Pattern Separation Using Logistic Regression

Amulya Tallamraju AI20BTECH11003 | Anita Dash MA20BTECH11001 | Anjali MA20BTECH11002 | Ruthwika Boyapally MA20BTECH11004 | Tapishi Kaur MA20BTECH11017

***Abstract*—A pattern separation problem is a problem of obtaining a criterion for distinguishing between the elements of two disjoint sets of patterns. We represent the patterns as coordinates (points in the Euclidean Space).One way through which the points can be separated is by constructing a line(2-D) or a hyperplane(n-D) or a nonlinear surface such that one set of points lie on one side of the surface, while the rest lie on the other side. We use logistic modeling to find such a separation and show how it is a convex optimization problem. We also use CVXPY to solve the problem.**

***Index Terms*—logistic regression,convexity, log likelihood**

I. Introduction

Logistic Regression is used to separate patterns and is based on the idea of maximum likelihood estimation. The classification algorithm Logistic Regression is used to predict the likelihood of a categorical dependent variable.

Logistic regression is a simple and more efficient method for binary and linear classification problems. It achieves very good performance with linearly separable classes.

Why the name Logistic Regression? In linear regression, our main aim is to predict the output variable Y which is continuous. In logistic regression, we do the same with one small addition. We pass the result through a function known as the Sigmoid Function to predict the output Y. Sigmoid funtion is defined as

$$\frac{1}{1+e^{-z}} \tag{1}$$

Sigmoid function bounds the input between 0 and 1.

Logistic regression uses the same underlying concept as linear regression with a few differences. Linear regression gives a continuous value of output y with respect to input X . In logistic regression we convert the numerical value obtained into a categorical variable based on a threshold value. Hence logistic regression has "Regression" in its name.

Similarities and Differences between Linear Regression and Logistic Regression

- Both are supervised machine learning algorithms. A Supervised machine learning algorithm takes a known set of input data and known responses data to model reasonable predictions for the responses to the new input data. We use linear mathematical equations for both regressions for reasonable predictions.
- But usage of Linear Regression and Logistic Regression algorithms is completely different.The primary difference between linear regression and logistic regression is that logistic regression's range is bounded between 0 and 1. In addition, as opposed to linear regression, logistic regression does not require a linear relationship between inputs and output variables.Linear Regression is used to handle regression problems whereas Logistic regression is used to handle the classification problems.Linear regression's goal is to determine the best-fitting line, but logistic regression goes one step farther and fits the line values to the sigmoid curve.In linear regression, the mean squared error is used to calculate the loss function, whereas in logistic regression, maximum likelihood estimation is used.

There are three main types of logistic regression:-

- **Binary Logistic Regression**: Binary logistic regression is used to classify objects into two groups, they either satisfy a particular condition or don't. There are just two possible outcome answers which are typically represented as a 0 or a 1.
- **Multinomial logistic regression**: Multinomial logistic regression is a model where there are multiple classes that an item can be classified as. There is a set of three or more predefined classes set up prior to running the model.
- **Ordinal logistic regression**: Ordinal logistic regression is a model where there are multiple classes that an item can be classified as; however, in this case an ordering of classes is required. Classes do not need to be proportionate. The distance between each class can vary.

Since we want to find a hyperplane (n-Dimensional) or line (2-D) that best separates the points in space, we will be going forward with **Binary Logistic Regression**

II. Logistic Regression Problem

We consider a random variable y ∈ {0, 1}

$$Pr(y=1)=p, \quad Pr(y=0)=1-p, \quad p\in[0,1] \tag{2}$$

We consider the data: x_i, i = 1,...,m. $x_i \in R^n$ are explanatory variables, and their corresponding outcomes y_i, i = 1,...,m. $y_i \in [0,1]$ are their associated Boolean class

We construct a linear classifier

$$\hat{\boldsymbol{y}} = \mathbf{1}[\beta^T x] \quad therefore, \tag{3}$$

$$\hat{\boldsymbol{y}} = \begin{cases} 1 & \text{if } \beta^T x > 0 \\ 0 & \text{if } otherwise \end{cases} \tag{4}$$

Identify applicable funding agency here. If none, delete this.

If the probability of $\hat{\boldsymbol{y}}$ for an explanatory variable x being 1 is higher than it being 0, it implies that $\beta^T x > 0$, else if probability of x being 0 is higher then $\beta^T x < 0$ therefore we model the posterior probabilities of the classes given the data linearly, with

$$\log \frac{Pr(Y=1|X=x)}{Pr(Y=0|X=x)} = \beta^T x \tag{5}$$

which implies:

$$\log \frac{p}{1-p} = \beta^T x \tag{6}$$

$$\implies \frac{p}{1-p} = e^{\beta^T x} \tag{7}$$

$$\implies p = \frac{e^{\beta^T x}}{1+e^{\beta^T x}} \tag{8}$$

$$1-p = \frac{1}{1+e^{\beta^T x}} \tag{9}$$

We need to find the maximum likelihood of the model parameter $\beta \in R^n$. Finding this ML is sometimes called logistic regression.
We can re-order the data so for $x_1, ..., x_q$, the outcome is y = 1, and for $x_{q+1}, ..., x_m$ the outcome is y = 0. The likelihood function then has the form:

