Rice University

COMP540 Assignment 2

Yawen Guo and Negar Erfanian
(yg57 & ne12)

February 7, 2020

On my honor, I have neither given nor received any unauthorized help on this assignment.

1 Gradient and Hessian of J(θ) for logistic regression

1.1

We know that $g(z) = \frac{1}{1+e^{-z}}$

$$\begin{aligned}\frac{\partial g(z)}{\partial z} &= \frac{e^{-z}}{(1+e^{-z})^2}\\ &= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}}\\ &= \frac{1}{1+e^{-z}} \cdot (1 - \frac{1}{1+e^{-z}})\\ &= g(z)(1-g(z))\end{aligned}$$

1.2

Using result in 1.1 and knowing that $h^(_\theta x)$ can be defined as:

$$\begin{aligned}h_\theta(x) = g(\theta^\top x) &= \frac{1}{1+e^{-\theta^\top}x}\\ g(z)' &= g(z) * (1-g(z))\end{aligned}$$

We can simply write the first part in $J(\theta)$ in summation as: $ylog(g) + (1-y)log(1-g)$
The derivation of it:

$$\begin{aligned} y\frac{1}{g}g' + (1-y)(\frac{1}{1-g})(-g') &= \frac{y(1-g)-g(1-y)}{g(1-g)}g' \\ &= \frac{y-y*g-g+g*y}{g(1-g)} * g(1-g)*x \\ &= (y-g)x \end{aligned}$$

The derivation of the second part in $J(\theta)$ is simple
Therefore,

$$\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)} + \frac{\lambda}{m}[0, \theta_1, ..., \theta_d]^\top$$

1.3

$$\begin{gathered} \frac{\partial}{\partial\theta}J(\theta) = \frac{1}{m}(X^\top(g(X\theta)-y)) + \frac{\lambda\theta}{m} = \\ \frac{1}{m}(X^\top(\frac{1}{1+exp(-X\theta)} - y)) + \frac{\lambda\theta}{m} \end{gathered} \tag{1}$$

where θ is a $d \times 1$ vector.

1.4

Having the vector form of $\frac{\partial}{\partial\theta}J(\theta)$ from the previous part, we can write

$$\frac{\partial^2}{\partial\theta^2}J(\theta) = \frac{1}{m}(X^\top g(X\theta)(1-g(X\theta))^\top X) + \frac{\lambda}{m} \tag{2}$$

where $g(X\theta)(1-g(X\theta))^\top$ is a $m \times m$ diagonal matrix with $g(x^{(i)}\theta)(1-g(x^{(i)}\theta)),\ i \in \{1, 2, ..., m\}$, are the scalar components on the diagonal.
Knowing that $H = \frac{1}{m}(X^T SX) + \lambda I$

$$\begin{aligned} 0 < h_\theta(x^{(i)}) < 1 \\ Thus, 0 < 1 - h_\theta(x^{(i)}) < 1 \end{aligned}$$

Since X is full rank matrix.
For any non-zero vector v_{m*1}:

$$v^T Hv = \frac{1}{m}\sum_{i}^{m} S^{(i)}(v^{(i)}\sum_{k}^{m} X_{ki}X_{kj})^2 + \lambda I > 0$$

Therefore, H is positive definite.

1.5

According to the newton's method we have:

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$$

If we span this equation in the matrix form for optimizing the features coefficients θ, we can write:

$$\theta_{\text{new}} = \theta_{\text{old}} - H^{-1}g$$

Where H and g are the $d \times d$ hessian matrix and the $d \times 1$ gradient descent vector that we formed in the previous sections. The code for this part is bellow:

```
import numpy as np

def sigmoid (z):

    sig = np.zeros(z.shape)
    sig = 1/(1+np.exp(-z))

    return sig


def loss(X, y, theta, reg):

    m,dim = X.shape
    J = 0
    sig = sigmoid(np.dot(X,theta))
    J = 1/m* (-np.dot(np.transpose(y),np.log(sig)) -np.dot(1-np.transpose(y), np.log(1- sig

    return J



def grad_loss(X,y, theta, reg):

    m,dim = X.shape
    grad = np.zeros((dim,))
    grad = np.dot(np.transpose(X),sigmoid(np.dot(X,theta))-y)/m +reg/m*(np.vstack([0, theta

    return grad

def Hessian(X, theta, reg):

    S = []
    m,dim = X.shape
    for i in range(m):
        S.append(list(sigmoid(np.dot(X[i,:],theta))*(1-sigmoid(np.dot(X[i,:],theta)))))
    Sdiag = np.diagflat(S)
    H = 1/m*(np.dot(np.transpose(X),np.dot(Sdiag,X)) + reg*np.eye(dim) )
    return H
```

```
def newtons_method(X, y, initial, reg):

    theta = initial

    delta_l = 10000
    L = loss(X, y, theta, reg)
    # Convergence Conditions
    sigma = .0000000001
    max_iterations = 10
    i = 0
    while i < max_iterations and abs(delta_l)>sigma:
        i += 1
        grad = grad_loss(X, y, theta, reg)
        H = Hessian(X, theta, reg)
        H_inv = np.linalg.inv(H)

        theta_new = theta - np.dot(H_inv,grad)
        l_new = loss(X,y, theta_new, reg)
        delta_l = L - l_new
        L = l_new
        theta = theta_new
        print('theta is :' ,theta)
        #print('Loss is:', np.abs(delta_l))
    return theta


X  = np.array([[1, 0, 3],[1, 1 ,3],[1 ,0, 1], [1, 1 ,1]])
initial = np.array([[0],[-2],[1]])
y = np.array([[1],[1],[0],[0]])
reg = 0.07
theta = newtons_method(X, y, initial, reg)
print(theta)
```

And θ after 10 iterations are:

```
θ is : [[-3.15199171] [-0.40585887] [ 1.81504991]]
θ is : [[-4.26505811] [-0.29747087] [ 2.33806757]]
θ is : [[-4.90809867] [-0.18354141] [ 2.5979506 ]]
θ is : [[-5.28312052] [-0.10788523] [ 2.7307514 ]]
θ is : [[-5.50526622] [-0.0643088 ] [ 2.806734 ]]
θ is : [[-5.63818741] [-0.03873452] [ 2.85196033]]
θ is : [[-5.71828372] [-0.02345439] [ 2.87915056]]
θ is : [[-5.76678512] [-0.01424811] [ 2.89558908]]
θ is : [[-5.79624891] [-0.00867314] [ 2.90556462]]
θ is : [[-5.81418413e+00] [-5.28625031e-03] [ 2.91163282e+00]]
```

2 Overfitting and unregularized logistic regression

When seperating classes in a linearly separable dataset using logistic datasets, we realize that we call a data sample with label 1 when the sigmoid function for that data sample is more than 0.5 and we call a data sample with label 0 when the sigmoid function for that data sample is less than 0.5. Therefore, the parameters that make the sigmoid function equal to 0.5 are chosen as the parameters to define the decision boundary. In logistic regression this happens when we have $\theta^\top x = \theta_0 + \theta_1 x_1 + ... = 0$. If we increase the amplitude of θ we realize that the slope of change of sigmoid function gets sharper. Therefore, as we increase the amplitude to infinity, the sigmoid function becomes like a step function that we can easily use for linearly separable datasets. To avoid this singular solution, we put a regularization term to not let θ get so big. Therefore this regularization term lessens the affect of very large θ by bringing them close to zero so that overfitting does not happen.

3 Implementing a k-nearest-neighbor classifier

The code and the accuracy results for this part are attached in knn folder.

3.1 Classifying test data with a kNN-classifier

Got 137 / 500 correct, accuracy: 0.274000
Got 145 / 500 correct, accuracy: 0.290000

3.2 Speeding up distance computations

Two loop version took 21.413049 seconds
One loop version took 29.188346 seconds
No loop version took 0.210047 seconds
For this problem, One loop method has the same time complexity as two loop method, so one loop may take a little longer time than two loop.

3.3 Choosing k by cross validation

As shown in figure 1, we realize that the best k can be between 5 and 8. Choosing $k = 8$ number of neighbors, we gained 29.4% accuracy on the test set.
k = 1, accuracy = 0.263000 k = 1, accuracy = 0.257000 k = 1, accuracy = 0.264000 k = 1, accuracy = 0.278000 k = 1, accuracy = 0.266000
k = 3, accuracy = 0.257000 k = 3, accuracy = 0.263000 k = 3, accuracy = 0.273000 k = 3, accuracy = 0.282000 k = 3, accuracy = 0.270000
k = 5, accuracy = 0.265000 k = 5, accuracy = 0.275000 k = 5, accuracy = 0.295000 k = 5, accuracy = 0.298000 k = 5, accuracy = 0.284000
k = 8, accuracy = 0.272000 k = 8, accuracy = 0.295000 k = 8, accuracy = 0.284000 k = 8, accuracy = 0.298000 k = 8, accuracy = 0.290000
k = 10, accuracy = 0.272000 k = 10, accuracy = 0.303000 k = 10, accuracy = 0.289000 k = 10, accuracy = 0.292000 k = 10, accuracy = 0.285000
k = 12, accuracy = 0.271000 k = 12, accuracy = 0.305000 k = 12, accuracy = 0.285000 k = 12, accuracy = 0.289000 k = 12, accuracy = 0.281000
k = 15, accuracy = 0.260000 k = 15, accuracy = 0.302000 k = 15, accuracy = 0.292000 k = 15, accuracy = 0.292000 k = 15, accuracy = 0.285000
k = 20, accuracy = 0.268000 k = 20, accuracy = 0.293000 k = 20, accuracy = 0.291000 k = 20, accuracy = 0.287000 k = 20, accuracy = 0.286000
k = 50, accuracy = 0.273000 k = 50, accuracy = 0.291000 k = 50, accuracy = 0.274000 k = 50, accuracy = 0.267000 k = 50, accuracy = 0.273000
k = 100, accuracy = 0.261000 k = 100, accuracy = 0.272000 k = 100, accuracy = 0.267000 k = 100, accuracy = 0.260000 k = 100, accuracy = 0.267000

