--------------------------------------- start from here ---

Background (问题背景):

Let's consider a binary classification. For example, students are going to apply for student loan from a financial institution. The instution decides whether it will approve the student application based on two factors: the student's credit score and his/her monthly income. Now we can use $X1$ to represent the student's monthly income and $X2$ to represent the student's credit score. Then we can use $Y \in \{1, 0\}$ to denote the application results. $Y_i = 1$ means that the institution approves the student's application and 0 for the reject of application.

Imagine that we have 100 students who will apply for the student loan. Obviously, some of them can get the load whereas others get rejected.

To model this problem, we can use logistic regression to classify the students who can and who cannot get the loan. $X1$ and $X2$ are two column vectors with the length of 100. Y is a column vector whose length is also 100. Each element in Y can either be 1 or 0.

我们现在来考虑一个二分类问题。例如，每年都会有学生向银行申请学生贷款。一般而言，银行会根据两个因素来判定是否通过该学生贷款的申请：信用记录分数以及该学生每个月的收入。这样的话，我们可以用$X1$和$X2$分别表达学生的月收入和信用分数。然后我们可以用$Y \in \{1, 0\}$来表达该学生是否可以成功从银行取得贷款。Y=1表示该学生成功获得贷款，而Y=0表示该生贷款申请遭到拒绝。

假设我们有200个学生向银行申请贷款。显然，这200名学生中有人将取得贷款，而另一部分学生可能因为较差的信用分数或者较低的月收入而申请被拒。

我们可以利用逻辑回归对该问题建模。这里，$X1$和$X2$分别是一个长度为100的列向量。Y也是一个长度为200的列向量。Y中的每一个元素只可能是1或者0.

First of all, the data of X1, X2, and Y are generated using some random function.

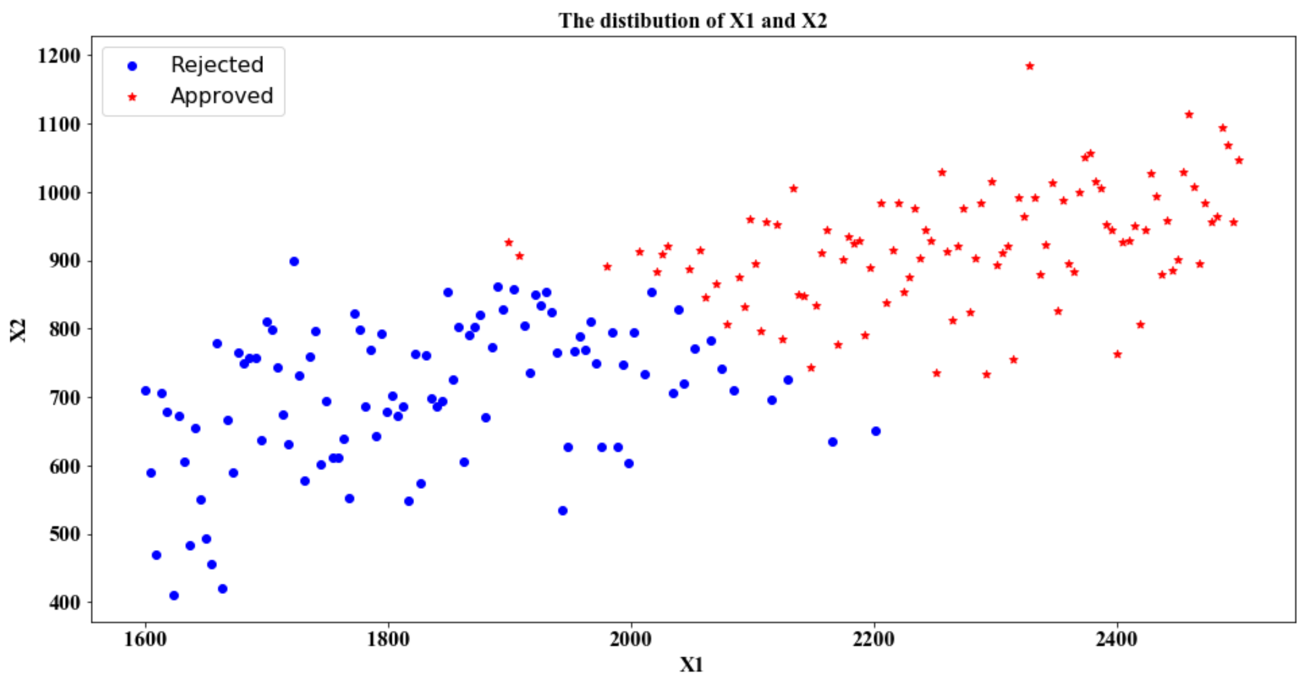
首先我们来生成X1, X2, 和Y的数据。

(It should be noted that the generation for X1, X2, and Y are kind of arbitrary as we don't have actual data from financial institution)

