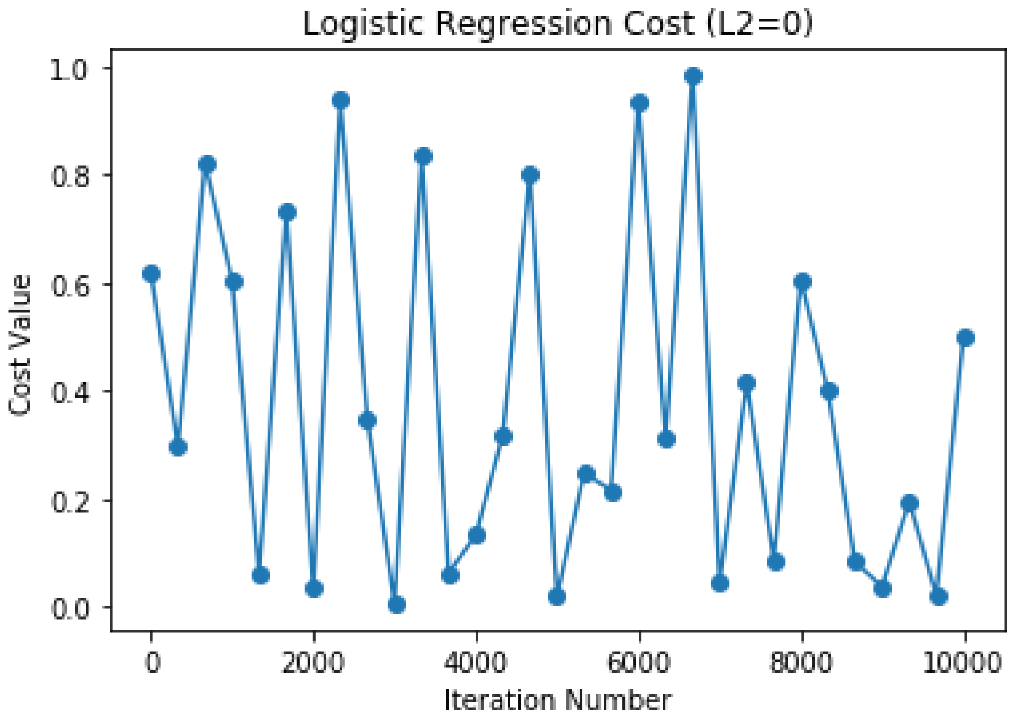
```
predictions = model.predict(x)
print("1C) Accuracy: {:.0f}%".format(sum(predictions.flatten() == y)/len(y)*100))
#1C) Accuracy: 60%

plot_cost(model, printed_values=30)
```



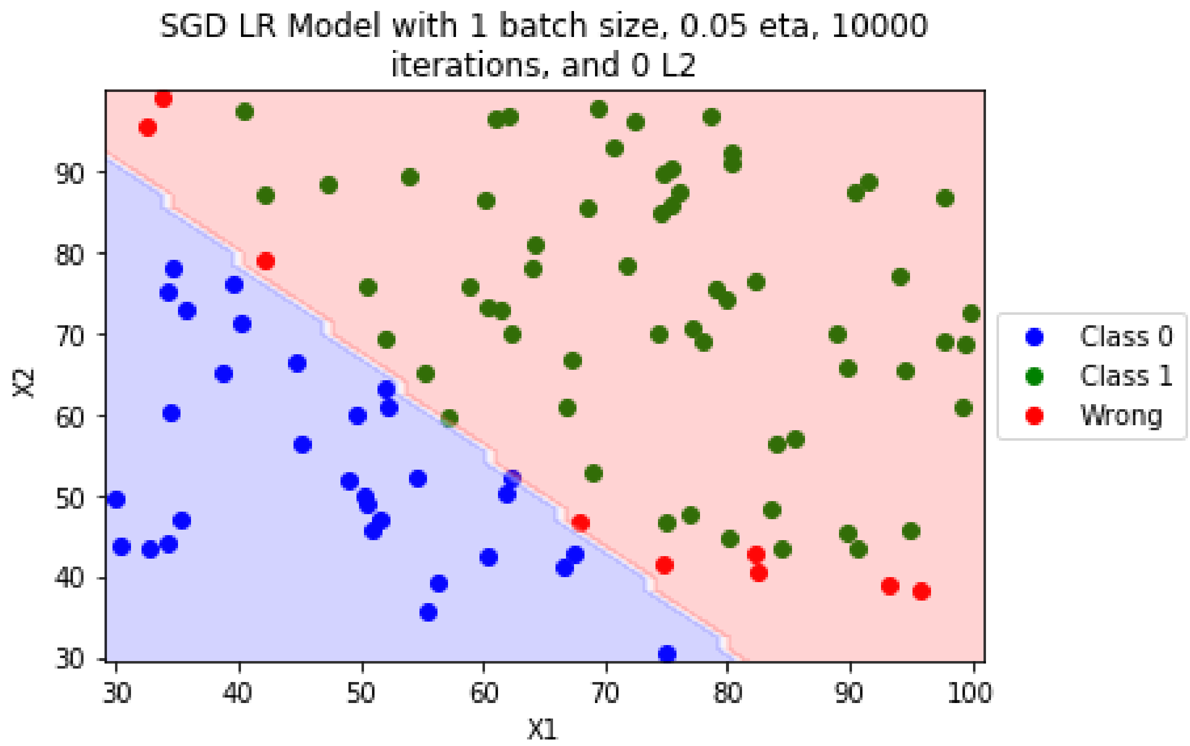

```
plot_decision_boundary(model, x, y)
```



