Report on Logistic Regression

XU Kaiyuan

31 Mai 2024

Table des matières

0 Introduction

Our objective is to write the code in Matlab about a simple extension of logistic regression according to the paper [2]. It takes the possibility of mislabeling directly into the objective by introducing sparse "movement parameters" to allow data points to move along the sigmoid function.

At the beginning of this report, we talk about the standard logistic regression, we will deduce the log loss function using the maximum likelihood estimation then we introduce the L2 regularisation for it to solve the overfitting problem. (**section 1**)

In the next section, we will talk about the robust extension for the Binary Logistic regression model which lets certain datapoints shift along the sigmaoid function. The mean ideal is to use a change of variable.

Then **Section 3** is about the simulation :

- In **Section 3.1** We will write by ourselves the log loss function and the gradient for the log loss function then we use the gradient descent method to solve a standard logistic regression problem.
- In **Section 3.2** We will also write by ourselves the gradient descent for a standard logistic regression problem. This time the L2 regularisation is added.
- In **Section 3.3** We will use the existing model in matlab : `glmfit` for the standard logistic regression and `lassoglm` for the robust binary Logistic regression model (L1 regularisation for the shift parameters $\mathbf{\Gamma}$ and the paramaeters $\mathbf{\Theta}$).

Finally all the code could be found in the annexes

1 Preliminary (Standard logistic regression)

1.1 From simple linear regression

We have studied the simple linear regression in UE 355 Optimisation where we use the least squares to solve :

$$y_i = \theta_0 + \theta_1 x_i$$

and we have also used the gradient descent (UE355 TP3 Gradient descent) to solve the linear system

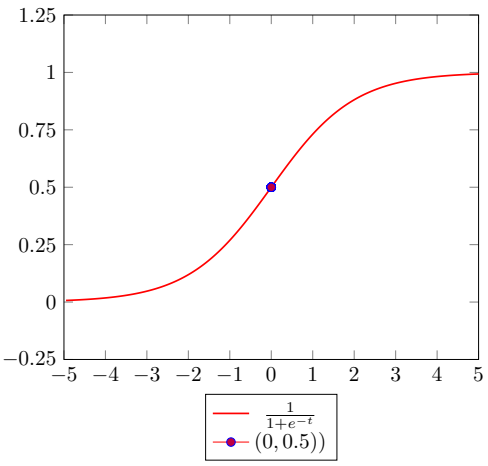
$$\mathbf{Ax} = \mathbf{b}$$

where $\mathbf{A}$ is a $(n \times n)$ matrix assumed to be **symmetric** ($A^T = A$) and **positive definite** ($\forall\ \mathbf{x} \neq \mathbf{0}$, $\mathbf{x}^T A\mathbf{x} > 0$) ; $\mathbf{b}$ is a $(n \times 1)$ vector that is assumed to be known. Here, the loss function is

$$\frac{\mathbf{x}^T\mathbf{Ax}}{\mathbf{x}^T\mathbf{x}} = \frac{\|\mathbf{x}\|_{\mathbf{A}}^2}{\|\mathbf{x}\|^2} \tag{1.1.1}$$

1.2 Sigmoid function

Considering the logistic regression, this time we want to map all values on $\mathbb{R}$ to 0 (false) or 1 (true). So here we introduce the sigmoid function $s(t)$:



$$s(t) = \frac{1}{1 + e^{-t}} \tag{1.2.1}$$

with the threshold equal to 0.5 normally (in blue).

1.3 Binary Logistic regression model

Considering the **Multiple Linear Regression Equation** :

$$y_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \dots \theta_m x_{im}$$

where $i = 1, 2, \dots, n$. It can also be written in the maticial form :

$$\mathbf{Y} = \mathbf{X\Theta}$$

with

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad , \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1m} \\ 1 & x_{21} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \cdots & x_{nm} \end{bmatrix} \quad , \quad \mathbf{\Theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

The submatrix

$$\begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

in $\mathbf{X}$ represents our training data of dimention $n \times m$: n is the number of samples, m is the number of features.

We note

$$\mathbf{X}_i{}^T = \begin{bmatrix} 1 & x_{i1} & x_{i2} & \dots & x_{im} \end{bmatrix}$$

the i-th samples.

Then we use the (1.2.1) with $t = \mathbf{X\Theta}$, the probability of each example being positive is modeled as :

$$\mathbf{P}(\mathbf{Y} = 1 | \mathbf{X}, \mathbf{\Theta}) = s(\mathbf{X\Theta}) = \frac{1}{1 + e^{-\mathbf{X\Theta}}} \tag{1.3.1}$$

and the probability of each example being negative is modeled as :

$$\mathbf{P}(\mathbf{Y} = 0 | \mathbf{X}, \mathbf{\Theta}) = 1 - \mathbf{P}(\mathbf{Y} = 1 | \mathbf{X}, \mathbf{\Theta}) \tag{1.3.2}$$

Since we have computed the logistic function for multiple variables, now we should try to find the vector Θ.

1.4 Maximum likelihood estimation

Probability describes how likely an event is to occur. While, we use **Maximum likelihood** when the event has already happened, then we want to compute the parameters that are more likely to have this event happes.

Known the feature $\mathbf{X}$ and parameter $\mathbf{\Theta}$, the conditional probability of predicting the occurrence of an event $\mathbf{Y}$ is

$$\mathbf{P}(\mathbf{Y}|\mathbf{X}, \mathbf{\Theta})$$

Known an event $\mathbf{Y}$ which has already happened, the likelihood function ($\mathcal{L}$) of the unknown parameter $\mathbf{\Theta}$ is :

$$\mathcal{L}(\mathbf{\Theta}|\mathbf{X})$$

we have the equality :

$$\mathcal{L}(\mathbf{\Theta}|\mathbf{X}) = \mathbf{P}(\mathbf{Y}|\mathbf{X}, \mathbf{\Theta}) \tag{1.4.1}$$

$\mathcal{L}(\mathbf{\Theta}|\mathbf{X})$ a function of $\mathbf{\Theta}$, now we want to find the optimal parameter $\mathbf{\Theta}_{opt}$ so that the likelihood function reaches $\boxed{maximum}$. It means the event is more likely to occur in this parameter.

Since in our case, we use the Binary Logistic regression model, which means $\mathbf{Y}$ is ether 0 or 1 so that :

$$\mathbf{P}(\mathbf{Y}|\mathbf{X}, \mathbf{\Theta}) = \mathbf{P}(\mathbf{Y} = 1|\mathbf{X}, \mathbf{\Theta})^{\mathbf{Y}}\ \mathbf{P}(\mathbf{Y} = 0|\mathbf{X}, \mathbf{\Theta})^{\mathbf{1-Y}}$$

Using (1.3.1) and (1.3.2), we have :

$$\mathbf{P}(\mathbf{Y}|\mathbf{X}, \mathbf{\Theta}) = \left(\frac{1}{1 + e^{-\mathbf{X\Theta}}}\right)^{\mathbf{Y}} \left(1 - \frac{1}{1 + e^{-\mathbf{X\Theta}}}\right)^{\mathbf{1-Y}}$$

or for the i-th sample :

$$\mathbf{P}(y_i|\mathbf{X}_i{}^T, \mathbf{\Theta}) = \mathbf{P}(y_i = 1|\mathbf{X}_i{}^T, \mathbf{\Theta})^{y_i}\ \mathbf{P}(y_i = 0|\mathbf{X}_i{}^T, \mathbf{\Theta})^{1-y_i} \tag{1.4.2}$$

Using (1.3.1) and (1.3.2), we have :

$$\mathbf{P}(y_i|\mathbf{X}_i{}^T, \mathbf{\Theta}) = \left(\frac{1}{1 + e^{-\mathbf{X}_i{}^T\mathbf{\Theta}}}\right)^{y_i} \left(1 - \frac{1}{1 + e^{-\mathbf{X}_i{}^T\mathbf{\Theta}}}\right)^{1-y_i}$$

with $i = 1, 2, \ldots, n$

So that probability of the obsevations $y_1, y_2, \ldots, y_n$ occurring at the same time is :

$$\prod_{i=1}^{n} \mathbf{P}(y_i|\mathbf{X}_i{}^T, \mathbf{\Theta})$$

Since (1.4.1), know find the optimal parameter $\mathbf{\Theta}_{opt}$ so that the function $\underline{\mathcal{L}(\mathbf{\Theta}|\mathbf{X}) \text{ or notation simplied } \mathcal{L}(\mathbf{\Theta})}$

$$\begin{aligned}
\mathcal{L}(\mathbf{\Theta}) &= \mathcal{L}(\mathbf{\Theta}|\mathbf{X}) \\
&= \mathbf{P}(\mathbf{Y}|\mathbf{X}, \mathbf{\Theta}) \\
&= \prod_{i=1}^{n} \mathbf{P}(y_i|\mathbf{X}_i{}^T, \mathbf{\Theta})
\end{aligned}$$

reaches maximum. We use log to reduce the product to a sum :

$$\begin{aligned}
\log \mathcal{L}(\mathbf{\Theta}) &= \log \prod_{i=1}^{n} \mathbf{P}(y_i|\mathbf{X}_i{}^T, \mathbf{\Theta}) \\
&= \sum_{i=1}^{n} \log \mathbf{P}(y_i|\mathbf{X}_i{}^T, \mathbf{\Theta}) \\
&= \sum_{i=1}^{n} \log \left[\mathbf{P}(y_i = 1|\mathbf{X}_i{}^T, \mathbf{\Theta})^{y_i}\ \mathbf{P}(y_i = 0|\mathbf{X}_i{}^T, \mathbf{\Theta})^{1-y_i}\right] \quad \text{used (1.4.2)} \\
&= \sum_{i=1}^{n} \left[y_i \log \mathbf{P}(y_i = 1|\mathbf{X}_i{}^T, \mathbf{\Theta}) + (1 - y_i) \log \mathbf{P}(y_i = 0|\mathbf{X}_i{}^T, \mathbf{\Theta})\right] \\
&= \sum_{i=1}^{n} \left[y_i \log \mathbf{P}(y_i = 1|\mathbf{X}_i{}^T, \mathbf{\Theta}) + (1 - y_i) \log(1 - \mathbf{P}(y_i = 1|\mathbf{X}_i{}^T, \mathbf{\Theta}))\right] \quad \text{used (1.3.2)} \\
&= \sum_{i=1}^{n} \left[y_i \log s(\mathbf{X}_i{}^T\mathbf{\Theta}) + (1 - y_i) \log(1 - s(\mathbf{X}_i{}^T\mathbf{\Theta}))\right] \quad \text{used (1.3.1)}
\end{aligned}$$