$$\prod_{i=1}^{q} p_i \prod_{i=q+1}^{m} (1-p_i) \tag{10}$$

The log-likelihood function has the form:

$$l(\beta) = \sum_{i=1}^{q} \log(p_i) + \sum_{i=q+1}^{m} \log(1-p_i) \tag{11}$$

substituting the value of p and $(1-p)$ from (9) and (10) respectively:

$$l(\beta) = \sum_{i=1}^{q} \log \frac{e^{\beta^T x_i}}{1+e^{\beta^T x_i}} + \sum_{i=q+1}^{m} \log \frac{1}{1+e^{\beta^T x_i}} \tag{12}$$

$$\implies l(\beta) = \sum_{i=1}^{q} \log e^{\beta^T x_i} + \sum_{i=1}^{m} \log \frac{1}{1+e^{\beta^T x_i}} \tag{13}$$

$$\implies l(\beta) = \sum_{i=1}^{q} \beta^T x_i - \sum_{i=1}^{m} \log(1+e^{\beta^T x_i}) \tag{14}$$

Checking the convexity of $log(1+e^{\beta^T x_i})$**,** $\beta \in R^n \quad x_i \in R^n$

$$f(\beta) = log(1+e^{\beta^T x_i}) \tag{15}$$

$$f(\beta) = log(1+e^{\sum_{i=1}^{n} \beta_i x_i}) \tag{16}$$

Hessian matrix H of f is:

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial \beta_1^2} & \frac{\partial^2 f}{\partial \beta_1 \partial \beta_2} & . & . & . & \frac{\partial^2 f}{\partial \beta_1 \partial \beta_n} \\ \frac{\partial^2 f}{\partial \beta_2 \partial \beta_1} & \frac{\partial f^2}{\partial \beta_2^2} & . & . & . & \frac{\partial^2 f}{\partial \beta_2 \partial \beta_n} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ \frac{\partial^2 f}{\partial \beta_n \partial \beta_1} & \frac{\partial^2 f}{\partial \beta_n \partial \beta_2} & . & . & . & \frac{\partial^2 f}{\partial \beta_n^2} \end{bmatrix} \tag{17}$$

$$Therefore, (H_f)_{i,j} = \frac{\partial^2 f}{\partial \beta_i \partial \beta_j} \quad i,j = 1, ..., n \tag{18}$$

$$\frac{\partial f}{\partial \beta_i} = \frac{x_i e^{\beta^T x}}{1+e^{\beta^T x}} \tag{19}$$

$$\frac{\partial^2 f}{\partial \beta_j \partial \beta_i} = \frac{x_i x_j e^{\beta^T x}(1+e^{\beta^T x}) - (x_i e^{\beta^T x})(x_j e^{\beta^T x})}{(1+e^{\beta^T x})^2} \tag{20}$$

$$\frac{\partial^2 f}{\partial \beta_j \partial \beta_i} = \frac{x_i x_j e^{\beta^T x}}{(1+e^{\beta^T x})^2} \tag{21}$$

Hence, substituting (21) in (18)

$$(H_f)_{i,j} = \frac{x_i x_j e^{\beta^T x}}{(1+e^{\beta^T x})^2} \tag{22}$$

$$let \quad k_\beta = \frac{e^{\beta^T x}}{(1+e^{\beta^T x})^2} \quad (note: k_\beta > 0) \tag{23}$$

$$(H_f)_{i,j} = x_i x_j k_\beta \tag{24}$$

Substituting (24) in H_f

$$H_f = \begin{bmatrix} x_1^2 k_\beta & x_1 x_2 k_\beta & . & . & . & x_1 x_n k_\beta \\ x_2 x_1 k_\beta & x_2^2 k_\beta & . & . & . & x_2 x_n k_\beta \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_n x_1 k_\beta & x_n x_2 k_\beta & . & . & . & x_n^2 k_\beta \end{bmatrix} \tag{25}$$

Taking k_β outside the matrix

$$H_f = k_\beta \begin{bmatrix} x_1^2 & x_1 x_2 & . & . & . & x_1 x_n \\ x_2 x_1 & x_2^2 & . & . & . & x_2 x_n \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_n x_1 & x_n x_2 & . & . & . & x_n^2 \end{bmatrix} \tag{26}$$

$$|H_f| = k_\beta^n \begin{vmatrix} x_1^2 & x_1 x_2 & . & . & . & x_1 x_n \\ x_2 x_1 & x_2^2 & . & . & . & x_2 x_n \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_n x_1 & x_n x_2 & . & . & . & x_n^2 \end{vmatrix} \tag{27}$$

By the properties of determinants, we do the following operations:-
Taking x_i outside the determinant from row R_i
And then taking x_i outside the determinant from column C_i

$$|H_f| = k_\beta^n (x_1 x_2 ... x_n)^2 \begin{vmatrix} 1 & 1 & 1 & . & . & 1 \\ 1 & 1 & 1 & . & . & 1 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 1 & 1 & . & . & . & 1 \end{vmatrix} \tag{28}$$

case-1: if n = 1

$$|H_f|_{1\times 1} = k_\beta(x_1)^2 \tag{29}$$

case-2: if $n > 1$

$$|H_f|_{n\times n} = 0 \tag{30}$$

for $n > 1$ all the rows in matrix $|H_f|_{n*n}$ are identical, Hence by the properties of determinants, the determinant of the matrix is 0