这里需要声明的是，X1, X2以及Y的是随意生成的，因为我们没有实际的数据。

```
# Add this command because of plot
# Otherwise, the plot may not show up in Anaconda
% matplotlib inline

# Import necessary packages
import matplotlib.pyplot as plt  # Package used for plotting the figure
import numpy as np  # Package relating to math

# Define length of X1 vector, which is dentical to the length of X2 and Y
# In total, 200 students are applying for the student load
number_observation = 200

# Generate X1 and X2
X1 = np.linspace(1600, 2500, number_observation)  # Assume the students' monthly income ranges
from 1600 to 2500
X2 = X1/2.5 + np.random.normal(0, 100, number_observation)  # Assume the credit scrore is
following this function

# Generate Y vector
```

```
Y = np.linspace(0, 0, number_observation)
# indicator represents whther the financial institution approves the student's application or
not
# indicator < 0 means the application is rejected.
# indicator >= 0 means the applcation is approved.
# A noise term is added because I don't want the data to be linearly saparable
indicator = (X1 - np.mean(X1))/np.std(X1) + (X2 - np.mean(X2))/np.std(X2) + np.random.normal(0,
0.25, number_observation)
Y[indicator < 0] = 0
Y[indicator >= 0] = 1

# Visualize the data points
plt.figure(figsize=(16,8))
plt.scatter(X1[indicator < 0], X2[indicator < 0], color='b', marker='o', label='Rejected')
plt.scatter(X1[indicator >= 0], X2[indicator >= 0], color='r', marker='*' , label='Approved')
font = {'family':'Times New Roman','weight' : 'normal','size': 16,}
plt.xlabel('X1', font)
plt.ylabel('X2', font)
plt.xticks(fontsize=16, fontname='Times New Roman')
plt.yticks(fontsize=16, fontname='Times New Roman')
plt.title('The distibution of X1 and X2', font)
plt.legend(fontsize=16)
plt.show()
```