1.5 The log loss function

For the simple linear regression, we use the residual sum of squares to calcule the loss function

$$\sum_{i}^{n} (y_i - \theta_1 x_i)^2$$

also for the system $\mathbf{Ax} = \mathbf{b}$ in preliminary :

$$\frac{\|\mathbf{x}\|_{\mathbf{A}}^2}{\|\mathbf{x}\|^2}$$

If we use the square loss to compute the loss function for the logistic regression, it would be something like :

$$\sum_{i}^{n} \left(y_i - s({\mathbf{X}_i}^T \mathbf{\Theta})\right)^2$$

Unfortunately, this is not a convex function, we might find the local but not global maximum in the optimisation.

Actually, we define the `log loss function` as

$$\begin{aligned} l(\Theta) &:= -\log \mathcal{L}(\mathbf{\Theta}) \\ &= -\sum_{i=1}^{n} \left[y_i \log s({\mathbf{X}_i}^T \mathbf{\Theta}) + (1 - y_i) \log(1 - s({\mathbf{X}_i}^T \mathbf{\Theta})) \right] \end{aligned} \tag{1.5.1}$$

since we notice the $l(\Theta)$ is convexe, we are able to use the methode **Gradient descent** to find the minimum global of $l(\Theta)$ which is equal to get the *maximum* global of $\mathcal{L}(\Theta)$.

To compute the gradient for the `log loss function` $\frac{\partial}{\partial \mathbf{\Theta}_j} l(\Theta)$:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{\Theta}_j} l(\Theta) &= \frac{\partial}{\partial \mathbf{\Theta}_j} - \sum_{i=1}^{n} \left[y_i \log s({\mathbf{X}_i}^T \mathbf{\Theta}) + (1 - y_i) \log(1 - s({\mathbf{X}_i}^T \mathbf{\Theta})) \right] \\ &= \frac{\partial}{\partial \mathbf{\Theta}_j} - \sum_{i=1}^{n} \left[y_i \log \frac{1}{1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}} + (1 - y_i) \log \left(1 - \frac{1}{1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}} \right) \right] \\ &= \frac{\partial}{\partial \mathbf{\Theta}_j} - \sum_{i=1}^{n} \left[-y_i \log \left(1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}} \right) - (1 - y_i) \log \left(1 + e^{+{\mathbf{X}_i}^T \mathbf{\Theta}} \right) \right] \\ &= -\sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{\Theta}_j} \left[-y_i \log \left(1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}} \right) - (1 - y_i) \log \left(1 + e^{+{\mathbf{X}_i}^T \mathbf{\Theta}} \right) \right] \\ &= -\sum_{i=1}^{n} \left[-y_i \frac{-{\mathbf{X}_i}^T{}_j \; e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}}{1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}} - (1 - y_i) \frac{+{\mathbf{X}_i}^T{}_j \; e^{+{\mathbf{X}_i}^T \mathbf{\Theta}}}{1 + e^{+{\mathbf{X}_i}^T \mathbf{\Theta}}} \right] \\ &= -\sum_{i=1}^{n} \left[\left(+y_i \frac{e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}}{1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}} - (1 - y_i) \frac{e^{+{\mathbf{X}_i}^T \mathbf{\Theta}}}{1 + e^{+{\mathbf{X}_i}^T \mathbf{\Theta}}} \right) {\mathbf{X}_i}^T{}_j \right] \\ &= -\sum_{i=1}^{n} \left[\left(y_i \frac{e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}}{1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}} - (1 - y_i) \frac{1}{1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}} \right) {\mathbf{X}_i}^T{}_j \right] \\ &= -\sum_{i=1}^{n} \left[\left(y_i - \frac{1}{1 + e^{-{\mathbf{X}_i}^T \mathbf{\Theta}}} \right) {\mathbf{X}_i}^T{}_j \right] \\ &= -\sum_{i=1}^{n} \left[\left(y_i - s({\mathbf{X}_i}^T \mathbf{\Theta}) \right) {\mathbf{X}_i}^T{}_j \right] \\ &= \sum_{i=1}^{n} \left[\left(s({\mathbf{X}_i}^T \mathbf{\Theta}) - y_i \right) {\mathbf{X}_i}^T{}_j \right] \end{aligned} \tag{1.5.2}$$

where ${\mathbf{X}_i}^T{}_j$ is the j-th element of vector ${\mathbf{X}_i}^T$

Now we take any Θ^0 and apply the following iteration :

$$\mathbf{\Theta}^{k+1} = \mathbf{\Theta}^k - \alpha \frac{\partial}{\partial \mathbf{\Theta}_j} l(\Theta^k)$$

with a constant step α.

1.6 L2 regularisation

Regularisation helps to solve the overfitting problem, we recall the log loss function for L2 regularized logistic regression :

$$l_2(\Theta) = -\sum_{i=1}^{n} \Big[y_i \log s(\mathbf{X}_i{}^T \boldsymbol{\Theta}) + (1-y_i) \log(1 - s(\mathbf{X}_i{}^T \boldsymbol{\Theta})) \Big] + \frac{1}{2\sigma^2} \sum_{j=0}^{m} |\theta_j|^2 \tag{1.6.1}$$

where σ is related to the variance.

Similarly, the gradient for the `log loss function` $\frac{\partial}{\partial \boldsymbol{\Theta}_j} l_2(\Theta)$:

$$\frac{\partial}{\partial \boldsymbol{\Theta}_j} l_2(\Theta) = \sum_{i=1}^{n} \Big[\Big(s(\mathbf{X}_i{}^T \boldsymbol{\Theta}) - y_i \Big) \mathbf{X}_i{}^T{}_j \Big] + \frac{1}{\sigma^2} \sum_{j=0}^{m} \theta_j \tag{1.6.2}$$

since $\frac{\partial}{\partial \boldsymbol{\Theta}_j} \theta_j{}^T \theta_j = 2\theta_j$

2 Robust extension for the Binary Logistic regression model

We introduce a real-valued shift parameter $\boldsymbol{\Gamma} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_n \end{bmatrix}$. These parameters let certain datapoints shift along the sigmoid, or

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\Theta} + \boldsymbol{\Gamma}$$

the probability of each example being positive is modeled as :

$$\mathbf{P}(\mathbf{Y} = 1 | \mathbf{X}, \boldsymbol{\Theta}) = s(\mathbf{X}\boldsymbol{\Theta} + \boldsymbol{\Gamma}) = \frac{1}{1 + e^{-\mathbf{X}\boldsymbol{\Theta} - \boldsymbol{\Gamma}}}$$

we have now the log loss function as :

$$l(\Theta) = -\sum_{i=1}^{n} \Big[y_i \log s(\mathbf{X}_i{}^T \boldsymbol{\Theta} + \boldsymbol{\Gamma}_i) + (1-y_i) \log(1 - s(\mathbf{X}_i{}^T \boldsymbol{\Theta} + \boldsymbol{\Gamma}_i)) \Big]$$

L_1-regularize the shift parameters $\boldsymbol{\Gamma}$

We L_1-regularize the shift parameters $\boldsymbol{\Gamma}$ to encourage sparsity :

$$\boxed{l(\boldsymbol{\Theta}, \boldsymbol{\Gamma}) = -\sum_{i=1}^{n} \Big[y_i \log s(\mathbf{X}_i{}^T \boldsymbol{\Theta} + \boldsymbol{\Gamma}_i) + (1-y_i) \log(1 - s(\mathbf{X}_i{}^T \boldsymbol{\Theta} + \boldsymbol{\Gamma}_i)) \Big] + \lambda \sum_{i=1}^{n} |\gamma_i|}$$

Regarding (1.5.1), if we use a change of variable, let

$$\tilde{\boldsymbol{\Theta}} = \begin{bmatrix} \boldsymbol{\Theta} \\ - \\ \boldsymbol{\Gamma} \end{bmatrix} \quad , \quad \tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X} & | & \mathbf{I}_n \end{bmatrix}$$

we have

$$l(\tilde{\Theta}) = -\sum_{i=1}^{n} \Big[y_i \log s(\tilde{\mathbf{X}}_i \tilde{\Theta}) + (1-y_i) \log(1 - s(\tilde{\mathbf{X}}_i \tilde{\Theta})) \Big] + \lambda \sum_{j=m+1}^{m+n} |\tilde{\boldsymbol{\Theta}}_j|$$

with the configuration $\tilde{\boldsymbol{\Theta}}_0 = \theta_0$ (first line of the vector $\tilde{\boldsymbol{\Theta}}$) which means $\tilde{\boldsymbol{\Theta}}_1 = \theta_1$ and so on.