A. Sylvester's Criterion

A matrix is positive semi definite if and only if all its principal minors ≥ 0.
Let S=$\{1, ..., n\}$. If A is an $n \times n$ matrix, I is a subset of S with k elements, and J is a subset of S with k elements, then we write $[A]_{I,J}$ for the $k \times k$ minor of A that corresponds to the rows with index in I and the columns with index in J. If I = J, then $[A]_{I,J}$ is called a principal minor. Variable x_i is only present in the i^{th} row and the i^{th} column of the matrix. If some index i is present in I as well as J, then the resulting principal minor will not have that particular variable in it. So the principal minor will look like

$$|P_{I,J}| = k_\beta^{n-k}(\prod x_i)^2 \begin{vmatrix} 1 & 1 & 1 & . & . & 1 \\ 1 & 1 & 1 & . & . & 1 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 1 & 1 & . & . & . & 1 \end{vmatrix}_{(n-k)\times(n-k)} \tag{31}$$

where $i \in S - I$ Thus we can see that when $n - k > 1$, the determinant of the principal minor turns out to be zero and when $n - k = 1$ then the principal minor is ≥ 0. Using Sylvester's criterion we conclude that the Hessian is positive semi definite which implies that the function is Convex.
Therefore, $f(\beta) = log(1 + e^{\beta^T x_i})$ is a convex function, which implies $-f(\beta) = -log(1+e^{\beta^T x_i})$ is concave. As sum of concave functions is concave, $(-\sum_{i=1}^{m} \log(1+e^{\beta^T x_i}))$ is concave
coming back to (14), $\sum_{i=1}^{q} \beta^T x_i$ is affine and hence is concave. Since $l(\beta)$ is a sum of two concave functions, $l(\beta)$ **is a concave function.**
Thus, as l is concave function of parameter β, the logistic regression problem can be solved as a convex optimization problem (as $(-l(\beta))$ will be a convex function).

III. Derivation of Geometric Interpretation of the Algorithm

Suppose we need to find a plane 'P' which separates the two classes of our dataset. General equation of a plane is given by:

$$P = \sum w_i x_i + w_0 \tag{32}$$

Let x_i and x_j be two points where $y_i = 1$ and $y_j = -1$. Let

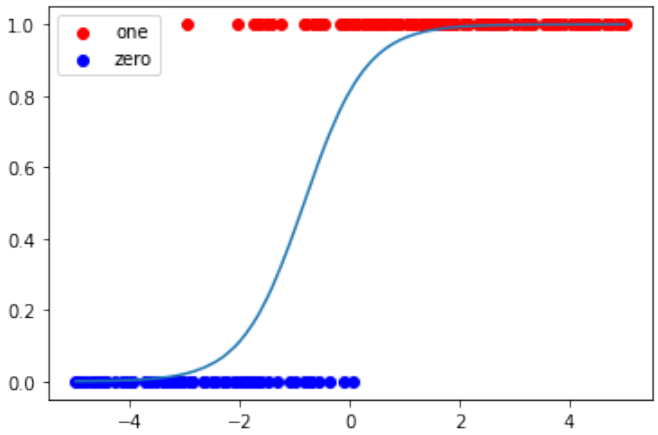

Figure 1. The data suggest that for $u < -1$ or so, the outcome is more likely to be y = 0, while for $u > -1$ or so, the outcome is more likely to be y = 1. The data also suggest that for $u < -3$ or so, the outcome is very likely to be y = 0, and for $u > 2$ or so, the outcome is very likely to be y = 1. The solid curve shows $prob(y = 1) = exp(au + b)/(1 + exp(au + b))$ for the maximum likelihood parameters a, b.

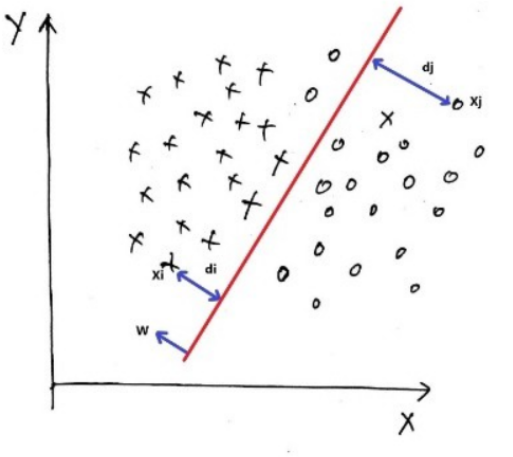

Figure 2. X_i and X_j are correctly classified points

the distances be

$$d_i = \frac{W^T x_i + w_0}{||W||} \tag{33}$$

$$d_j = \frac{W^T x_j + w_0}{||W||} \tag{34}$$

Here W is normal to the plane. When a point is in direction to normal to the plane then the distance is positive else it's negative. If the points were correctly classified then

$$d_i > 0, d_j < 0 \tag{35}$$

We can this say that $d_i * y_i > 0$. For a wrongly classified point, $(y_i * d_i)$ is always negative Hence to get the optimal solution

we need to maximize $(y_i * d_i)$. We need to find optimal W, w_0 which maximizes the below equation.