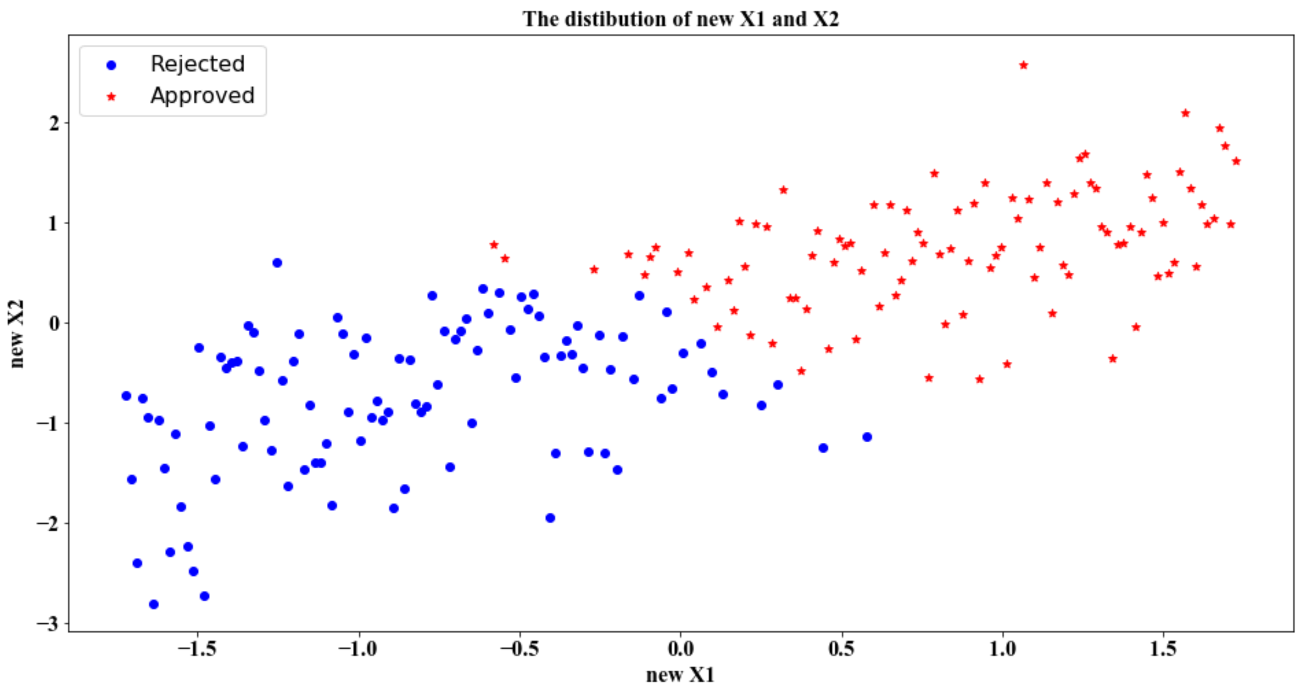


Later we will use sigmoid function to deal with these data points. As we know, sigmoid function invovles the exponential algorithm. Since X1 is large (the maximum is about 2400), we may encounter overflow problem in coding. To avoid the overflow problem, we have to normailize X1 and X2.

在后面我们将会用sigmoid函数。这个函数包含了以自然指数e为底的运算。因为X1的数值很大，因此我们很有可能会碰到数值溢出的问题。为了避免该类问题，我们需要将X1和X2正则化。

```
# Normalize X1 and X2
new_X1 = (X1 - np.mean(X1))/np.std(X1)
new_X2 = (X2 - np.mean(X2))/np.std(X2)

# Visualize the data points
plt.figure(figsize=(16,8))
plt.scatter(new_X1[indicator < 0], new_X2[indicator < 0], color='b', marker='o',
label='Rejected')
```

```
plt.scatter(new_X1[indicator >= 0], new_X2[indicator >= 0], color='r', marker='*' ,
label='Approved')
font = {'family':'Times New Roman','weight' : 'normal','size': 16,}
plt.xlabel('new X1', font)
plt.ylabel('new X2', font)
plt.xticks(fontsize=16, fontname='Times New Roman')
plt.yticks(fontsize=16, fontname='Times New Roman')
plt.title('The distibution of new X1 and X2', font)
plt.legend(fontsize=16)
plt.show()
```



As we can see from the figure, the process of normalization only affects the values of X1 and X2. It doesn't affect the distribution of X1 and X2 (the layout of X1 and X2 are still the same).

从上图可以看出，对X1和X2正则化只是影响了横纵坐标的数值大小，并不会影响数据的分布趋势。

In the figure, red stars represent the case that a student's loan application is approved, whereas the blue dots represent the case that a student's loan application is declined.

图中，红色五角星表示学生的贷款申请被批准；蓝色圆心表示学生的贷款申请被拒。

Although the data points are generated in an arbitrary method, these data points are consistent with our common sense. A student with higher monthly income and higher credit score tends to get approved for his appliation. In contrast, a student with low income and low credit score tends to be rejected for applying the loan.

尽管这些数据点是随意生成的，但是其符合我们的正常认知。当一个学生有较高的月收入和较高的信用分数，他将会大概率获得贷款。而当一个学生的月收入较低，信用分数也很低时，他的贷款申请将会被拒。

Now, we would like to use the sigmoid function to quantify the possibility (or probability) for a single student to get the loan.

现在我们需要用下面的这个叫做sigmoid的函数来表征单个学生能够拿到贷款的概率:

$$h^i(X_1, X_2) = \frac{e^{\beta_1 X_1^i + \beta_2 X_2^i}}{1+e^{\beta_1 X_1^i + \beta_2 X_2^i}}$$

where the superscript i represents the i^{th} student.

式中上标i表示第i个学生。

The question is then how to determine the two parameters: β_1 and β_2

现在的问题便是如何确定$\beta_1$和$\beta_2$的数值。

As what we did in week 1 Linear regression, we need to introduce an objective function (loss function). Then we use gradient descent to find the global minimum of the objective function.