Finally, with a vector of penalty factors

$$\tilde{\mathbf{F}} = \begin{bmatrix} \mathbf{Zeros}(m+1, 1) \\ \mathbf{Ones}(n, 1) \end{bmatrix}$$

where $\mathbf{Zeros}(m+1, 1)$ means a column-vector with $m+1$ element 0 and $\mathbf{Ones}(n, 1)$ means a column-vector with n element 1. We have :

$$\boxed{l(\tilde{\Theta}) = -\sum_{i=1}^{n} \Big[y_i \log s(\tilde{\mathbf{X}}_i \tilde{\Theta}) + (1-y_i) \log(1 - s(\tilde{\mathbf{X}}_i \tilde{\Theta})) \Big] + \lambda \sum_{j=0}^{m+n} \tilde{\mathbf{F}}_\mathbf{j} |\tilde{\boldsymbol{\Theta}}_j|}$$

Exemple 2.0.1

Let's take a training data with $n = 4$ samples and $m = 2$ features for exemple :

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \\ 1 & x_{41} & x_{42} \end{bmatrix} \quad , \quad \mathbf{\Theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \quad , \quad \mathbf{\Gamma} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix}$$

From one part, we have

$$\mathbf{X\Theta} + \mathbf{\Gamma} = \begin{bmatrix} \theta_0 + \theta_1 x_{11} + \theta_2 x_{12} \\ \theta_0 + \theta_1 x_{21} + \theta_2 x_{22} \\ \theta_0 + \theta_1 x_{31} + \theta_2 x_{32} \\ \theta_0 + \theta_1 x_{41} + \theta_2 x_{42} \end{bmatrix} + \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix}$$

and

$$\sum_{i=1}^{n} |\gamma_i| = \sum_{i=1}^{4} |\gamma_i| = |\gamma_1| + |\gamma_2| + |\gamma_3| + |\gamma_4|$$

Now we use a change of variable :

$$\tilde{\mathbf{X}} = \begin{bmatrix} 1 & x_{11} & x_{12} & 1 & 0 & 0 & 0 \\ 1 & x_{21} & x_{22} & 0 & 1 & 0 & 0 \\ 1 & x_{31} & x_{32} & 0 & 0 & 1 & 0 \\ 1 & x_{41} & x_{42} & 0 & 0 & 0 & 1 \end{bmatrix} \quad , \quad \tilde{\mathbf{\Theta}} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix} \quad , \quad \tilde{\mathbf{F}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

From the other, we have

$$\tilde{\mathbf{X}}\tilde{\mathbf{\Theta}} = \begin{bmatrix} \theta_0 + \theta_1 x_{11} + \theta_2 x_{12} + \gamma_1 \\ \theta_0 + \theta_1 x_{21} + \theta_2 x_{22} + \gamma_2 \\ \theta_0 + \theta_1 x_{31} + \theta_2 x_{32} + \gamma_3 \\ \theta_0 + \theta_1 x_{41} + \theta_2 x_{42} + \gamma_4 \end{bmatrix}$$

and

$$\sum_{j=0}^{m+n} \tilde{\mathbf{F}}_\mathbf{j} |\tilde{\mathbf{\Theta}}_j| = \sum_{j=m+1}^{n} |\mathbf{\Theta}_j| = \sum_{j=2+1}^{2+4} |\mathbf{\Theta}_j| = |\gamma_1| + |\gamma_2| + |\gamma_3| + |\gamma_4|$$

Thus

$$\mathbf{X\Theta} + \mathbf{\Gamma} = \tilde{\mathbf{X}}\tilde{\mathbf{\Theta}}$$

and

$$\sum_{i=1}^{n} |\gamma_i| = \sum_{j=0}^{m+n} \tilde{\mathbf{F}}_\mathbf{j} |\tilde{\mathbf{\Theta}}_j|$$

this explains why the change of variable works.

Then L_1-regularize the parameters $\mathbf{\Theta}$

Then we chose to use an L_1-regularize penalty to $\mathbf{\Theta}$:

$$\boxed{l(\mathbf{\Theta}, \mathbf{\Gamma}) = -\sum_{i=1}^{n} \Big[y_i \log s(\mathbf{X}_i{}^T \mathbf{\Theta} + \mathbf{\Gamma}_i) + (1 - y_i) \log(1 - s(\mathbf{X}_i{}^T \mathbf{\Theta} + \mathbf{\Gamma}_i)) \Big] + \kappa \sum_{j=0}^{m} |\theta_j| + \lambda \sum_{i=1}^{n} |\gamma_i|}$$

or

$$l(\mathbf{\Theta}, \mathbf{\Gamma}) = -\sum_{i=1}^{n} \Big[y_i \log s(\mathbf{X}_i{}^T \mathbf{\Theta} + \mathbf{\Gamma}_i) + (1 - y_i) \log(1 - s(\mathbf{X}_i{}^T \mathbf{\Theta} + \mathbf{\Gamma}_i)) \Big] + \kappa \left(\sum_{j=0}^{m} |\theta_j| + \frac{\lambda}{\kappa} \sum_{i=1}^{n} |\gamma_i| \right)$$

Remarque

We can also chose to use an L_2-regularize penalty to $\mathbf{\Theta}$:

$$l(\mathbf{\Theta}, \mathbf{\Gamma}) = -\sum_{i=1}^{n} \Big[y_i \log s(\mathbf{X}_i{}^T \mathbf{\Theta} + \mathbf{\Gamma}_i) + (1 - y_i) \log(1 - s(\mathbf{X}_i{}^T \mathbf{\Theta} + \mathbf{\Gamma}_i)) \Big] + \frac{1}{2\sigma^2} \sum_{j=0}^{m} |\theta_j|^2 + \lambda \sum_{i=1}^{n} |\gamma_i|$$

Regarding (1.5.1), this time if we use a change of variable, just let

$$\tilde{\mathbf{\Theta}} = \begin{bmatrix} \mathbf{\Theta} \\ - \\ \frac{\lambda}{\kappa}\mathbf{\Gamma} \end{bmatrix} \quad , \quad \tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X} & | & \frac{\kappa}{\lambda}\mathbf{I}_n \end{bmatrix} \quad , \quad \tilde{\mathbf{F}} = \begin{bmatrix} \mathbf{Ones}(m+1, 1) \\ \mathbf{Ones}(n, 1) \end{bmatrix} = \begin{bmatrix} \mathbf{Ones}(m+1+n, 1) \end{bmatrix}$$

we have

$$\boxed{l(\tilde{\Theta}) = -\sum_{i=1}^{n} \Big[y_i \log s(\tilde{\mathbf{X}}_i \tilde{\Theta}) + (1 - y_i) \log(1 - s(\tilde{\mathbf{X}}_i \tilde{\Theta})) \Big] + \kappa \sum_{j=0}^{m+n} \tilde{\mathbf{F}}_\mathbf{j} |\tilde{\mathbf{\Theta}}_j|}$$

Exemple 2.0.2

Let's take a training data with $n = 4$ samples and $m = 2$ features for exemple :

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \\ 1 & x_{41} & x_{42} \end{bmatrix} \quad , \quad \mathbf{\Theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \quad , \quad \mathbf{\Gamma} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix}$$

From one part, we have

$$\mathbf{X\Theta} + \mathbf{\Gamma} = \begin{bmatrix} \theta_0 + \theta_1 x_{11} + \theta_2 x_{12} \\ \theta_0 + \theta_1 x_{21} + \theta_2 x_{22} \\ \theta_0 + \theta_1 x_{31} + \theta_2 x_{32} \\ \theta_0 + \theta_1 x_{41} + \theta_2 x_{42} \end{bmatrix} + \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix}$$

and

$$\kappa \sum_{j=0}^{m} |\theta_j| = \kappa \sum_{j=0}^{2} |\theta_j| = \kappa|\theta_0| + \kappa|\theta_1| + \kappa|\theta_2|$$

$$\lambda \sum_{i=1}^{n} |\gamma_i| = \lambda \sum_{i=1}^{4} |\gamma_i| = \lambda|\gamma_1| + \lambda|\gamma_2| + \lambda|\gamma_3| + \lambda|\gamma_4|$$

Now we use a change of variable :

$$\tilde{\mathbf{X}} = \begin{bmatrix} 1 & x_{11} & x_{12} & \frac{\kappa}{\lambda} & 0 & 0 & 0 \\ 1 & x_{21} & x_{22} & 0 & \frac{\kappa}{\lambda} & 0 & 0 \\ 1 & x_{31} & x_{32} & 0 & 0 & \frac{\kappa}{\lambda} & 0 \\ 1 & x_{41} & x_{42} & 0 & 0 & 0 & \frac{\kappa}{\lambda} \end{bmatrix} \quad , \quad \tilde{\mathbf{\Theta}} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \frac{\lambda}{\kappa}\gamma_1 \\ \frac{\lambda}{\kappa}\gamma_2 \\ \frac{\lambda}{\kappa}\gamma_3 \\ \frac{\lambda}{\kappa}\gamma_4 \end{bmatrix} \quad , \quad \tilde{\mathbf{F}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

From the other, we have

$$\tilde{\mathbf{X}}\tilde{\mathbf{\Theta}} = \begin{bmatrix} \theta_0 + \theta_1 x_{11} + \theta_2 x_{12} + \gamma_1 \\ \theta_0 + \theta_1 x_{21} + \theta_2 x_{22} + \gamma_2 \\ \theta_0 + \theta_1 x_{31} + \theta_2 x_{32} + \gamma_3 \\ \theta_0 + \theta_1 x_{41} + \theta_2 x_{42} + \gamma_4 \end{bmatrix}$$

and

$$\kappa \sum_{j=0}^{m+n} \tilde{\mathbf{F}}_\mathbf{j} |\tilde{\mathbf{\Theta}}_j| = \kappa \left(|\theta_0| + |\theta_1| + |\theta_2| + \frac{\lambda}{\kappa}|\gamma_1| + \frac{\lambda}{\kappa}|\gamma_2| + \frac{\lambda}{\kappa}|\gamma_3| + \frac{\lambda}{\kappa}|\gamma_4| \right)$$

Thus

$$\mathbf{X\Theta} + \mathbf{\Gamma} = \tilde{\mathbf{X}}\tilde{\mathbf{\Theta}}$$

and

$$\kappa \sum_{j=0}^{m} |\theta_j| + \lambda \sum_{i=1}^{n} |\gamma_i| = \kappa \sum_{j=0}^{m+n} \tilde{\mathbf{F}}_\mathbf{j} |\tilde{\mathbf{\Theta}}_j|$$

this explains why the change of variable works.