$$W, w_0 = argmax(\sum y_i * (W^T * x_i + w_0)) \quad (36)$$

The need for Sigmoid Squashing: Since an outlier point of the

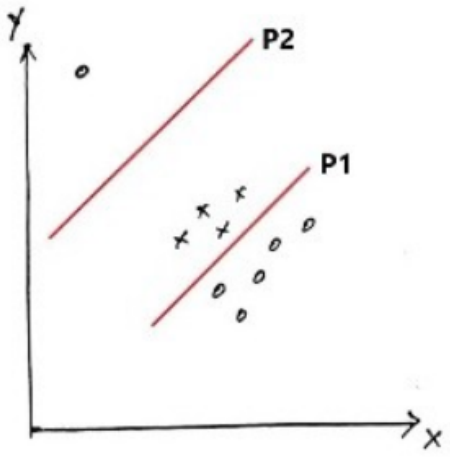

Figure 3. P1 is a better classifier than P2

negative class is present, cost function will choose 'P2' as an optimal plane even though 'P1' best separates the two class of points. The presence of an outlier or extreme point can affect the plane to a great extent. To avoid such condition we need to find a function that makes the $(y_i * d_i)$ value small if it is too large and if $(y_i * d_i)$ value is small it should remain small. We hence resolve to the Sigmoid function. Our equation becomes

$$W, w_0 = argmax \left(\sum \frac{1}{1 + \exp\left(-y_i * (W^T * x_i + w_0)\right)} \right) \quad (37)$$

Since $\log x$ is a monotonically increasing function, maximising $\log x$ is equivalent to maximizing x.Thus,

$$W, w_0 = argmax \left(\sum \log \frac{1}{1 + \exp\left(-y_i * (W^T * x_i + w_0)\right)} \right) \quad (38)$$

We can re-order the data so for $x_1, ..., x_q$, the outcome is y = 1, and for $x_{q+1}, ..., x_m$ the outcome is y = -1. The likelihood function then has the same form as (14).

IV. Approximate linear discrimination via logistic modeling

In linear discrimination, we seek an affine function $f(x) = \beta^T x$ that classifies the points, i.e.,

$$\beta^T x_i > 0, \quad i = 1, .., M \quad \beta^T y_i < 0 \quad i = 1, .., N \quad (39)$$

When two sets of points cannot be linearly separated, we need to seek an affine function that approximately classifies the points, such that the number of points misclassified , is minimum.

An approach to finding an affine function that approximately classifies two sets of points that cannot be linearly separated is based on the logistic model as we have described in the logistic regression problem.

After the maximum likelihood value β has been found by solving the convex optimization problem that minimized $(-l(\beta))$, we can form a linear classifier $f(x) = \beta^T x$ for the two sets of points.

This classifier has the following property:

1.Assuming the data points are in fact generated from a logistic model with parameters β, it has the smallest probability of misclassification, over all linear classifiers.

2.The hyperplane $\beta^T x = 0$ corresponds to the points where prob(y = 1) = 1/2, i.e., the two outcomes are equally likely

V. Regularization

We have the following convex optimization problem:-

To minimize (-(14)), i.e

$$minimize \quad L(\beta) = -\sum_{i=1}^{q} \beta^T x_i + \sum_{i=1}^{m} \log(1 + e^{\beta^T x_i}) \quad (40)$$

Regularization is a technique used to prevent overfitting problem. It adds a regularization term to the equation (39) in order to prevent overfitting of the model.

By using L2 Regularization also known as the Ridge Regularization (39) becomes,

$$L(\beta) = -\sum_{i=1}^{q} \beta^T x_i + \sum_{i=1}^{m} \log(1 + e^{\beta^T x_i}) + \frac{\lambda}{2m} \sum_{i=1}^{m} \beta_i^2 \quad (41)$$

(*Note:* We know that $|\beta|_2 = \frac{\lambda}{2m} \sum_{i=1}^{m} \beta_i^2$ is a convex function, and the sum of convex functions is a convex function, Hence after regularization $L(\beta)$ remains convex and hence can be solved as a convex optimization problem.)

In (41) λ is called the regularization parameter. It controls the trade of between two goals:

1.Fitting the data well.