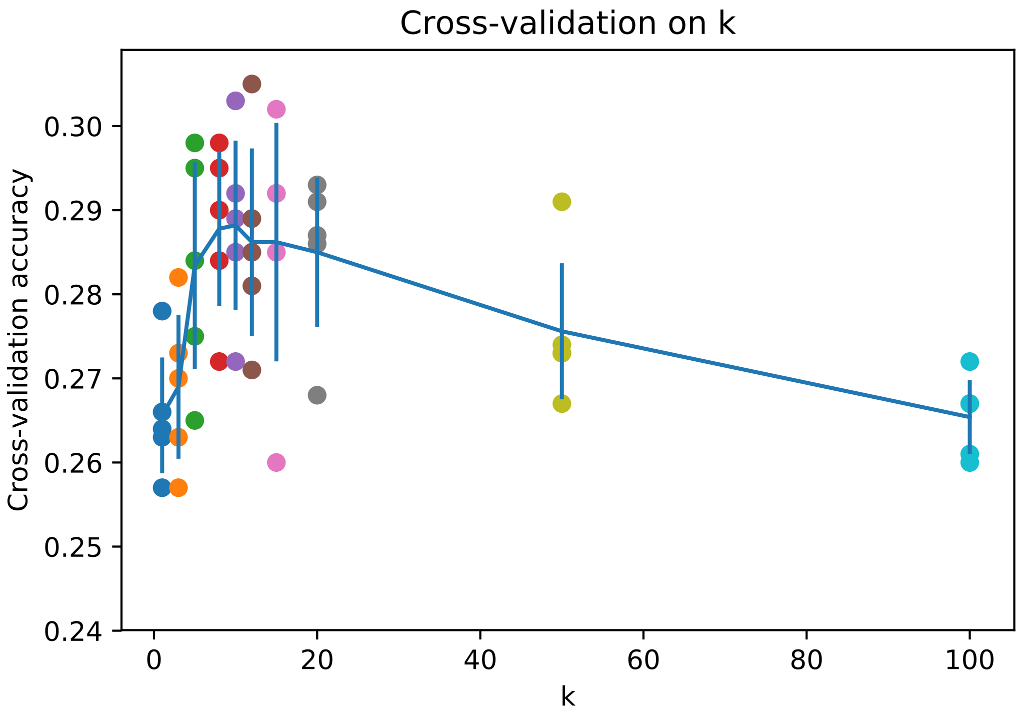

Figure 1: Choosing k by crossvalidation on the CIFAR-10 dataset

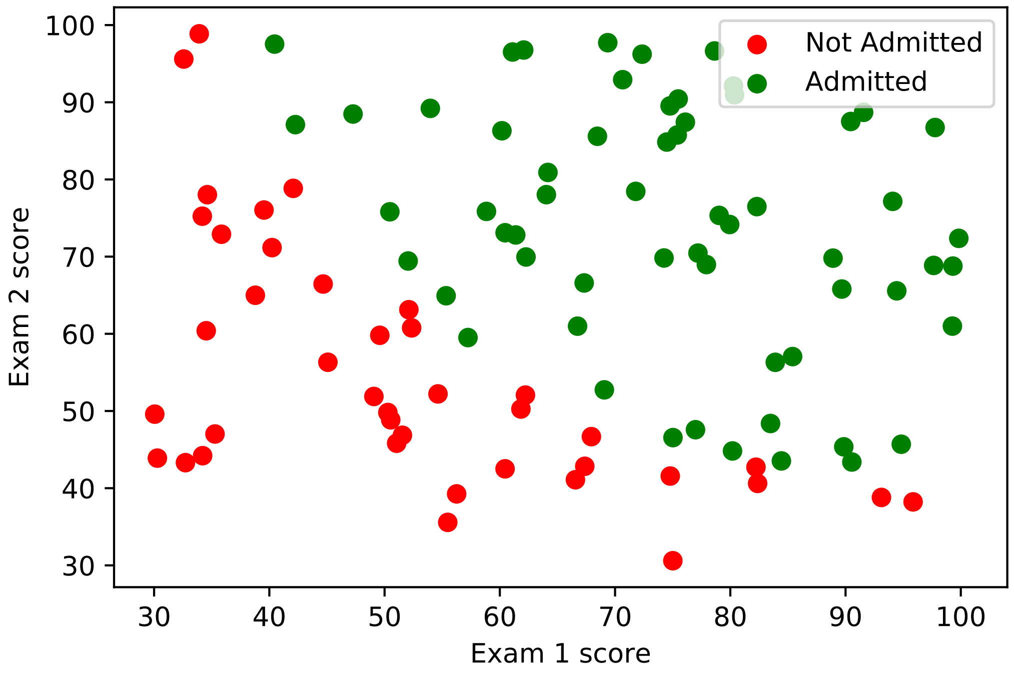

Figure 2: The Training data

4 Implementing logistic regression

Part A: Logistic regression

Visualizing the data

Problem 4A1 Implementing logistic regression: the sigmoid function

Problem 4A2: Cost function and gradient of logistic regression

```
Loss on all-zeros theta vector (should be around 0.693) = 0.6931
Gradient of loss wrt all-zeros theta vector (should be around [-0.1, -12.01, -11.26]) = [ -0.1 -12.00921659
-11.26284221]
Optimization terminated successfully.
Current function value: 0.203498
Iterations: 19
Function evaluations: 20
Gradient evaluations: 20
Theta found by fmin bfgs: [-25.16056945 0.20622963 0.20146073]
Final loss = 0.2035
```

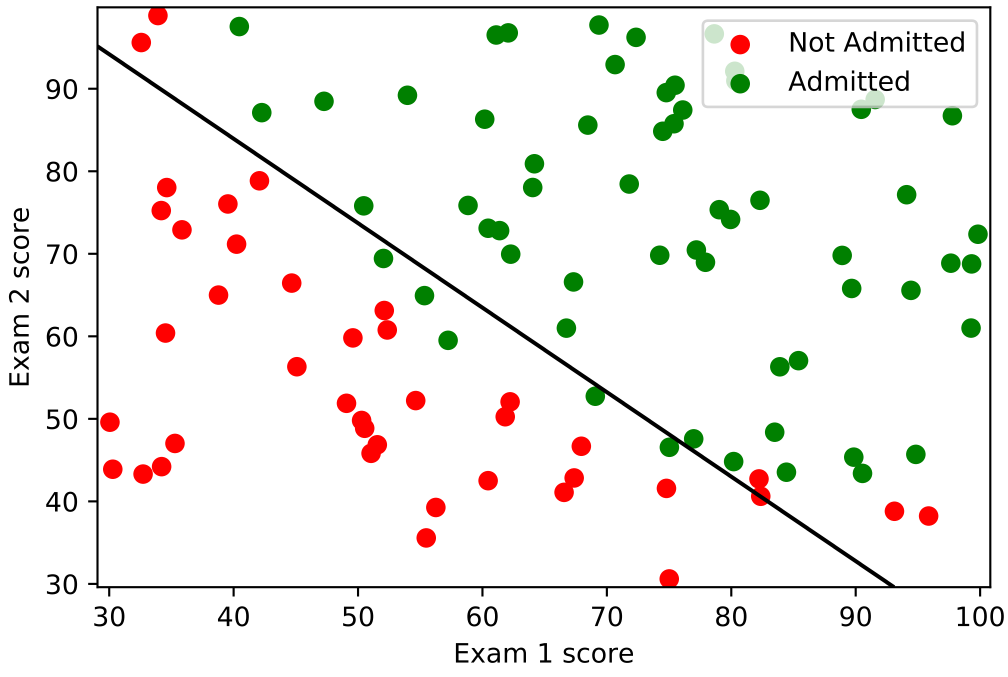

Figure 3: The decision boundary

Learning parameters using fminbfgs

Loss on all-zeros theta vector (should be around 0.693) = 0.6931
Gradient of loss wrt all-zeros theta vector (should be around [-0.1, -12.01, -11.26]) = [-0.1 -12.00921659 -11.26284221]
Optimization terminated successfully.
Current function value: 0.203498
Iterations: 19
Function evaluations: 20
Gradient evaluations: 20
Theta found by fmin bfgs: [-25.16056945 0.20622963 0.20146073] Final loss = 0.2035

Problem 4A3: Prediction using a logistic regression model

For a student with 45 on exam 1 and 85 on exam 2, the probability of admission = 0.7762
Accuracy on the training set = 0.8900

Theta found by sklearn: [[-25.15293066 0.20616459 0.20140349]

Part B: Regularized logistic regression

Visualizing the data
Plotting data with green circle indicating (y=1) examples and red circle indicating (y=0) examples ...