3 Simulation

3.1 From formula to Matlab code : standard logistic regression

This subsection explains how to convert formula to Matlab code for standard logistic regression. To verify that the code we wrote is correct, we use the database from [1] since the exercises in this tutorial give the results of the run.

ex2data1.txt (snippet)

Listing 1 – ex2data1.txt (snippet)

```
34.62365962451697,78.0246928153624,0
30.28671076822607,43.89499752400101,0
35.84740876993872,72.90219802708364,0
60.18259938620976,86.30855209546826,1
79.0327360507101,75.3443764369103,1
...
```

We want to compute a classification model that estimates an applicant's probability of admission based on the scores from those two exams. The first column of the data represents the results of the first exam, the second column represents the results of the second exam, and the third column represents whether or not you were accepted (1 : admitted ; 0 : not admitted). This dataset has 100 samples (rows) and 2 features (first two columns).

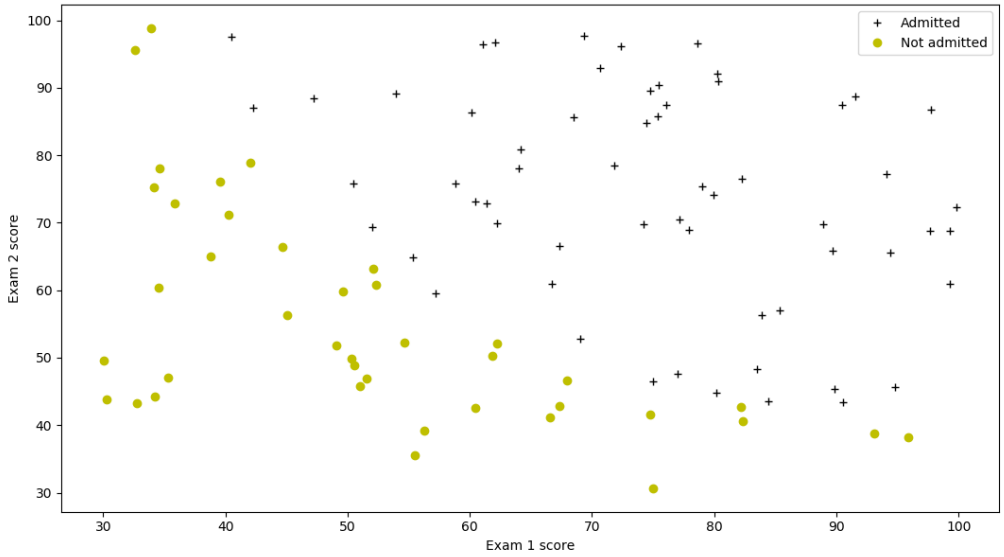


FIGURE 3.1.1 – A classification model that estimates an applicant's probability of admission based on the scores from those two exams

1. The sigmoid function (1.2.1) can be written as following code :

$$s(t) = \frac{1}{1 + e^{-t}}$$

Listing 2 – Sigmoid function

```
s = @(t) 1 ./ (1 + exp(-t));
% figure(1); x = linspace(-5,+5,100); y = s(x); plot(x,y)
```

2. Now we read this dataset, remember we need to add a column of 1 in matrix $\mathbf{X}$ for the bais θ_0 :

Listing 3 – Read dataset

```
data = readmatrix('ex2data1.txt');
X0_train = data(:, 1:2);
Y_train = data(:, 3);
```

```
4
5  [n, m] = size(X0_train); % n samples; m features
6  In1 = ones(n,1);
7  X_train = [In1, X0_train]; % add one column of 1 for the bais \theta_0
```

3. The log loss function (1.5.1) can be written as following code :

$$l(\Theta) = -\sum_{i=1}^{n} \left[y_i \log s({\mathbf{X}_i}^T \mathbf{\Theta}) + (1 - y_i) \log(1 - s({\mathbf{X}_i}^T \mathbf{\Theta})) \right]$$

Listing 4 – Log loss function

```
1  loss_function = @(X, Y, Theta, lambda_) ...
2      sum(arrayfun(@(i) ...
3      -Y(i) * log(s((X(i, :) * Theta))) - ...
4      (1 - Y(i)) * log(1 - s((X(i, :) * Theta))), 1:size(X, 1)));
5
6  % Theta = zeros(m+1,1);
7  % loss_function(X_train, Y_train, Theta)
```

4. The gradient for log loss function (1.5.2) can be written as following code :

$$\frac{\partial}{\partial \mathbf{\Theta}_j} l(\Theta) = \sum_{i=1}^{n} \left[\left(s({\mathbf{X}_i}^T \mathbf{\Theta}) - y_i \right) {\mathbf{X}_i}^T{}_j \right]$$

Listing 5 – Gradient for log loss function

```
1  gradient_loss_function = @(X, y, Theta, lambda_) ...
2      sum(cell2mat(arrayfun(@(i) (s(X(i, :) * Theta) - y(i)) * X(i, :)', 1:size(X, 1), '
           UniformOutput', false)), 2);
3
4  % cell2mat() : convert the cell array to matrix
5  % sum( , 2) : the summed gradients for each feature
6
7  % gradient_loss_function(X_train, Y_train, Theta)
```

5. Apply the gradient descent method using the above function to find the optimal parameters of a standard logistic regression model :

$$\mathbf{\Theta}^{k+1} = \mathbf{\Theta}^k - \alpha \frac{\partial}{\partial \mathbf{\Theta}_j} l(\Theta^k)$$

Listing 6 – Gradient descent method

```
1  % Some gradient descent settings & stop test value
2
3  k = 1;
4  k_max = 20000; % iterations
5
6  alpha = 0.00001; % learning rate
7
8  Theta = [-8; 0.01 * (rand(2, 1) - 0.5)]; % We take any \Theta^0
9  l_history = zeros(k_max, 1);
10 Theta_history = zeros(k_max, length(Theta));
11
12 epsilon_l(k) = 1; % for stop test value
13 epsilon_l_k_min = 1e-8; % relative variation of the criterion
14
15 while k < k_max && abs(epsilon_l(k)) > epsilon_l_k_min
16
17     k = k + 1;
18
19     % Calculate the gradient
20     dl_dTheta = gradient_loss_function(X_train, Y_train, Theta);
21
22     % Update parameters using alpha and gradient
23     Theta = Theta - alpha * dl_dTheta;
```

```
24
25      % Save cost J at each iteration
26      l_history(k) = loss_function(X_train, Y_train, Theta);
27
28      % Save Theta at each iteration
29      Theta_history(k, :) = Theta';
30
31      % Print cost every 10% of the iterations
32      if mod(k, ceil(k_max / 10)) == 0 || k == k_max
33          fprintf('Iteration %4d/%4d: Cost %8.2f\n', k, k_max, l_history(k));
34      end
35
36      % Relative variation of the criterion
37      epsilon_l(k) = (l_history(k-1) - l_history(k)) / l_history(k-1);
38  end
```

6. Now we apply the parameters we have found to our training set to see how well the learned model predicts by comparing with the third columns.

Listing 7 – Evaluating standard logistic regression model

```
1  Theta_opt = Theta_history(k,:)';
2  Y_prediction_0to1 = s(X_train*Theta_opt); % use sigmoid fuinction
3  Y_prediction_0or1(Y_prediction_0to1 < 0.5) = 0; % threshold = 0.5
4  Y_prediction_0or1(Y_prediction_0to1 >= 0.5) = 1;
5  Y_prediction_0or1 = Y_prediction_0or1';
6
7  % Compute accuracy on our training set
8  accuracy = mean(Y_prediction_0or1 == Y_train) * 100;
9  fprintf('Train Accuracy: %f%%\n', accuracy);
```

Here, we choose the threshold = 0.5. We use `accuracy = mean(Y_prediction_0or1 == Y_train) * 100;` to calculate the proportion of matches between the prediction $\mathbf{Y}_{prediction}$ and the training data $\mathbf{Y}$. Finally, we find `Train Accuracy: 92.000000%` which matches the answers to the exercises.

7. We export the data using

`writematrix(Theta_opt,'Theta_opt.txt')`

Then we used a function written in Python to plot the decision boundary.