2.Keeping the parameter β small to avoid overfitting

Note: we need to be careful while deciding on the value of λ as a large value of λ will lead to the values of β_i shrinking to 0 and taking $\lambda = 0$ will have no regularization effect

VI. Comparison to other models

A. Advantages of Logistic regression

- **Simplicity**- Easy to implement and interpret.
- **Efficiency** - Provides great training efficiency and does not require high computation power.Training time is also much less than most of the complex algorithms.
- **Scalable**-Works well with large data sets in moderate time
- **Versatile** - It is easy to update to reflect new data.
- **Diverse Application** - It can easily be extended to multinomial regression

- **Accuracy** - Less prone to over fitting in low dimensional data sets and very accurate for linearly separable data sets
- **Diverse Output**-It provides not only relevance of predictors but probability predictions as well as its direction of association

B. Disadvantages of Logistic regression

- **Over fitting**-In case of high dimensional data set and little training data, over fitting can be an issue
- **Linearity**-it produces linear boundaries ,hence is only applicable for linear problems
- **Limited Use**- It is only applicable for discrete functions
- **High Data Maintenance**- Multiple repetitions in training data set leads to less accurate classification
- **Data Requirement**-Requires diverse data sets that cover all variations
- **Sensitivity**-It is overly sensitive to outliers

VII. Real-world Examples of Logistic Regression Application

- It can be used in the field of medicine to determine the probability of a patient developing a particular disease, and help in the research of finding the potential risk factors of a particular disease.
- Weather prediction.
- In Natural Language Processing (NLP), it's used to determine the sentiment of movie reviews
- Predict if an analog-sensor reading is normal or anomalous
- Image Segmentation and Categorization
- Handwriting recognition
- Geographic Image Processing

VIII. Finding the linear classifier using Logistic Regression in CVXPY

A. One dimensional Data

We use one dimensional data to easily visualise the line produced. The code can be found here-

```
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(z):
  return 1/(1 + np.exp(-z))
## dataset
n=1
m=100
beta_true = np.array([ 0.5] )
print(beta_true.shape)
a=np.random.random(1)
X = (np.random.random((m, n))-0.5 )*10
Y = np.round(sigmoid(X@beta_true+a
+ np.random.randn(m)*0.5))
X_test = (np.random.random((2*m, n))-0.5
          )*10
Y_test = np.round(sigmoid(X_test @
beta_true +
a+ np.random.randn(2*m)*0.5))
def error_lr(scores, labels):
  scores[scores >=0.5] = 1
  scores[scores < 0.5] = 0

  return np.sum(np.abs(scores - labels))
  /float(np.size(labels))
def error(scores, labels):
  scores[scores > 0] = 1
  scores[scores <= 0] = 0

  return np.sum(np.abs(scores - labels))
  / float(np.size(labels))

beta=cp.Variable((n))
b=cp.Variable(1)
print((X@beta).shape)
log_likelihood = cp.sum(cp.multiply(Y,
X @ beta+b)- cp.logistic(X @ beta+b))
problem = cp.Problem(
cp.Maximize(log_likelihood/m))
problem.solve()
beta=beta.value
b=b.value
train_error_lr = error_lr(
sigmoid(X @ beta+b), Y)
test_error_lr = error_lr(
sigmoid(X_test @ beta+b), Y_test)
train_error_lm = error(
(X @ beta+b), Y)
test_error_lm = error(
(X_test @ beta+b), Y_test)

print(train_error_lr,train_error_lm)
print(test_error_lr,test_error_lm)

one=[]
zero=[]
for i in range(0,len(X)):
  if Y[i]==1:
    one.append(X[i])
  else:
    zero.append(X[i])

one=np.array(one)
zero=np.array(zero)

plt.scatter(one, [1]*len(one), c="red",
label="one")
plt.scatter(zero, [0]*len(zero), c="blue",
label="zero")
z=np.sort(X.ravel())
y=[]
for i in range(0,len(z)):
```

```
    if sigmoid(z[i]*beta+b)>=0.5:
      y.append(1)
    else:
      y.append(0)
plt.plot(z,sigmoid(z*beta+b))
plt.legend()
plt.show()
plt.scatter(one, [1]*len(one), c="red",
label="one")
plt.scatter(zero, [0]*len(zero), c="blue",
label="zero")
print(beta)
z=np.sort(X.ravel())
plt.plot(z,sigmoid(z*beta+b))
plt.plot(z,(z*beta+b))
plt.legend()
plt.show()
"""TEST POINTS"""

onet=[]
zerot=[]
for i in range(0,len(X_test)):
    if Y_test[i]==1:
      onet.append(X_test[i])
    else:
      zerot.append(X_test[i])
onet=np.array(onet)
zerot=np.array(zerot)

plt.scatter(onet, [1]*len(onet), c="red",
label="one")
plt.scatter(zerot, [0]*len(zerot),
c="blue", label="zero")
print(beta)
zt=np.sort(X_test.ravel())
yt=[]
for i in range(0,len(zt)):
    if sigmoid(zt[i]*beta+b)>=0.5:
      yt.append(1)
    else:
      yt.append(0)
plt.plot(zt,sigmoid(zt*beta+b))
plt.legend()
plt.show()
plt.scatter(onet, [1]*len(onet),
c="red",label="one")
plt.scatter(zerot, [0]*len(zerot),
c="blue", label="zero")
print(beta)
zt=np.sort(X_test.ravel())
plt.plot(zt,sigmoid(zt*beta+b))
plt.plot(zt,(zt*beta+b))

plt.legend()
plt.show()
```

```
alpha=cp.Variable((n))
a=cp.Variable(1)
constraints=[cp.norm(a)<=1]
t=cp.Variable(1)
for i in range(0,n):
    constraints+=[a.T@one[i]-b >=t]
for i in range(0,n):
    constraints+=[a.T@zero[i]-b <=-t]

obj=cp.Maximize(t)
prob = cp.Problem(obj,constraints)
prob.solve(verbose=True)
alpha=alpha.value
a=a.value
print(alpha)
"""As we can see above, we cannot find
a plane that perfectly separates the two
sets of points.Hence, we try to use
logistic regression and linear
discrimination via logistic modelling."""
```