正如我们在第一周线性回归中讲述的一样，我们需要引入一个目标函数（损失函数）。然后我们利用梯度下降方法来寻找目标函数的最小值。

The objective function is listed below and it is revised from the maximum likelihood function of logistic regression:

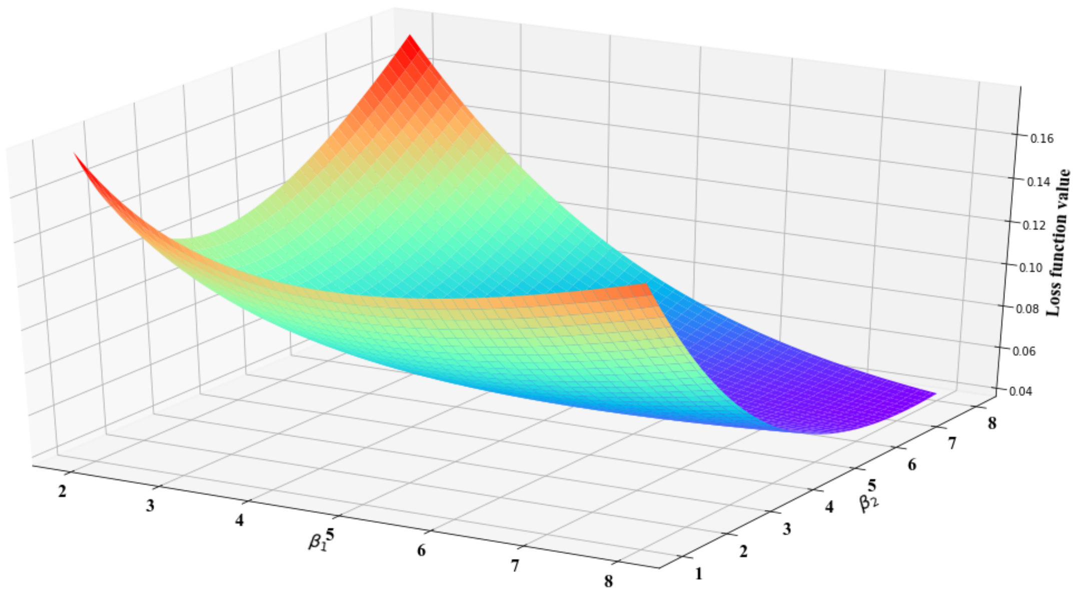
目标函数定义如下，它是基于逻辑回归的最大似然函数修改得到的：

$L(\beta_1, \beta_2) = -\frac{1}{N}\sum_{i=0}^{N}\{y^i ln[h^i(X_1^i, X_2^i)] + (1-y^i)ln[1-h(X_1^i, X_2^i)]\}$

The plot for the objective function is shown below:

该目标函数图象如下所示：

```
# Import the package for three-dimensional plot
from mpl_toolkits.mplot3d import Axes3D

# Generate values for beta_1 and beta_2
number_beta = 100
beta_1 = np.linspace(2, 8, number_beta)
beta_2 = np.linspace(1, 8, number_beta)

# Initialize a matrix to store loss function values
loss = np.zeros([number_beta, number_beta])

# Initialize a matrix to store grid values
grid_beta_1 = np.zeros([number_beta, number_beta])
grid_beta_2 = np.zeros([number_beta, number_beta])

# Calculate the loss function values under different beta_1 and beta_2
for i in range(number_beta):
    for j in range(number_beta):
        grid_beta_1[i,j] = beta_1[i]
        grid_beta_2[i,j] = beta_2[j]
        s = 0
        for indx in range(number_observation):
            h = np.exp(new_X1[indx] * beta_1[i] + new_X2[indx] * beta_2[j]) /
(1+np.exp(new_X1[indx] * beta_1[i] + new_X2[indx] * beta_2[j]))
            temp = Y[indx]*np.log(h) + (1-Y[indx])*np.log(1-h)
            s = s + temp
        loss[i,j] = -1/number_observation * s

# Plot the loss function values over the beta_0 and beta_1 values
fig = plt.figure(figsize=(16,8))
ax = Axes3D(fig)
ax.set_xlabel(r"$\beta_1$", font)
ax.set_ylabel(r"$\beta_2$", font)
ax.set_zlabel("Loss function value", font)
plt.xticks(fontsize=16, fontname='Times New Roman')
plt.yticks(fontsize=16, fontname='Times New Roman')
ax.plot_surface(grid_beta_1, grid_beta_2, loss, cmap='rainbow')
plt.show()
```



From the figure above, the objective function has a global minimum and we are trying to find this global minimum. Again we will use gradient descent to reach our target.