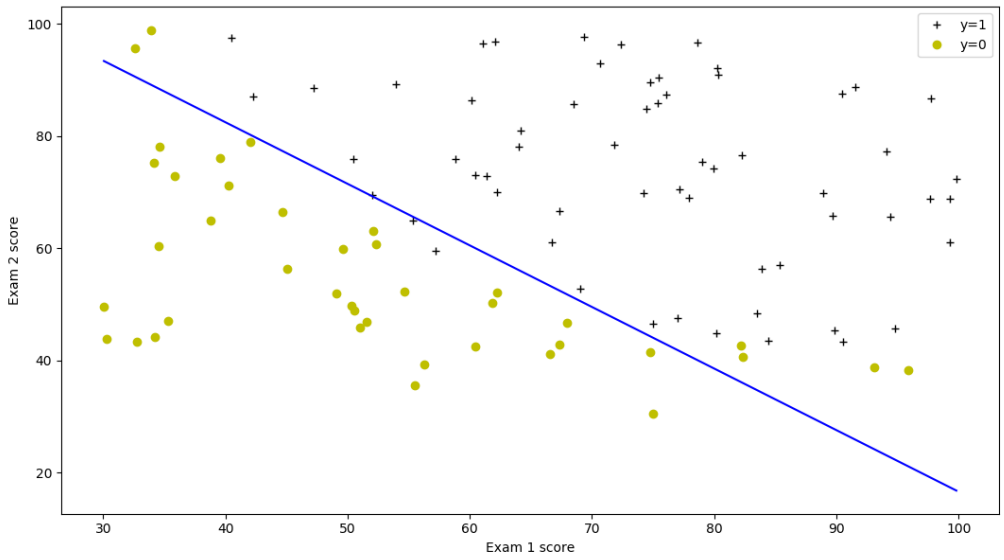


FIGURE 3.1.2 – The decision boundary

3.2 From formula to Matlab code : standard logistic regression + L2 regularisation

This subsection explains how to convert formula to Matlab code for standard logistic regression where L2 regularisation is used. To verify that the code we wrote is correct, we also use the database from [1] since the exercises in this tutorial give

the results of the run.

ex2data2.txt (snippet)

Listing 8 – ex2data2.txt (snippet)

```
0.051267,0.69956,1
-0.092742,0.68494,1
-0.21371,0.69225,1
-0.375,0.50219,1
-0.51325,0.46564,1
...
```

This is the result for some microchips on two different tests. According to the these two tests determine whether the microchips should be accepted (1) or rejected (0). This dataset has 118 samples (rows) and 2 features (first two columns).

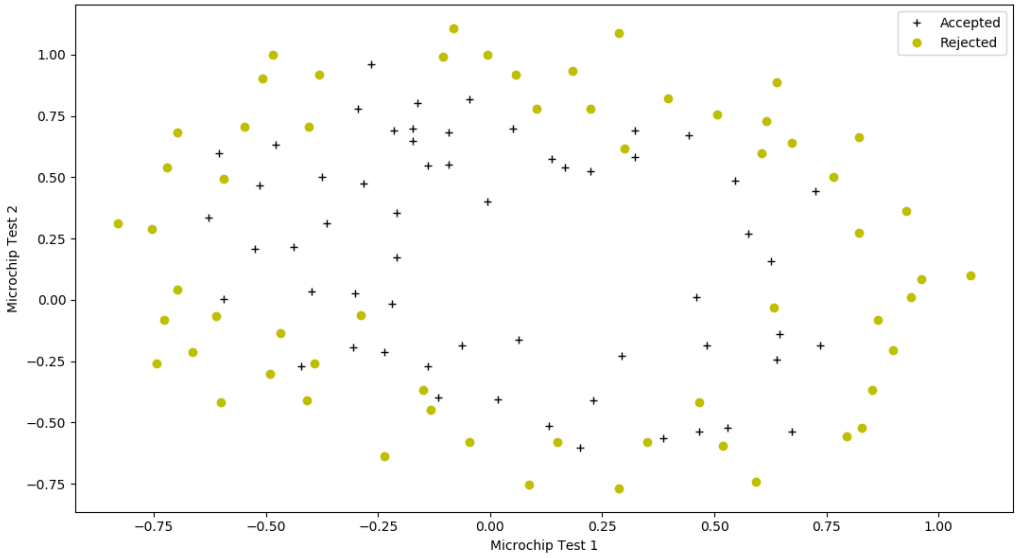


FIGURE 3.2.1 – A regularized logistic regression model that predicts whether microchips from a fabrication plant passes quality assurance

This time, we find that we cannot use a straight-line to separate the dataset into positive and negative examples.

1. One way to fit the data better is to create more features from each data point. Since we have two features x_{i1} and x_{i2}, we can generate polynomially (for exemple Degree 6) :

$$\begin{bmatrix} x_{i1} \\ x_{i2} \\ {x_{i1}}^2 \\ x_{i1}x_{i2} \\ {x_{i2}}^2 \\ {x_{i1}}^3 \\ \vdots \\ x_{i1}{x_{i2}}^5 \\ {x_{i2}}^6 \end{bmatrix}$$

   Finally we have $\sum_{j=2}^{7} j = 27$ features in total :

Listing 9 – Map feature

```
X1 = data(:, 1);
X2 = data(:, 2);

degree = 6; % polynomial features mapping
X_train_map_feature = [];

```

```
for i = 1:degree
    for j = 0:i
        X_train_map_feature = [X_train_map_feature, (X1.^(i-j)) .* (X2.^j)];
    end
end
```

2. The L2 regularized log loss function (1.6.1) can be written as following code :

$$l_2(\Theta) = -\sum_{i=1}^{n}\left[y_i \log s({\mathbf{X}_i}^T\mathbf{\Theta}) + (1-y_i)\log(1-s({\mathbf{X}_i}^T\mathbf{\Theta}))\right] + \frac{1}{2\sigma^2}\sum_{j=0}^{m}|\theta_j|^2$$

Listing 10 – L2 regularized Log loss function

```
loss_function_L2 = @(X, Y, Theta, sigma) ...
    -sum(arrayfun(@(i) ...
    Y(i) * log(s(X(i, :) * Theta)) + ...
    (1 - Y(i)) * log(1 - s(X(i, :) * Theta)), 1:size(X, 1))) + ...
    (1 / (2 * sigma^2)) * sum(Theta .^ 2);

% Theta = [0.5 ; rand(m, 1) - 0.5];
% sigma = sqrt(2); % Controls amount of regularization
% loss_function_L2(X_train, Y_train, Theta, sigma)
```

3. The gradient for L2 regularized log loss function (1.6.2) can be written as following code :

$$\frac{\partial}{\partial \mathbf{\Theta}_j} l_2(\Theta) = \sum_{i=1}^{n}\left[\left(s({\mathbf{X}_i}^T\mathbf{\Theta}) - y_i\right){\mathbf{X}_i}^T{}_j\right] + \frac{1}{\sigma^2}\sum_{j=0}^{m}\theta_j$$

Listing 11 – Gradient for L2 regularized log loss function

```
gradient_loss_function_L2 = @(X, y, Theta, sigma) ...
    sum(cell2mat(arrayfun(@(i) (s(X(i, :) * Theta) - y(i)) * X(i, :)', 1:size(X, 1), '
        UniformOutput', false)), 2) + (1 / sigma^2) * sum(Theta);

% cell2mat() : convert the cell array to matrix
% sum( , 2) : the summed gradients for each feature

gradient_loss_function_L2(X_train, Y_train, Theta, sigma)
```

4. Apply the gradient descent method using the above function to find the optimal parameters of a standard logistic regression model :

$$\mathbf{\Theta}^{k+1} = \mathbf{\Theta}^k - \alpha \frac{\partial}{\partial \mathbf{\Theta}_j} l_2(\Theta^k)$$

Listing 12 – Gradient descent method

```
% Some gradient descent settings & stop test value
lambda = 0.01;
sigma = sqrt(1/lambda);

k = 1;
k_max = 10000; % iterations

alpha = 0.001; % learning rate

Theta = [1 ; rand(m, 1) - 0.5]; % We take any \Theta^0
l_history = zeros(k_max, 1);
Theta_history = zeros(k_max, length(Theta));

epsilon_l(k) = 1; % for stop test value
epsilon_l_k_min = 1e-5; % relative variation of the criterion


while k < k_max && abs(epsilon_l(k)) > epsilon_l_k_min

    k = k + 1;
```

```
21
22      % Calculate the gradient
23      dl_dTheta = gradient_loss_function_L2(X_train, Y_train, Theta, sigma);
24
25      % Update parameters using alpha and gradient
26      Theta = Theta - alpha * dl_dTheta;
27
28      % Save cost J at each iteration
29      l_history(k) = loss_function_L2(X_train, Y_train, Theta, sigma);
30
31      % Save Theta at each iteration
32      Theta_history(k, :) = Theta';
33
34      % Print cost every 10% of the iterations
35      if mod(k, ceil(k_max / 10)) == 0 || k == k_max
36          fprintf('Iteration %4d/%4d: Cost %8.2f\n', k, k_max, l_history(k));
37      end
38
39      % Relative variation of the criterion
40      epsilon_l(k) = (l_history(k-1) - l_history(k)) / l_history(k-1);
41
42  end
```

5. Finally we have `Train Accuracy: 82.203390%` which matches the answers to the exercises.
6. We export the data using

   `writematrix(Theta_opt,'Theta_opt_L2.txt')`

   Then we used a function written in Python to plot the decision boundary.