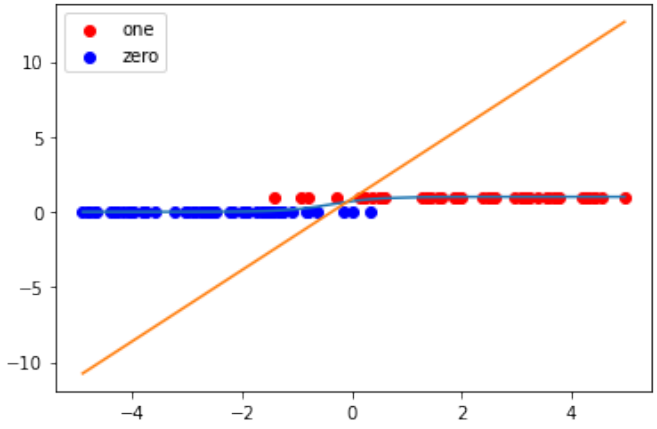

Figure 4. Linear classifier found using Logistic Modelling -Training dataset

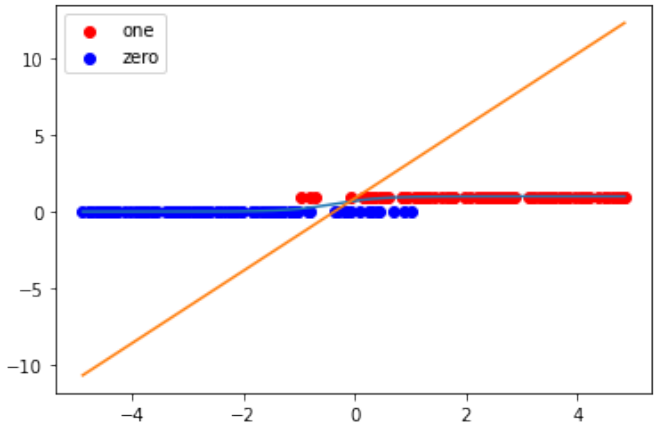

Figure 5. Linear classifier found using Logistic Modelling -Testing dataset

We also found that the points cannot be perfectly linearly separated. Higher dimensional data-

```
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(z):
  return 1/(1 + np.exp(-z))

## dataset
n=100
m=100
beta_true = np.array([1, 0.5, -0.5]
+ [0]*(n - 3))
print(beta_true.shape)
a=np.random.random(1)
X = (np.random.random((m, n))-0.5 )*10
Y = np.round(sigmoid(X@beta_true+a
+ np.random.randn(m)*0.5))

X_test = (np.random.random((2*m, n))
-0.5 )*10
Y_test = np.round(sigmoid(X_test
@ beta_true
+a+ np.random.randn(2*m)*0.5))

def error(scores, labels):
  scores[scores > 0] = 1
  scores[scores <= 0] = 0
  return np.sum(np.abs(scores - labels))
  / float(np.size(labels))

beta=cp.Variable((n))
b=cp.Variable(1)
log_likelihood = cp.sum(cp.multiply
(Y, X @ beta+b)- cp.logistic(X @ beta+b))
problem = cp.Problem(cp.Maximize
(log_likelihood/m ))
problem.solve()
beta=beta.value
b=b.value
train_error_lm = error( (X @ beta+b), Y)
test_error_lm = error( (X_test @ beta+b),
Y_test)

print(train_error_lm)
print(test_error_lm)

"""In case of High Dimensional data,
the training error is very low but the
test error is significantly higher which
is an indication of over fitting."""
```