从上图可以看出，目标函数是有全局最小值的。我们会用梯度下降的方法来寻找该最小值。

Let's calculate the derivate for the objective function.

先来计算目标函数关于β_1和β_2的导数。

$\frac{\partial L(\beta_1,\beta_2)}{\partial \beta_1} = -\frac{1}{N}\sum_{i=1}^{N}\{y^i\frac{1}{h^i}(h^i)' + (1 - y^i\frac{1}{1-h^i}(-h^i)'\}$

$= -\frac{1}{N}\sum_{i=1}^{N}\{\frac{y^i}{h^i} - \frac{(1-y^i)}{1-h^i}\}(h^i)'$

$= -\frac{1}{N}\sum_{i=1}^{N}\{\frac{y^i}{h^i} - \frac{(1-y^i)}{1-h^i}\}h^i(1-h^i)x_1^i$

$= -\frac{1}{N}\sum_{i=1}^{N}\{y^i - h^i\}x_1^i$

$\frac{\partial L(\beta_1,\beta_2)}{\partial \beta_2} = -\frac{1}{N}\sum_{i=1}^{N}\{y^i\frac{1}{h^i}(h^i)' + (1 - y^i\frac{1}{1-h^i}(-h^i)'\}$

$= -\frac{1}{N}\sum_{i=1}^{N}\{\frac{y^i}{h^i} - \frac{(1-y^i)}{1-h^i}\}(h^i)'$

$= -\frac{1}{N}\sum_{i=1}^{N}\{\frac{y^i}{h^i} - \frac{(1-y^i)}{1-h^i}\}h^i(1-h^i)x_2^i$

$= -\frac{1}{N}\sum_{i=1}^{N}\{y^i - h^i\}x_2^i$

Where the superscript i means the i^{th} element in X1, X2, or Y. h is just shorthand for $h(X1, X2)$

上式中，上标i表示X1，X2或者Y中的第i个元素。h是$h(X1, X2)$的简写。

Then we can use the negative gradient direction to update the two parameters:

接下来我们可以用负梯度方向来更新两个参数：

$\beta_1^{n+1} = \beta_1^n - s\frac{\partial L(\beta_1^n,\beta_2^n)}{\partial \beta_1^n}$

$\beta_2^{n+1} = \beta_2^n - s\frac{\partial L(\beta_1^n,\beta_2^n)}{\partial \beta_2^n}$

First of all, let's arbitrarily give two initial values to β_1 and β_2 and visualize the classification boundary.

首先我们先给β_1和β_2任意赋两个初始值，然后画出区分边界。

The boundary line is drawn using the following equation:

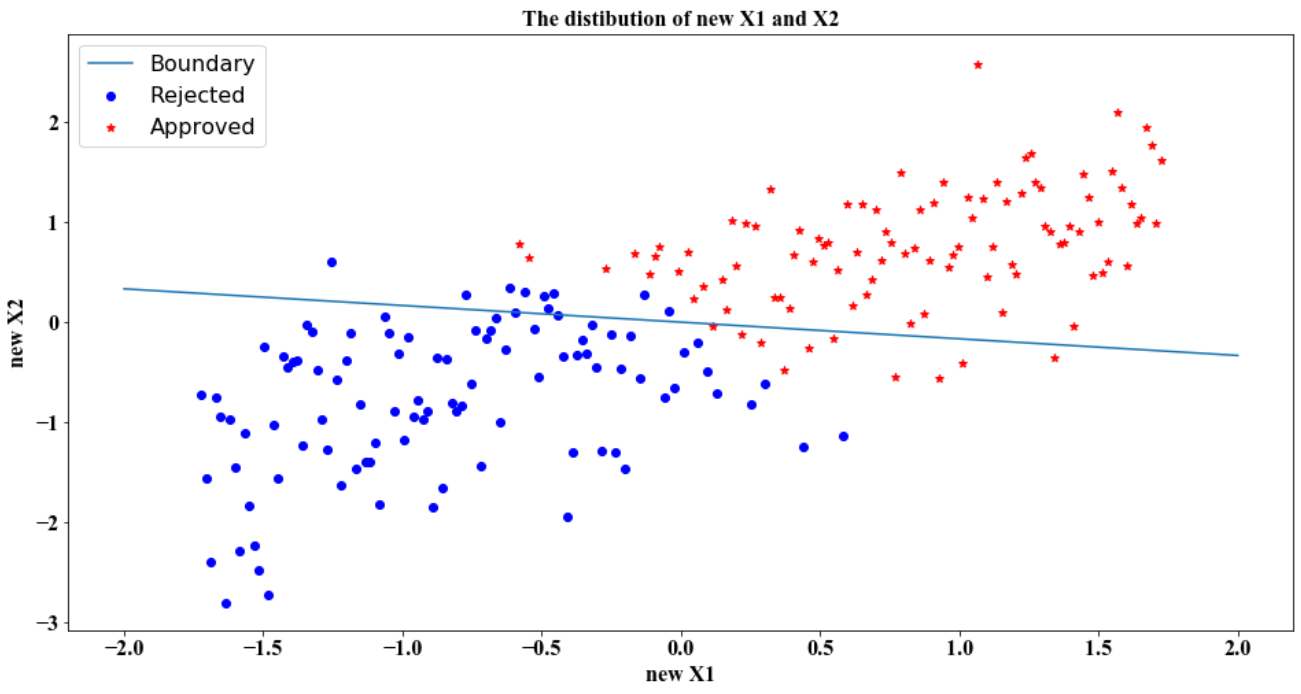
边界的直线由下面的等式来确定：

$\beta_1 xx + \beta_2 yy = 0$

It should be noted that xx and yy here only refers to the coordinates of horizontal or vertical axes. They don't refer to any predictor or response mentioned in the backgroun section. They are used for plotting the boundary only. No special meanings! Don't get confused! β_1 and β_2 are two parameters of our interest, both of which are obtained from gradient descent algorithm.

需要注意的是，上面等式里面的xx和yy仅仅指二维坐标系中的坐标值，仅作画边界直线用，与问题背景交代中的x和y没有任何关系。不要混淆了。β_1和β_2是我们所感兴趣的参数，由梯度下降得到。

```
# Two arbitrary values for beta_1 and beta_2
beta_1 = 0.1
beta_2 = 0.6

# Get the boundary
xx = np.linspace(-2, 2, 100)
yy = -beta_1 * xx / beta_2

# Plot the data points and boundary line at initial beta_0 and beta_1
fig = plt.figure(figsize=(16,8))
plt.scatter(new_X1[indicator < 0], new_X2[indicator < 0], color='b', marker='o',
label='Rejected')
plt.scatter(new_X1[indicator >= 0], new_X2[indicator >= 0], color='r', marker='*' ,
label='Approved')
plt.plot(xx, yy, label='Boundary')
font = {'family':'Times New Roman','weight' : 'normal','size': 16,}
plt.xlabel('new X1', font)
plt.ylabel('new X2', font)
plt.xticks(fontsize=16, fontname='Times New Roman')
plt.yticks(fontsize=16, fontname='Times New Roman')
plt.title('The distibution of new X1 and X2', font)
plt.legend(fontsize=16)
plt.show()
```



Not surprisingly, the line does not classify two types of data points well.