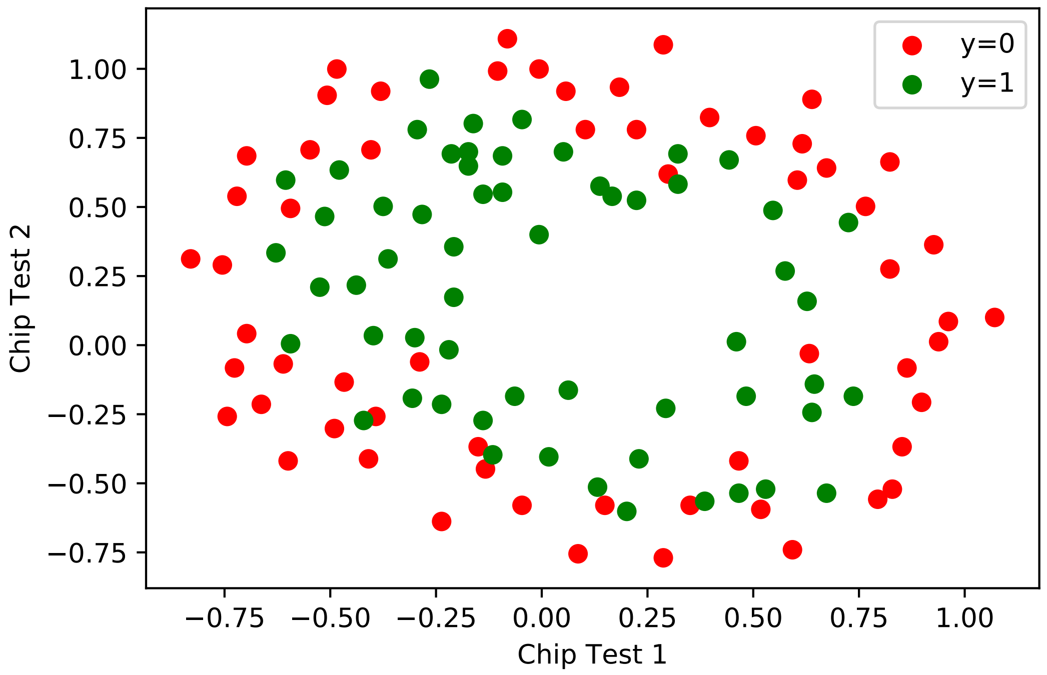

Figure 4: Plot of trianing data

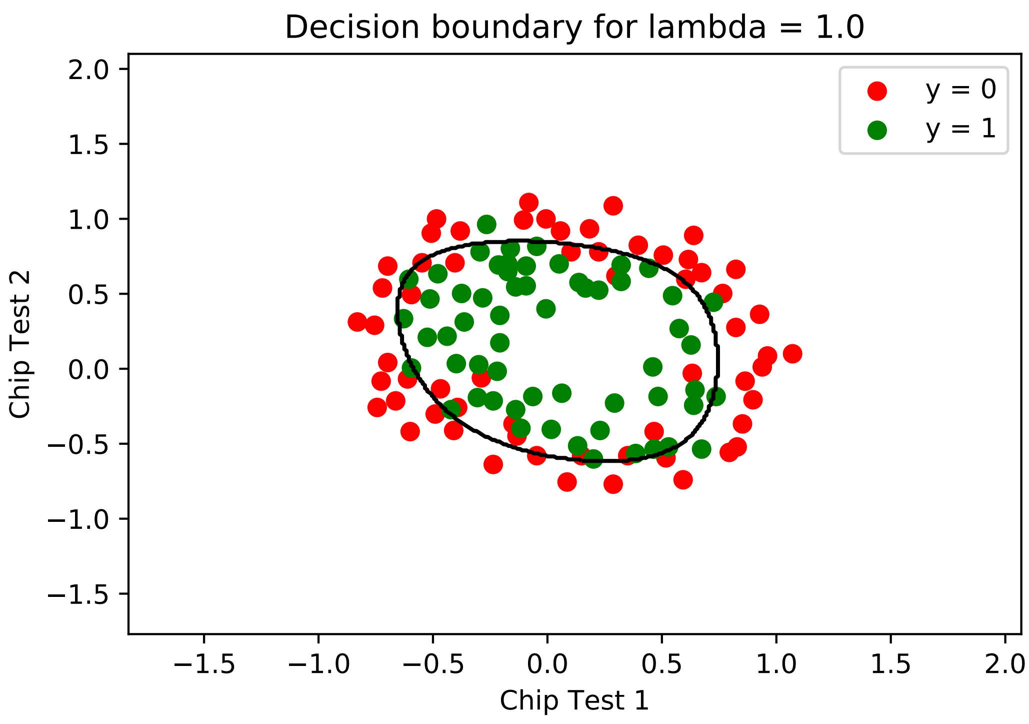

Figure 5: Training data with decision boundary for lambda = 1

Problem 4B1: Cost function and gradient for regularized logistic regression

Optimization terminated successfully.
Current function value: 0.529003
Iterations: 47
Function evaluations: 48
Gradient evaluations: 48
Theta found by fminbfgs: [1.27268739 0.62557016 1.1809665 -2.01919822 -0.91761468 -1.43194199 0.12375921 -0.36513086 -0.35703388 -0.17485805 -1.45843772 -0.05129676 -0.61603963 -0.2746414 -1.19282569 -0.24270336 -0.20570022 -0.04499768 -0.27782709 -0.29525851 -0.45613294 -1.04377851 0.02762813 -0.29265642 0.01543393 -0.32759318 -0.14389199 -0.92460119]
Final loss = 0.4625

Plotting the decision boundary

Problem 4B2: Prediction using the model

Accuracy on the training set = 0.8305

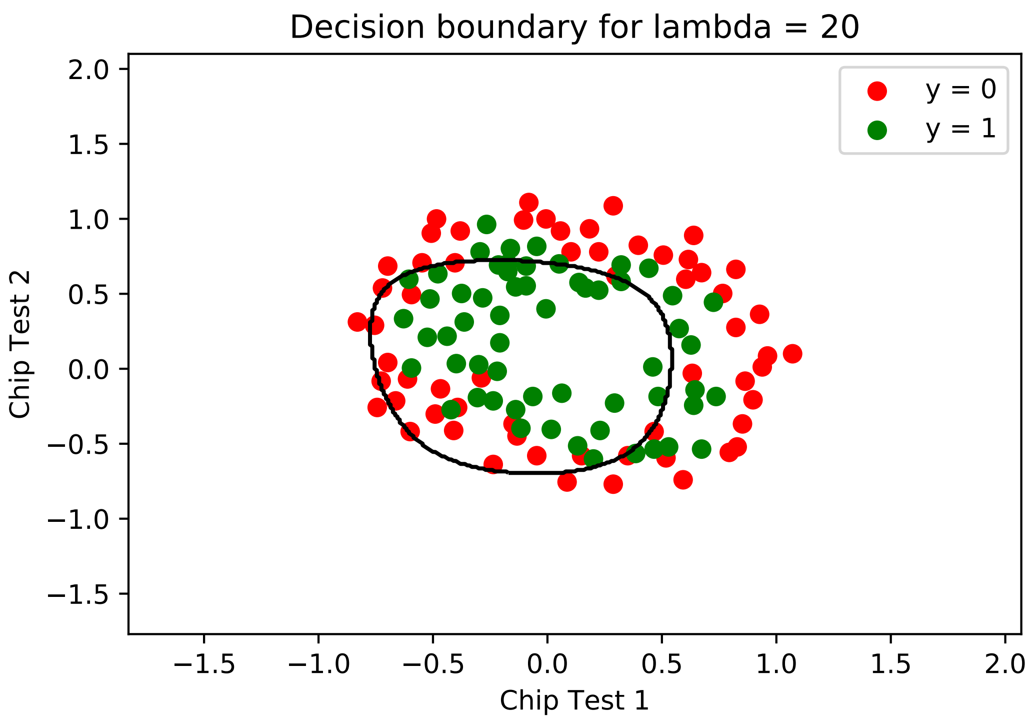

Figure 6: Training data with decision boundary for lambda = 20 (underfitting)

Problem 4B3: Varying λ

As shown in figure 6 and 7, by choosing $\lambda = 20$ and $\lambda = 0.0001$,we face underfitting and overfitting, respectively.