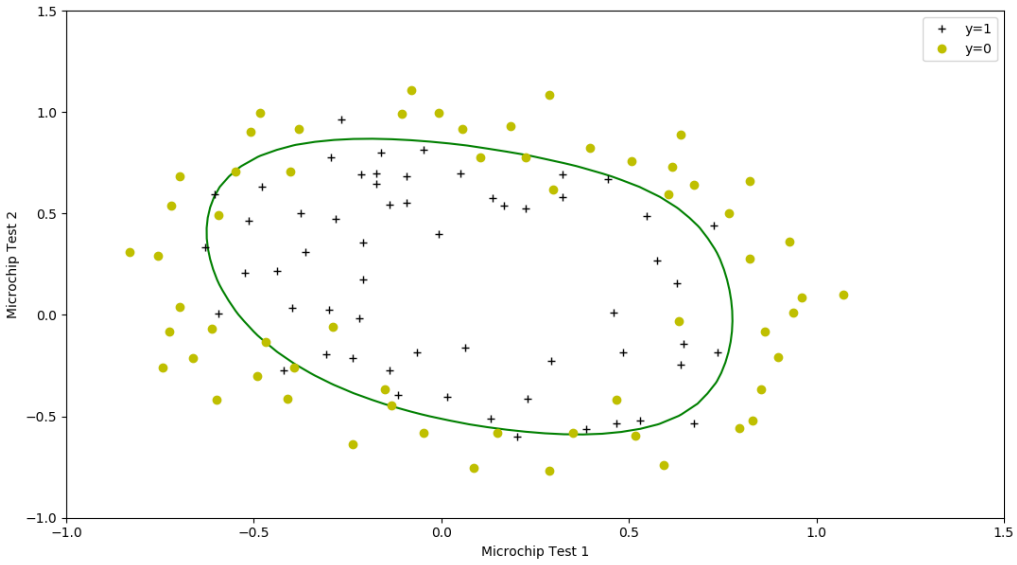


FIGURE 3.2.2 – The decision boundary

3.3 Simulated Data : standard logistic regression v.s. robust logistic regression

We try to realise what the author did [2] in section **4.1 Simulated Data** :

1. Defining parameters with $n = 500$ samples and $m = 10$ features :

   `lambda = 0.01; kappa = 0.02; num_samples = 500; num_features = 10;`

2. Generate a randomized eigenmatrix (with eigenvalues uniformly distributed between [-5, 5]) :

   `train_data = -5 + (5 - (-5)) * randn(num_samples, num_features);`

3. Generate a weight vector `theta`, set the weights of all features to 2 :

```
theta = 2 * ones(num_features+1, 1);
```

4. The intercept be zero

```
gamma = zeros(num_samples, 1);
```

5. Compute the matrix $\tilde{\mathbf{X}}$:

```
robust_train_data_local = [ones(num_samples,1),train_data, (kappa/lambda) * eye(num_samples)];
```

Compute the matrix $\tilde{\mathbf{\Theta}}$:

```
robust_theta = [theta; (kappa/lambda) * gamma];
```

6. Calculating probabilities using sigmoid functions :

```
robust_probabilities = 1 ./ (1 + exp(-robust_train_data_local * robust_theta - gamma));
```

7. Generate binary labels using a threshold of 0.5

```
robust_train_labels = robust_probabilities > 0.5;
```

Remarque: Predictions

In our test, predictions are made as usual :

$$\mathbf{I}\{s(\mathbf{X\Theta}) > 0.5\}$$

which means

$$\begin{cases} 1 & \text{if } s(\mathbf{X\Theta}) > 0.5 \\ 0 & \text{if } s(\mathbf{X\Theta}) \leq 0.5 \end{cases}$$

8. Introducing label noise (e.g., randomly flipping 30% of labels of 0 to be 1) :

```
flip_probability = 0.3; robust_zero_label_indices = find(robust_train_labels == 0);

num_to_flip = round(flip_probability * length(robust_zero_label_indices));

robust_flip_indices = zero_label_indices(randperm(length(robust_zero_label_indices), num_to_flip));

robust_train_labels(robust_flip_indices) = 1;
```

9. Concerning the loss function :

$$l(\tilde{\Theta}) = -\sum_{i=1}^{n} \Big[y_i \log s(\tilde{\mathbf{X}}\tilde{\Theta}) + (1 - y_i) \log(1 - s(\tilde{\mathbf{X}}\tilde{\Theta})) \Big] + \kappa \sum_{j=0}^{m+n} \tilde{\mathbf{F}}_{\mathbf{j}} |\tilde{\mathbf{\Theta}}_j|$$

Remarque: Lasso Regularization of Generalized Linear Models

According to Matlab, for a nonnegative value of λ, `lassoglm` sovles the problem :

$$\min_{\beta_0, \beta} \left(\frac{1}{N} \text{Deviance}(\beta_0, \beta) + \lambda \sum_{j=1}^{p} |\beta_j| \right)$$

— The function Deviance in this equation is the deviance of the model fit to the responses using the intercept β_0 and the predictor coefficients β. The formula for Deviance depends on the `distr` parameter we supply to `lassoglm`. Minimizing the λ-penalized deviance is equivalent to maximizing the λ-penalized loglikelihood.
— N is the number of observations.
— λ is a nonnegative regularization parameter corresponding to one value of Lambda.
— The parameters β_0 and β are a scalar and a vector of length p, respectively.

So the code snippet given by [2] page 23 :

```
robust.fit = glmnet(robust.train.data.local, as.factor(train.labels), lambda=kappa,
                    family='binomial', standardize=FALSE)
```

In matlab it turns to

```
[RblogitCoef, FitInfo_R] = lassoglm(robust_train_data_local, robust_train_labels, 'binomial',
                            'Lambda', kappa, 'Standardize', false);
```

where `B` is our $\tilde{\mathbf{\Theta}}$.

10. Finally, we compute accuracy on our training set just as we did before :

```
robust_Y_prediction = RblogitFit >= 0.5;

rb_accuracy = mean(robust_Y_prediction == robust_train_labels) * 100;
```

11. We run this for 100 times for both the robust logistic regression and standard logistic regression (this code is similar to the robust more details in **Listing 17** (Simulated Data - Simulated_Data.m). Then we store each result in vectors `Train_Accuracy_Standard` and `Train_Accuracy_Robust`. We calculate the Mean and Standard Deviation just using the function `mean()` and `std()` .

Output in (Listing 13)

Listing 13 – Output - Simulated Data - Simulated_Data.m

```
>> Simulated_Data
Train Accuracy Standard: Mean = 80.466000%, Std Dev = 2.004249%
Train Accuracy Robust: Mean = 81.196000%, Std Dev = 1.183721%
```

Code in (Listing 17)

We can see that in the case (regularized) where we give the probability of class 0 flipping to 1 with a probability of 30%, the robust logistic regression model performs better than the standard logistic regression model since the mean is higher and the standard deviation is smaller.

4 Some notes about regularization (L1-Regularization and L2-Regularization)

Considering our objectif :

$$l(\Theta)$$

with L1-Regularization :

$$l(\Theta) + \kappa \sum_{j=0}^{m} |\theta_j|$$

just take an example in 2 dimension, $|\theta_1| + |\theta_2|$ represents a rhombus, we need to find the rhombus that is tangent to a certain contour line. In most cases (in higner dimension), we are more likely to have in axis just like Figure 4.0.1 thus we will find lots of 0 in the vector :

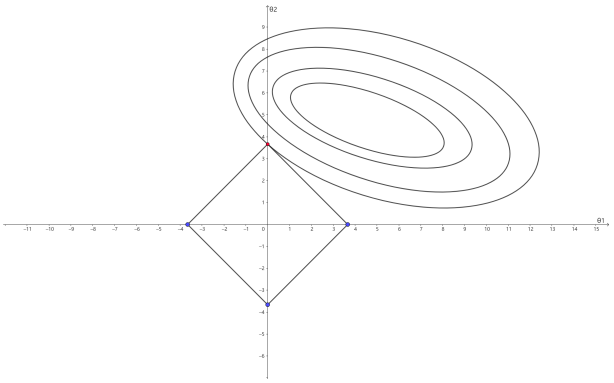


FIGURE 4.0.1 – L_1-Regularization

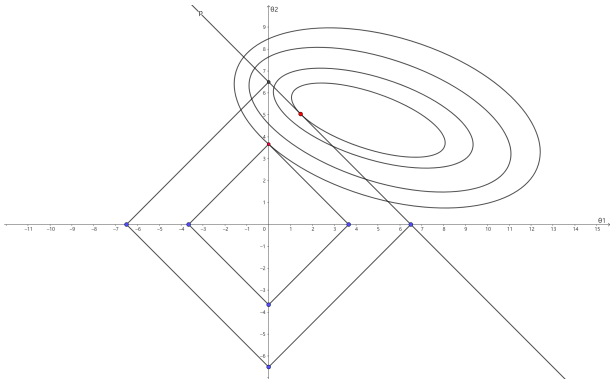


FIGURE 4.0.2 – L_1-Regularization

with L2-Regularization :

$$l(\Theta) + \frac{1}{2\sigma^2} \sum_{j=0}^{m} |\theta_j|^2$$

just take an example in 2 dimension, $|\theta_1|^2 + |\theta_2|^2$ represents a cercle, we need to find the cercle that is tangent to a certain contour line, at this time, the tangent points do not easily appear on the axes :

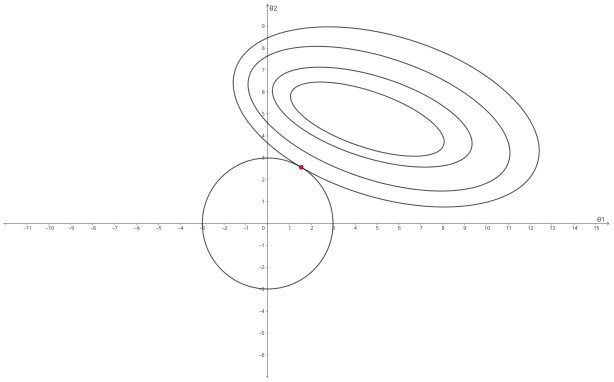


FIGURE 4.0.3 – L_2-Regularization

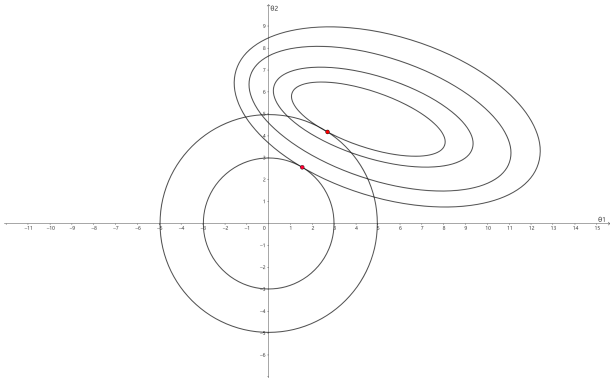


FIGURE 4.0.4 – L_2-Regularization

Besides let's take the L_2-regularization as we did in **section 3.2** for example, we choose 3 different $\lambda = \frac{1}{2\sigma^2}$