With higher dimensional data we could observe overfitting. The training and testing errors for points with 100 dimensions were 0 and 0.47. The test error is significantly higher than the training error. Thus, to combat this we use regularisation.

```
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(z):
  return 1/(1 + np.exp(-z))

## dataset
n=100
m=100
beta_true = np.array([1, 0.5, -0.5]
+ [0]*(n - 3))
print(beta_true.shape)
a=np.random.random(1)
X = (np.random.random((m, n))-0.5 )
*10
Y = np.round(sigmoid(X@beta_true+a
+ np.random.randn(m)*0.5))

X_test = (np.random.random((2*m, n))-0.5 )
*10
Y_test = np.round(sigmoid(X_test
@ beta_true
+a+ np.random.randn(2*m)*0.5))

def error(scores, labels):
  scores[scores > 0] = 1
  scores[scores <= 0] = 0
  return np.sum(np.abs(scores - labels))
  / float(np.size(labels))

beta=cp.Variable((n))
b=cp.Variable(1)
log_likelihood = cp.sum(cp.multiply
(Y, X @ beta+b) - cp.logistic(X @ beta+b))
problem = cp.Problem(cp.Maximize
(log_likelihood/m ))
problem.solve()
beta=beta.value
b=b.value

train_error_lm = error( (X @ beta+b), Y)
test_error_lm = error( (X_test @ beta+b)
, Y_test)

print(train_error_lm)
print(test_error_lm)

"""In case of High Dimensional data,
the training error is very low but the
test error is significatnly higher
which is an indication
of over fitting.
```

```
Using Regularisation
"""

beta = cp.Variable(n)
b=cp.Variable(1)
log_likelihood = cp.sum(cp.multiply
(Y, X @ beta+b)-
cp.logistic(X @ beta+b))
intervals=100
train_error = []
test_error = []
lambda_vals =
np.logspace(-2, 5, intervals)
beta_vals = []
b_vals=[]
for i in range(intervals):
    lamb=lambda_vals
    obj=
    cp.Maximize(log_likelihood/m
    - lamb[i]* cp.norm(beta, 1))
    problem = cp.Problem(obj)
    problem.solve()
    train_error.append(error(
    (X @ beta+b).value, Y))
    test_error.append(error(
    (X_test @ beta+b).value, Y_test))
    beta_vals.append(beta.value)
    b_vals.append(b.value)

plt.plot(lambda_vals, train_error,
label="Train error")
plt.plot(lambda_vals, test_error,
label="Test error")
plt.xscale("log")
plt.legend(loc="upper left")
plt.xlabel(r"$\lambda$", fontsize=16)
plt.show()

t=test_error
aa=t.index(min(t))
print(lambda_vals[aa])
print(aa)

print(train_error[aa])
print(test_error[aa])

normbeta=[]
l=[]
for j in range(0,len(beta_vals)):
  normbeta.append(np.sqrt(
  (np.linalg.norm(beta_true -
  beta_vals[j]))**2+(a-b_vals[j])**2))
  l.append(j)

plt.plot(l,normbeta, label=r"Norm of
difference in values for choices
of $\lambda$")
plt.xlabel(r"$i$", fontsize=16)
plt.ylabel(r"$\beta_i$", fontsize=16)
plt.legend(loc="upper right")
plt.show()
"""The above is a reasonable choice
for lambda as it minimises the test
error"""
```

Using regularisation we see a significant reduction in test error which becomes 0.065 and train error becomes 0.06.

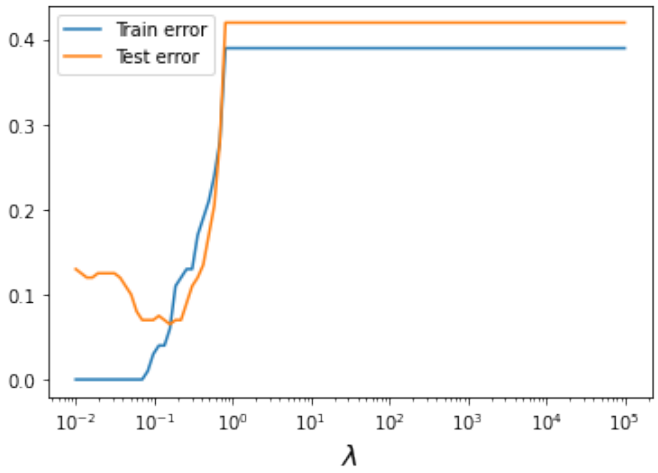

Figure 6. Training and testing error for different values of the regularisation parameter.

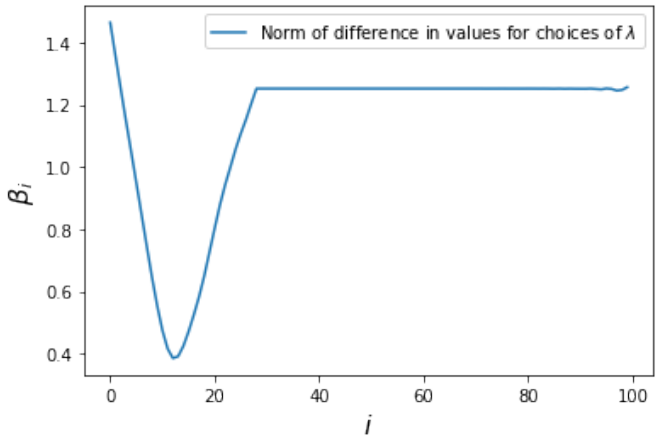

Figure 7. Norm of the difference in original weights and weights found using regularisation for different values of λ