Problem 4B4: Exploring L1 and L2 penalized logistic regression

L2:Theta found by sklearn with L2 reg: [[1.1421394 0.60141117 1.16712554 -1.87160974 -0.91574144 -1.26966693 0.12658629 -0.3686536 -0.34511687 -0.17368655 -1.42387465 -0.04870064 -0.60646669 -0.26935562 -1.16303832 -0.24327026 -0.20702143 -0.04326335 -0.28028058 -0.286921 -0.46908732 -1.03633961 0.02914775 -0.29263743 0.01728096 -0.32898422 -0.13801971 -0.93196832]]
Loss with sklearn theta: 0.4684
see figure 8
Theta found by sklearn with L1 reg: [[1.87011148 0.68676271 1.28052351 -4.86295933 -1.62199516 -2.34501334 0. 0. 0. 0. 0. 0. 0. 0. -2.36476395 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Loss with sklearn theta: 0.4381

see figure 9

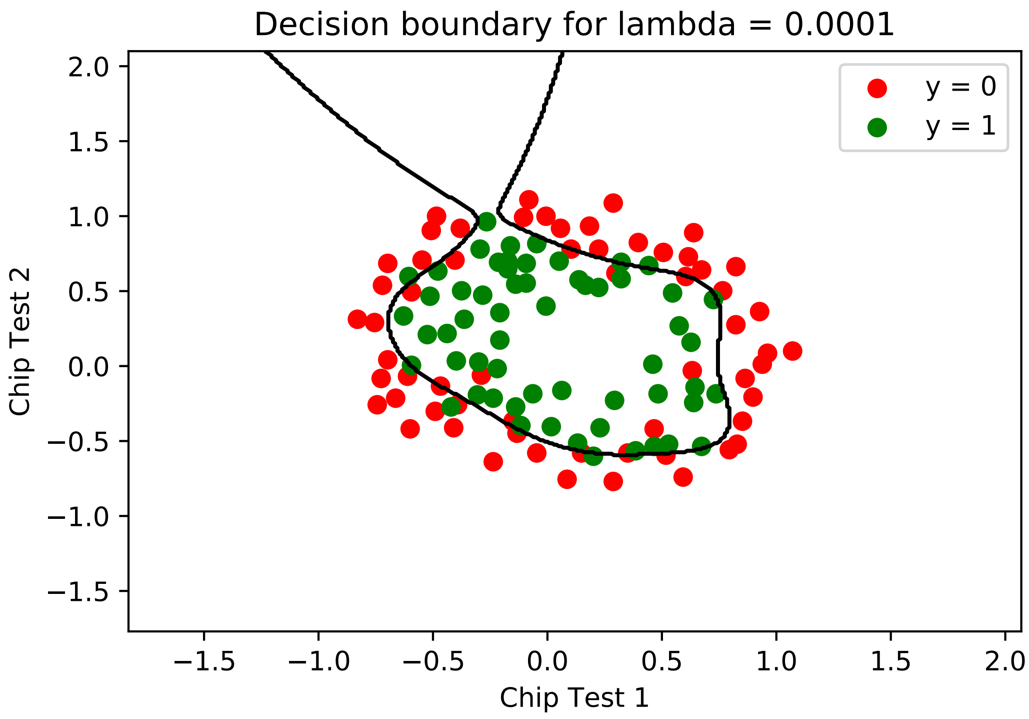

Figure 7: Training data with decision boundary for lambda = 0.0001 (overfitting)