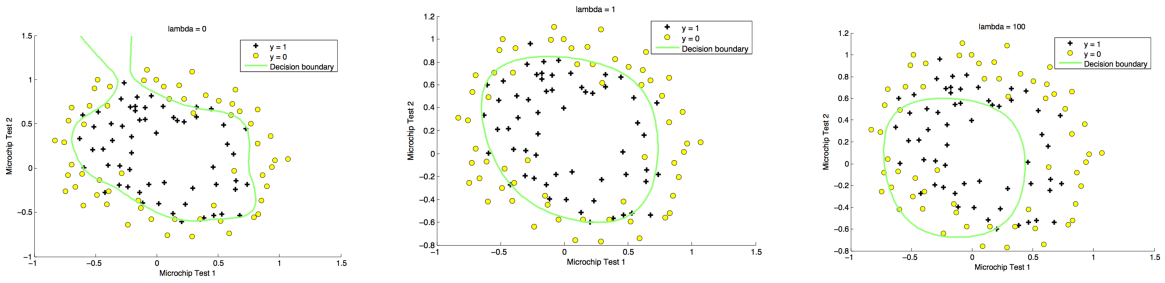


FIGURE 4.0.5 – Overfitting | fitting | Underfitting [1]

We could see that if the σ that we choose is too large, the training model will be overfitted ; while if we the σ that we choose is too small, the training model will be underfitted.

5 Conclusion

During this project, we have

— Studied the standard logistic regression without and with L2 regularisation. We calculated the gradient of the log loss function for each model. Then we do the programming in Matlab : we write the log loss function, its gradient and the gradient descent method by ourselves . We store the coefficient in texts and use a function written in python to visualize the decision boundary.

— Applied the standard logistic regression model and the robust logistic regression to a Uniform(-5, 5) simulated data, the robust model performs better than the standard one both in terms of mean and in terms of standard deviation.

Références

[1] Ritvik. C1 - supervised machine learning - regression and classification, 2022.

[2] J. Tibshirani and C. D. Manning. Robust logistic regression using shift parameters (long version). *arXiv preprint arXiv :1305.4987*, 2013.

A Table of Programmes

Listings

B Programmes

Simulated Data - standard_logistic_regression.m

Listing 14 – Simulated Data - standard_logistic_regression.m

```
clear all; close all;

%% 1. Sigmoid function
s = @(t) 1 ./ (1 + exp(-t));
% figure(1); x = linspace(-5,+5,100); y = s(x); plot(x,y)

%% 2. Read data
data = readmatrix('ex2data1.txt');
X0_train = data(:, 1:2);
Y_train = data(:, 3);

[n, m] = size(X0_train); % n samples; m features
In1 = ones(n,1);
X_train = [In1, X0_train]; % add one column of 1 for the bais \theta_0

% figure(2);
% admitted_idx = Y_train == 1;
% not_admitted_idx = Y_train == 0;
% scatter(X0_train(admitted_idx, 1), X0_train(admitted_idx, 2), 'r', 'filled');
% hold on;
% scatter(X0_train(not_admitted_idx, 1), X0_train(not_admitted_idx, 2), 'b', 'filled');
% legend('Admitted', 'Not Admitted');
% xlabel('Exam 1 score');
% ylabel('Exam 2 score');
% title('A classification model that estimates an applicant's probability of admission based on the scores from those two exams');
% hold off;

%% 3. Loss function
loss_function = @(X, Y, Theta, lambda_) ...
    sum(arrayfun(@(i) ...
    -Y(i) * log(s((X(i, :) * Theta))) - ...
    (1 - Y(i)) * log(1 - s((X(i, :) * Theta))), 1:size(X, 1)));

Theta = zeros(m+1,1);
loss_function(X_train, Y_train, Theta)

%% 4. The gradient for Loss function
gradient_loss_function = @(X, y, Theta, lambda_) ...
    sum(cell2mat(arrayfun(@(i) (s(X(i, :) * Theta) - y(i)) * X(i, :)', 1:size(X, 1), 'UniformOutput', false)), 2);

% cell2mat() : convert the cell array to matrix
% sum( , 2) : the summed gradients for each feature

gradient_loss_function(X_train, Y_train, Theta)

%% 5. Gradient descent
% Some gradient descent settings & stop test value

k = 1;
k_max = 20000; % iterations

alpha = 0.00001; % learning rate

Theta = [-8; 0.01 * (rand(2, 1) - 0.5)]; % We take any \Theta^0
l_history = zeros(k_max, 1);
Theta_history = zeros(k_max, length(Theta));

epsilon_l(k) = 1; % for stop test value
epsilon_l_k_min = 1e-8; % relative variation of the criterion

```

```
61
62  while k < k_max && abs(epsilon_l(k)) > epsilon_l_k_min
63
64      k = k + 1;
65
66      % Calculate the gradient
67      dl_dTheta = gradient_loss_function(X_train, Y_train, Theta);
68
69      % Update parameters using alpha and gradient
70      Theta = Theta - alpha * dl_dTheta;
71
72      % Save cost J at each iteration
73      l_history(k) = loss_function(X_train, Y_train, Theta);
74
75      % Save Theta at each iteration
76      Theta_history(k, :) = Theta';
77
78      % Print cost every 10% of the iterations
79      if mod(k, ceil(k_max / 10)) == 0 || k == k_max
80          fprintf('Iteration %4d/%4d: Cost %8.2f\n', k, k_max, l_history(k));
81      end
82
83      % Relative variation of the criterion
84      epsilon_l(k) = (l_history(k-1) - l_history(k)) / l_history(k-1);
85
86  end
87
88
89  %% 6. Evaluating standard logistic regression model
90  Theta_opt = Theta_history(k,:)';
91  Y_prediction_0to1 = s(X_train*Theta_opt); % use sigmoid fuinction
92  Y_prediction_0or1(Y_prediction_0to1 < 0.5) = 0; % threshold = 0.5
93  Y_prediction_0or1(Y_prediction_0to1 >= 0.5) = 1;
94  Y_prediction_0or1 = Y_prediction_0or1';
95
96  % Compute accuracy on our training set
97  accuracy = mean(Y_prediction_0or1 == Y_train) * 100;
98  fprintf('Train Accuracy: %f%%\n', accuracy);
99
100 writematrix(Theta_opt,'Theta_opt.txt')
```

Simulated Data - regularized_logistic_regression.m

Listing 15 – Simulated Data - regularized_logistic_regression.m

```
1  clear all; close all;
2
3  %% 1. Sigmoid function
4  s = @(t) 1 ./ (1 + exp(-t));
5  % figure(1); x = linspace(-5,+5,100); y = s(x); plot(x,y)
6
7  %% 2. Read data
8  data = readmatrix('ex2data2.txt');
9  X0_train = data(:, 1:2);
10 Y_train = data(:, 3);
11
12 [n, m] = size(X0_train); % n samples; m features
13 In1 = ones(n,1);
14 X_train = [In1, X0_train]; % add one column of 1 for the bais \theta_0
15
16 figure(2);
17 admitted_idx = Y_train == 1;
18 not_admitted_idx = Y_train == 0;
19 scatter(X0_train(admitted_idx, 1), X0_train(admitted_idx, 2), 'r', 'filled');
20 hold on;
21 scatter(X0_train(not_admitted_idx, 1), X0_train(not_admitted_idx, 2), 'b', 'filled');
22 legend('Accepted', 'Rejected');
```

```
xlabel('Microchip Test 1');
ylabel('Microchip Test 2');
title('A regularized logistic regression model that predicts whether microchips from a fabrication plant passes quality assurance ');
hold off;

%% 3. Map feature
X1 = data(:, 1);
X2 = data(:, 2);

degree = 6; % polynomial features mapping
X_train_map_feature = [];

for i = 1:degree
    for j = 0:i
        X_train_map_feature = [X_train_map_feature, (X1.^(i-j)) .* (X2.^j)];
    end
end

[n, m] = size(X_train_map_feature); % n samples; m features
X_train = [In1, X_train_map_feature]; % add one column of 1 for the bais \theta_0

%% 4. Loss function L2
loss_function_L2 = @(X, Y, Theta, sigma) ...
    -sum(arrayfun(@(i) ...
    Y(i) * log(s(X(i, :) * Theta)) + ...
    (1 - Y(i)) * log(1 - s(X(i, :) * Theta)), 1:size(X, 1))) + ...
    (1 / (2 * sigma^2)) * sum(Theta .^ 2);

Theta = [0.5 ; rand(m, 1) - 0.5];
sigma = sqrt(2); % Controls amount of regularization
loss_function_L2(X_train, Y_train, Theta, sigma)

%% 5. The gradient for Loss function L2
gradient_loss_function_L2 = @(X, y, Theta, sigma) ...
    sum(cell2mat(arrayfun(@(i) (s(X(i, :) * Theta) - y(i)) * X(i, :)', 1:size(X, 1), '
        UniformOutput', false)), 2) + (1 / sigma^2) * sum(Theta);

% cell2mat() : convert the cell array to matrix
% sum( , 2) : the summed gradients for each feature

gradient_loss_function_L2(X_train, Y_train, Theta, sigma)

%% 6. Gradient descent
% Some gradient descent settings & stop test value
% lambda = 0.01;
lambda = 0.01;
sigma = sqrt(1/lambda);

k = 1;
k_max = 10000; % iterations

alpha = 0.001; % learning rate

Theta = [1 ; rand(m, 1) - 0.5]; % We take any \Theta^0
l_history = zeros(k_max, 1);
Theta_history = zeros(k_max, length(Theta));

epsilon_l(k) = 1; % for stop test value
epsilon_l_k_min = 1e-8; % relative variation of the criterion


while k < k_max && abs(epsilon_l(k)) > epsilon_l_k_min

    k = k + 1;

    % Calculate the gradient
    dl_dTheta = gradient_loss_function_L2(X_train, Y_train, Theta, sigma);