B. Real World Implementation

We tried to apply Logistic regression of CVXPY on a real world dataset concerning with heart disease prediction based on various factors like bp,cholestrol etc. We performed a little data cleansing like replacing null values and added dummy variables for categorical data. Our model performed well and gave us an accuracy of 86.88% accuracy.

```
import pandas as pd
import numpy as np
import io
from matplotlib import pyplot as plt
import math
from sklearn.model_selection import
train_test_split

X_data_frame = pd.read_csv('heart.csv')

X_data_frame.head()
#checking the categories

X_data_frame.tail(3)
#converting categorical variables
to dummy variables
cp = pd.get_dummies(X_data_frame['cp']
, prefix = "cp")
th = pd.get_dummies(X_data_frame['thal']
, prefix = "thal")
sl = pd.get_dummies(X_data_frame['slope']
, prefix = "slope")
frames = [X_data_frame, cp, th, sl]
X_data_frame = pd.concat(frames, axis = 1)
#X_data_frame.head()

#after converting it into categorical
#variables, dropping those variables
X_data_frame = X_data_frame.drop
(columns = ['cp', 'thal', 'slope'])
X_data_frame.head()
y = X_data_frame.target.values
x_data = X_data_frame.drop
(['target'], axis = 1)

#normalising the values of x so
#that the cluster isnt widely spread out
x = (x_data - np.min(x_data))
/ (np.max(x_data) - np.min(x_data)).values

#splitting the data into testing and
#training sets
x_tr, x_te, y_train, y_test =
train_test_split(x,y,test_size = 0.2,
random_state=0)

x_train = (x_tr).to_numpy()
x_test = (x_te).to_numpy()

print(x_train.shape)
print(y_train)

def sigmoid(z):
  return 1/(1 + np.exp(-z))

def error(scores, labels):
  scores[scores > 0] = 1
  scores[scores <= 0] = 0
  return np.sum(np.abs(scores - labels))
  / float(np.size(labels))

import cvxpy as cp

n=21
m=242
beta=cp.Variable((n))
b=cp.Variable(1)
log_likelihood = cp.sum(cp.multiply
(y_train, x_train @ beta+b)
- cp.logistic(x_train @ beta+b))
problem = cp.Problem(cp.Maximize
(log_likelihood/m ))
problem.solve()
beta=beta.value
b=b.value
train_error_lm =
error( (x_train @ beta+b), y_train)
print(train_error_lm)

test_error_lm =
error( (x_test @ beta+b), y_test)
print(test_error_lm)

beta = cp.Variable(n)
b=cp.Variable(1)
log_likelihood =
cp.sum(cp.multiply(y_train,
x_train @ beta+b)
- cp.logistic(x_train @ beta+b))
intervals=100
train_error = []
test_error = []
lambda_vals =
np.logspace(-2, 5, intervals)
beta_vals = []
b_vals=[]
for i in range(intervals):
    lamb=lambda_vals
    obj=cp.Maximize(log_likelihood/m -
    lamb[i] * cp.norm(beta, 1))
    problem = cp.Problem(obj)
    problem.solve()
    train_error.append(error(
    (x_train @ beta+b).value, y_train))
    test_error.append(error(
    (x_test @ beta+b).value, y_test))
    beta_vals.append(beta.value)
    b_vals.append(b.value)

t=test_error
aa=t.index(min(t))
print(lambda_vals[aa])
```

```
print(t[aa])

plt.plot(lambda_vals, train_error,
label="Train error")
plt.plot(lambda_vals, test_error,
label="Test error")
plt.xscale("log")
plt.legend(loc="upper left")
plt.xlabel(r"$\lambda$", fontsize=16)
plt.show()

y_prediction=[]
for i in range(0,len(y_test)):
  h=x_test[i] @ beta_vals[aa]
  +b_vals[aa]
  if h <= 0:
    y_prediction.append(0)
  else:
    y_prediction.append(1)

acc=0
wro=0
for i in range(0,len(y_test)):
  if y_test[i]==y_prediction[i]:
    acc=acc+1
  else:
    wro=wro+1
print(acc/(acc+wro))
#accuracy percentage of my model
```

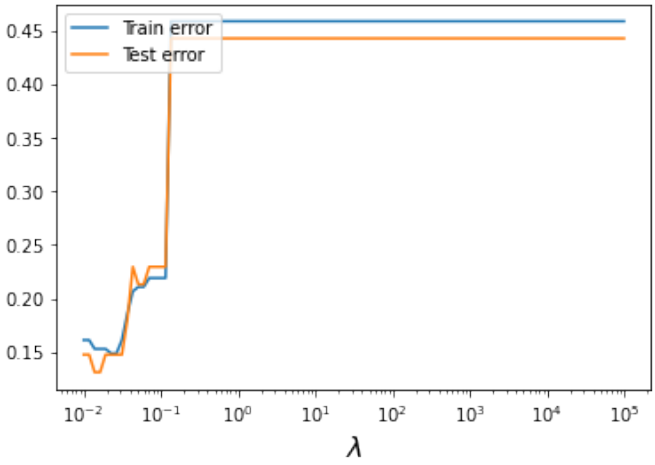

Figure 8. Training and testing errors for different values of λ

REFERENCES

[1] https://web.stanford.edu/boyd/cvxbook/bv_cvxbook.pdf
[2] https://towardsdatascience.com/geometric-interpretation-of-logistic-regression-4f85047a5860
[3] https://medium.com/@aditya97p/l1-and-l2-regularization-237438a9caa6
[4] https://www.geeksforgeeks.org/advantages-and- disadvantages-of-logistic-regression/
[5] https://iq.opengenus.org/advantages-and-disadvantages-of- logistic-regression/
[6] https://www.cvxpy.org/examples/machine_learning/logistic_regression.html
[7] Kaggle for the heart disease dataset