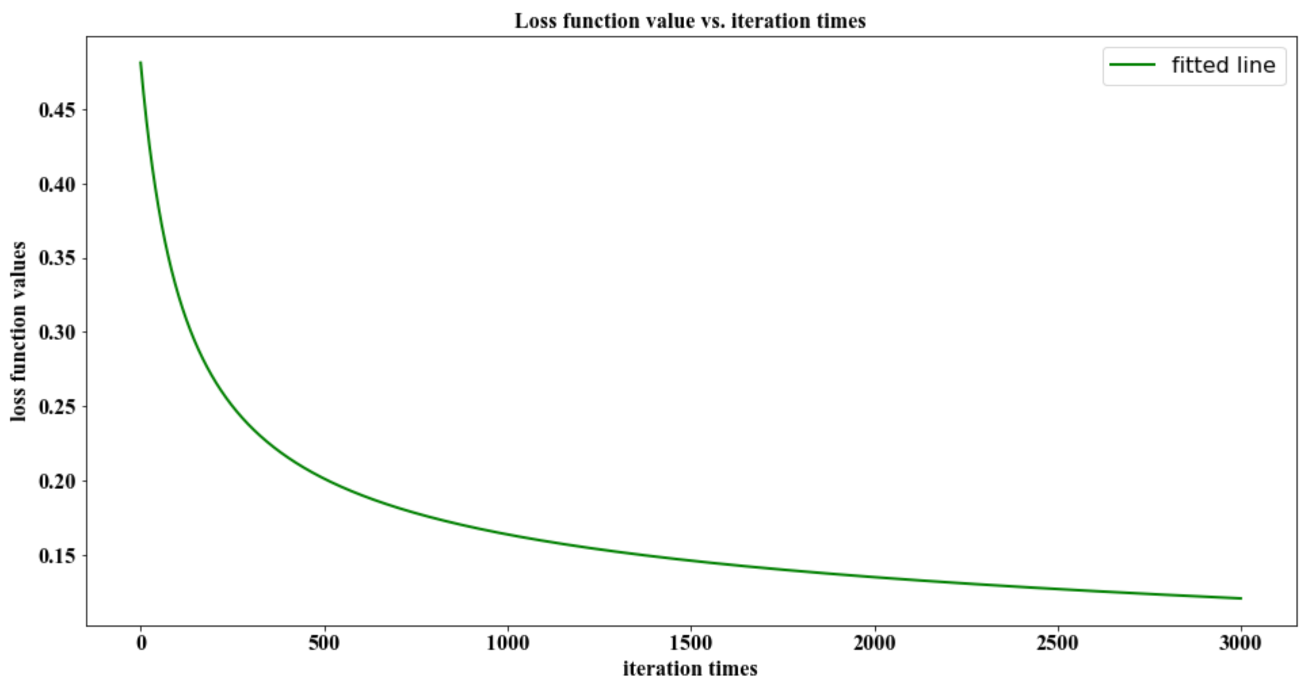
随意给出的这条直线并不能很好的划分两类数据点。

Now let's use gradient descent to update β_1 and β_2.

现在用梯度下降的方法来更新β_1和β_2的数值：

```
# Assume we are going to iterate 1000 times
max_iteration = 3000

# Define step size
step_size = 0.0001

# Initialize beta_1, beta_2, and loss to track the update algorithm
beta_1_update = np.zeros([max_iteration+1, 1])
beta_2_update = np.zeros([max_iteration+1, 1])
loss_update = np.zeros([max_iteration+1, 1])

# Initial value for beta_1 and beta_2
beta_1_update[0,0] = beta_1
beta_2_update[0,0] = beta_2

# Iterate many times in order to get the best values for beta_0 and beta_1
for iter in range(max_iteration):
    # Calculate h function results
    h = np.exp(new_X1 * beta_1_update[iter,0] + new_X2 * beta_2_update[iter,0]) / (1 +
np.exp(new_X1 * beta_1_update[iter,0] + new_X2 * beta_2_update[iter,0]))
    # Calculate the loss function value
    loss_update[iter,0] = -np.mean(Y * np.log(h) + (1-Y) * np.log(1-h))
    # Calculate the gradient
    gradient_beta_1 = -np.sum((Y-h) * X1)/number_observation
    gradient_beta_2 = -np.sum((Y-h) * X2)/number_observation
    # Update beta_1 and beta_2
    beta_1_update[iter+1] = beta_1_update[iter] - step_size * gradient_beta_1
    beta_2_update[iter+1] = beta_2_update[iter] - step_size * gradient_beta_2
```

```
# Plot the loss vs. iteration times
plt.figure(figsize=(16,8))
plt.plot(range(max_iteration), loss_update[0:max_iteration,0], color='g', linewidth=2.0,
label='fitted line')
font = {'family':'Times New Roman','weight' : 'normal','size': 16,}
plt.xlabel('iteration times', font)
plt.ylabel('loss function values', font)
plt.xticks(fontsize=16, fontname='Times New Roman')
plt.yticks(fontsize=16, fontname='Times New Roman')
plt.title('Loss function value vs. iteration times', font)
plt.legend(fontsize=16)
plt.show()
```