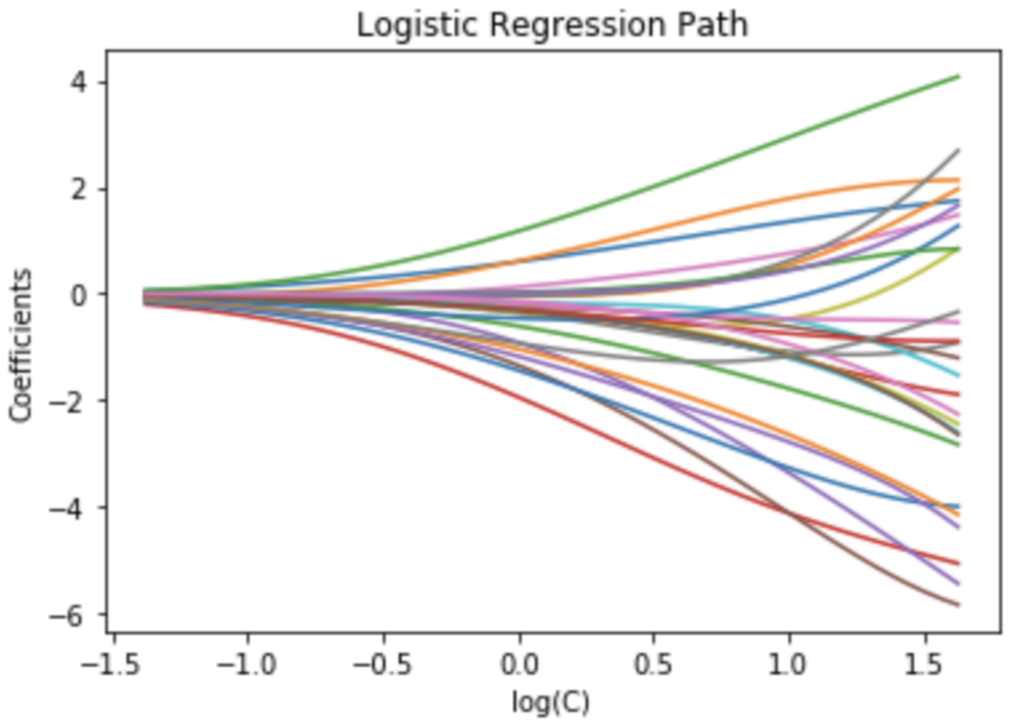

Figure 8: L2 regression path

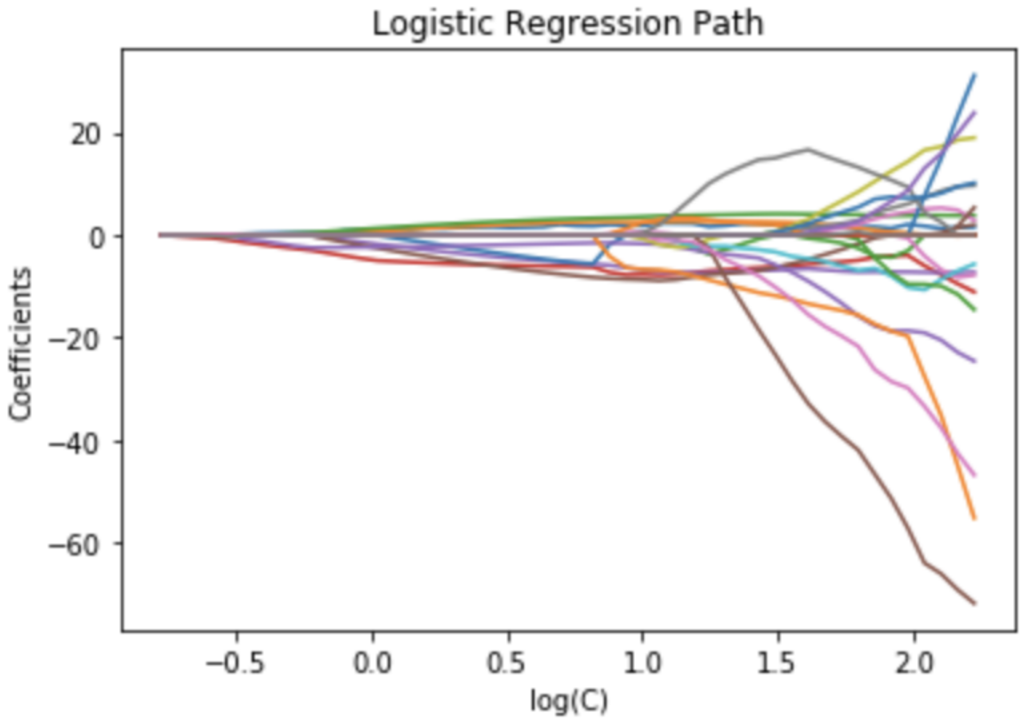

Figure 9: L1 regression path

Problem 4 Part C: Logistic regression for spam classification

4.0.1 Feature transformation

4.0.2 Fitting regularized logistic regression models (L2 and L1)

Overall both models are giving high accuracies on the train and test sets as indicated in tables below

L2 Feature transform	Train	Test	λ
std	0.9305	0.9290	0.1
logt	0.9465	0.9434	0.6
bin	0.9367	0.9277	1.1

L1 Feature transform	Train	Test	λ
std	0.9223	0.9219	4.6
logt	0.9481	0.9440	1.6
bin	0.9370	0.9258	3.6

But, if we look at the coefficients after training the data using the best lambda, we realize that most coefficients are very close to zero (in L2 model) or zero (in L1 model). Therefore, due to the built-in feature selection of L1 model, as now we are facing sparcity in the coefficients we can conclude that using L1 regularization is more efficient.