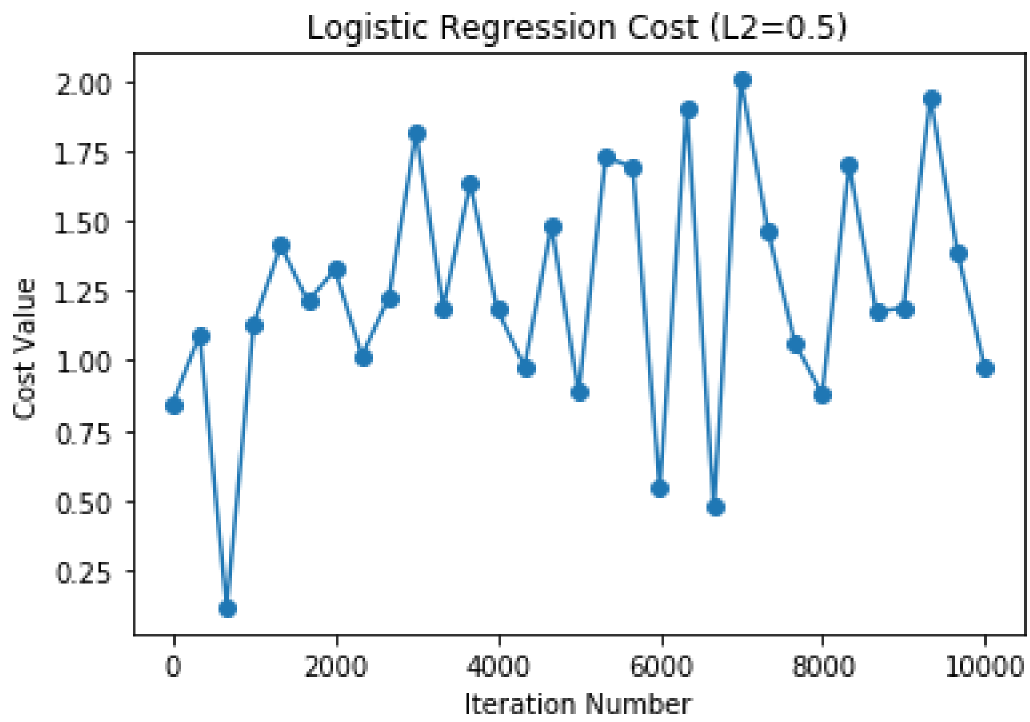
```
#Part 2)
#Part 2A) Logistic Regression Stochastic Gradient Descent: l_rate=0.05, n=10,000, L2=0.0, batch_size=1

model = Log_Reg_L2_SGD(l2=0, n=10000,l_rate=0.05, batch_size=1)
model.fit(x, y)
predictions = model.predict(x)
print("2A) Accuracy: {:.0f}%".format(sum(predictions.flatten() == y)/len(y)*100))
#2A) Accuracy: 91%

plot_cost(model, printed_values=30, is_sgd=True)
```



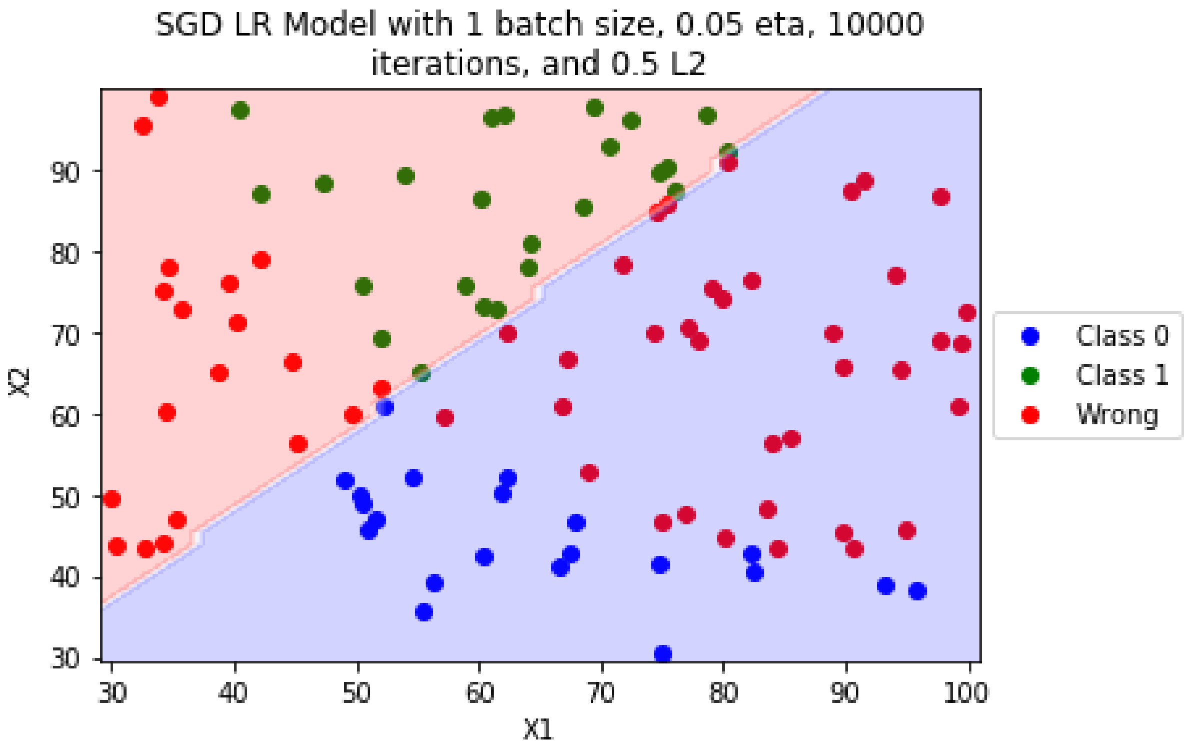

```
plot_decision_boundary(model, x, y, is_sgd=True)
```