```

```
    % Update parameters using alpha and gradient
    Theta = Theta - alpha * dl_dTheta;

    % Save cost J at each iteration
    l_history(k) = loss_function_L2(X_train, Y_train, Theta, sigma);

    % Save Theta at each iteration
    Theta_history(k, :) = Theta';

    % Print cost every 10% of the iterations
    if mod(k, ceil(k_max / 10)) == 0 || k == k_max
        fprintf('Iteration %4d/%4d: Cost %8.2f\n', k, k_max, l_history(k));
    end

    % Relative variation of the criterion
    epsilon_l(k) = (l_history(k-1) - l_history(k)) / l_history(k-1);

end

%% 6. Evaluating standard logistic regression model
Theta_opt = Theta_history(k,:)';
Y_prediction_0to1 = s(X_train*Theta_opt); % use sigmoid fuinction
Y_prediction_0or1(Y_prediction_0to1 < 0.5) = 0; % threshold = 0.5
Y_prediction_0or1(Y_prediction_0to1 >= 0.5) = 1;
Y_prediction_0or1 = Y_prediction_0or1';

% Compute accuracy on our training set
accuracy = mean(Y_prediction_0or1 == Y_train) * 100;
fprintf('Train Accuracy: %f%%\n', accuracy);


writematrix(Theta_opt,'Theta_opt_L2.txt')
```

Simulated Data - plot_decision_boundary.py

Listing 16 – Simulated Data - plot_decision_boundary.py

```
def plot_decision_boundary(w, b, X, y):
    # Credit to dibgerge on Github for this plotting code

    plot_data(X[:, 0:2], y)

    if X.shape[1] <= 2:
        plot_x = np.array([min(X[:, 0]), max(X[:, 0])])
        plot_y = (-1. / w[1]) * (w[0] * plot_x + b)

        plt.plot(plot_x, plot_y, c="b")

    else:
        u = np.linspace(-1, 1.5, 50)
        v = np.linspace(-1, 1.5, 50)

        z = np.zeros((len(u), len(v)))

        # Evaluate z = theta*x over the grid
        for i in range(len(u)):
            for j in range(len(v)):
                z[i,j] = sig(np.dot(map_feature(u[i], v[j]), w) + b)

        # important to transpose z before calling contour
        z = z.T

        # Plot z = 0
        plt.contour(u,v,z, levels = [0.5], colors="g")
```

Simulated Data - Simulated_Data.m

Listing 17 – Simulated Data - Simulated_Data.m

```
clear all;

% Number of iterations (times)
times = 100;

% Initialize vectors to store accuracy results
Train_Accuracy_Standard = zeros(1, times);
Train_Accuracy_Robust = zeros(1, times);

for t = 1:times

% 1. Data dimension
num_samples = 500;
num_features = 10;

% 2. Generate a randomized eigenmatrix (with eigenvalues uniformly distributed between (-5, 5))
train_data = -5 + (5 - (-5)) * rand(num_samples, num_features); % 500 samples, 10 features

% 3. Generate a weight vector theta, set the weights of all features to 2
theta = 2 * ones(num_features+1, 1); % binary label

%% standard logistic regression
% 4. Calculating probabilities using sigmoid functions
I = ones(num_samples,1);
train_data_local = [I,train_data];
%linear_combination = train_data_local * theta;
%probabilities = 1 ./ (1 + exp(-linear_combination));
probabilities = 1 ./ (1 + exp(-train_data_local * theta));

% 5. Generate binary labels using a threshold of 0.5
train_labels = probabilities > 0.5;

% 6. Introducing label noise (e.g., randomly flipping 30% of labels of 0 to 1)
flip_probability = 0.3;
% noise_indices = rand(num_samples, 1) < flip_probability;
% train_labels(noise_indices) = ~train_labels(noise_indices);
zero_label_indices = find(train_labels == 0);
num_to_flip = round(flip_probability * length(zero_label_indices));
flip_indices = zero_label_indices(randperm(length(zero_label_indices), num_to_flip));
train_labels(flip_indices) = 1;

% 7. Use glmfit for the standard logistic regression
% with distri='binomal',link='logit'
[StdlogitCoef, FitInfo_S] = glmfit(train_data, train_labels, 'binomial','logit');
%disp('standard logistic regression: ');
%disp(StdlogitCoef);

% 8. Evaluating standard logistic regression model
StdlogitFit = (1 ./ (1 + exp(-train_data_local * StdlogitCoef)));
%StdlogitFit2 = glmval(StdlogitCoef,train_data,'logit');
% Y_prediction(StdlogitFit<0.5) = 0;
% Y_prediction(StdlogitFit>=0.5) = 1;
% Y_prediction = Y_prediction';

Y_prediction = StdlogitFit >= 0.5;
Y_prediction = Y_prediction(:); % Ensure Y_prediction is a column vector

% Compute accuracy on our training set
std_accuracy = mean(Y_prediction == train_labels) * 100;
%fprintf('Train Accuracy (standard) : %f%%\n', std_accuracy);

% figure(1);
% plot(train_data, train_labels, '.');
```

```
% hold on;
% plot(train_data, StdlogitFit, '.');
% hold on;
% plot(linspace(-5,5,100), (1 ./ (1 + exp(-linspace(-5,5,100)))), '--')
% legend('orignal data', 'standard logistic fit', 'sigmoid');
% title('standard logistic regression');

%% robust logistic regression
% 9. Defining parameters
lambda = 0.01; % ex.
kappa = 0.02; % ex.

% 10. The intercept be zero
gamma = zeros(num_samples, 1);

% 11. The matrix \mathbf{\Tilde{X}}
I = ones(num_samples,1);
identity_matrix = (kappa/lambda) * eye(num_samples);
robust_train_data_local = [I,train_data, identity_matrix];

robust_theta = [theta; (kappa/lambda) * gamma];

% 12. Calculating probabilities using sigmoid functions
%linear_combination = train_data_local * theta + gamma;
%probabilities = 1 ./ (1 + exp(-linear_combination));
robust_probabilities = 1 ./ (1 + exp(-robust_train_data_local * robust_theta - gamma));

% 13. Generate binary labels using a threshold of 0.5
robust_train_labels = robust_probabilities > 0.5;

% 14. Introducing label noise (e.g., randomly flipping 30% of labels of 0 to 1)
flip_probability = 0.3;
% noise_indices = rand(num_samples, 1) < flip_probability;
% robust_train_labels(noise_indices) = ~robust_train_labels(noise_indices);

robust_zero_label_indices = find(robust_train_labels == 0);
num_to_flip = round(flip_probability * length(robust_zero_label_indices));
robust_flip_indices = zero_label_indices(randperm(length(robust_zero_label_indices),
    num_to_flip));
robust_train_labels(robust_flip_indices) = 1;

% 15. Lasso Regularization of Generalized Linear Models (use lassoglm)
[RblogitCoef, FitInfo_R] = lassoglm(robust_train_data_local, robust_train_labels, 'binomial', '
    Lambda', kappa, 'Standardize', false);

% print lassoglm model info
%disp(FitInfo_R);

% print \Theta
%disp('robust logistic regression: ');
%disp(RblogitCoef(1:num_features+1));

RblogitFit = (1 ./ (1 + exp(-robust_train_data_local * RblogitCoef)));

% 16. Evaluating robust logistic regression model

% robust_Y_prediction(RblogitFit<0.5) = 0;
% robust_Y_prediction(RblogitFit>=0.5) = 1;
% robust_Y_prediction = robust_Y_prediction';

robust_Y_prediction = RblogitFit >= 0.5;
robust_Y_prediction = robust_Y_prediction(:); % Ensure robust_Y_prediction is a column vector

% Compute accuracy on our training set
rb_accuracy = mean(robust_Y_prediction == robust_train_labels) * 100;
%fprintf('Train Accuracy (robust) : %f%%\n', rb_accuracy);


% figure(2);
```

```
% plot(train_data, train_labels, '.');
% hold on;
% plot(train_data, StdlogitFit, '.');
% hold on;
% plot(train_data, robust_train_labels, 'o');
% hold on;
% plot(train_data, RblogitFit, 'o');
% hold on;
% plot(linspace(-5,5,100), (1 ./ (1 + exp(-linspace(-5,5,100)))), '--')
% legend('standard orignal data', 'standard logistic fit', 'robust orignal data ', 'robust logistic fit', 'sigmoid');
% title('standard and robust logistic regression');

% Store accuracies in vectors
Train_Accuracy_Standard(t) = std_accuracy;
Train_Accuracy_Robust(t) = rb_accuracy;
end

fprintf('Train Accuracy Standard: Mean = %f%%, Std Dev = %f%%\n', mean(Train_Accuracy_Standard), std(Train_Accuracy_Standard));

fprintf('Train Accuracy Robust: Mean = %f%%, Std Dev = %f%%\n', mean(Train_Accuracy_Robust), std(Train_Accuracy_Robust));
```