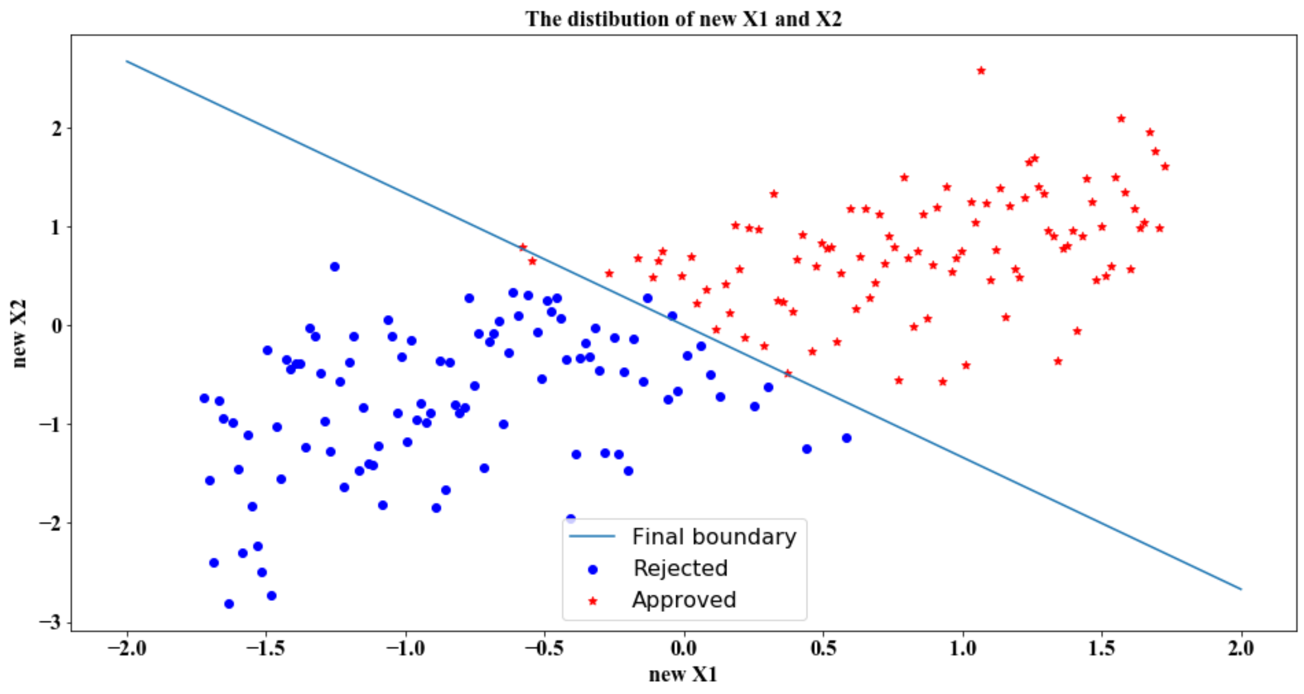
As shown in the figure above, the loss already reaches convergence, which means increasing the iteration times would not further decrease the loss.

从上图可以看出，损失值已达到收敛。这意味着进一步增加迭代次数不会减少损失值。

The final boundary and data points are shown below:

最终的边界如下图所示：

```
# Plot the data points and boundary line at final beta_0 and beta_1
xx = np.linspace(-2, 2, 100)
yy = -beta_1_update[-1,0] * xx / beta_2_update[-1,0]

fig = plt.figure(figsize=(16,8))
plt.scatter(new_X1[indicator < 0], new_X2[indicator < 0], color='b', marker='o',
label='Rejected')
plt.scatter(new_X1[indicator >= 0], new_X2[indicator >= 0], color='r', marker='*' ,
label='Approved')
plt.plot(xx, yy, label='Final boundary')
font = {'family':'Times New Roman','weight' : 'normal','size': 16,}
plt.xlabel('new X1', font)
plt.ylabel('new X2', font)
plt.xticks(fontsize=16, fontname='Times New Roman')
plt.yticks(fontsize=16, fontname='Times New Roman')
plt.title('The distibution of new X1 and X2', font)
plt.legend(fontsize=16)
plt.show()
```



As shwon in the figure above, the boundary separates the blue dots and red stars. Although there are some mis-classification points, these mis-classifications are inevitable as the original data points are not linear separable.

如上图所示，最终的边界直线可以较好的区分蓝色圆点和红色星状点。尽管有一些数据点没有被正确的划分，这种“错误”是无法避免的，因为我们原始的数据并不是被线性函数完全分割的(即两类数据点相互侵入对方区域)。

---------------------------------- end ---

Review (往期回顾):

Week 1 Linear regression (线性回归) https://blog.csdn.net/weixin_42515443/article/details/80768841