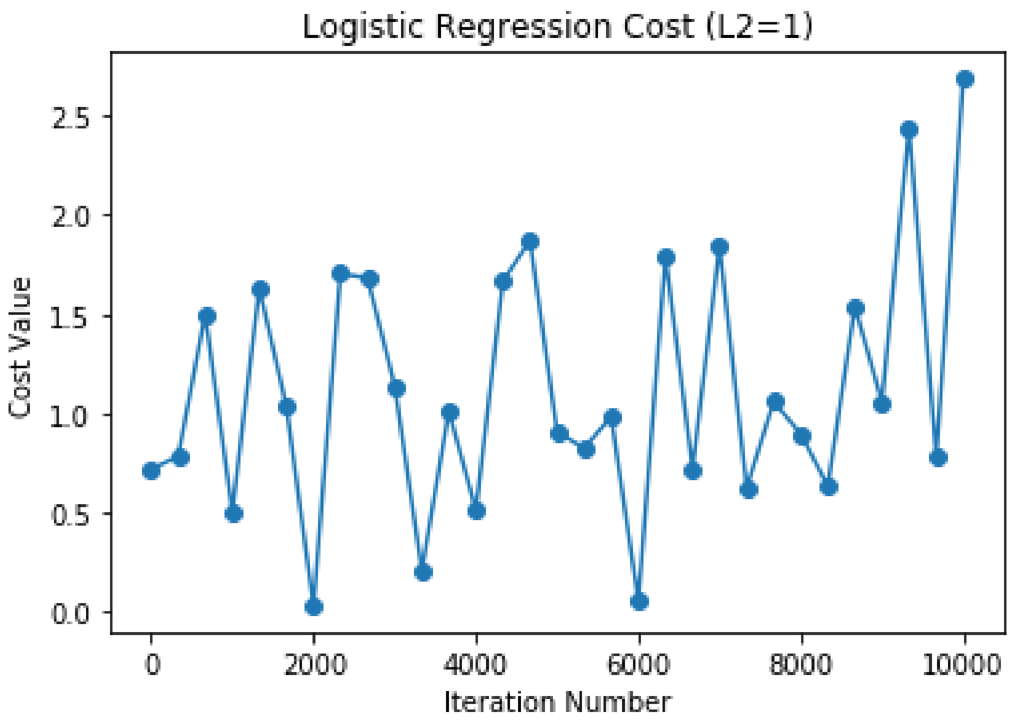

```
#Part 2B) Logistic Regression Stochastic Gradient Descent: l_rate=0.05, n=10,000, L2=0.5, batch_size=1
model = Log_Reg_L2_SGD(l2=0.5, n=10000,l_rate=0.05, batch_size=1)
model.fit(x, y)
predictions = model.predict(x)
print("2B) Accuracy: {:.0f}%".format(sum(predictions.flatten() == y)/len(y)*100))
#2B) Accuracy: 45%

plot_cost(model, printed_values=30, is_sgd=True)
```



plot_decision_boundary(model, x, y, is_sgd=True)




```
#Part 2C) Logistic Regression Stochastic Gradient Descent: l_rate=0.05, n=10,000, L2=1, batch_size=1
model = Log_Reg_L2_SGD(l2=1, n=10000,l_rate=0.05, batch_size=1)
model.fit(x, y)
predictions = model.predict(x)
print("2C) Accuracy: {:.0f}%".format(sum(predictions.flatten() == y)/len(y)*100))
#2C) Accuracy: 60%

plot_cost(model, printed_values=30, is_sgd=True)
```




```
plot_decision_boundary(model, x, y, is_sgd=True)
```