Sali
M'jahad
msm2243

MACHINE LEARNING

P1

(a)

$$\hat{\pi} = \arg\max_{\pi} \sum_{i=1}^{n} \ln p(y_i | \pi)$$

$$= \arg\max \sum_{i=1}^{n} y_i \ln \pi + (1-y_i) \ln(1-\pi)$$

$$\frac{\partial \ell}{\partial \pi} = 0 \Rightarrow \frac{\sum y_i}{n}$$

(b)

$$\hat{\theta}_y^{(1)} = \arg\max_{\theta_y^{(1)}} \sum \ln p(x_{i1} | \theta_y^1)$$

$$= \arg\max_{\theta_y^{(1)}} \sum x_{i1} \ln \theta_{yi}^1 + (1-x_{i1}) \ln(1-\theta_{yi}^1)$$

$$= \arg\max_{\theta_y^{(1)}} \sum x_{i1} \ln \theta_y^1 + (1-x_{i1}) \ln(1-\theta_{yi}^1) \mathbb{1}(y_i=y)$$

$$\frac{\partial \ell}{\partial \hat{\theta}_y^{(1)}} = 0 \iff \hat{\theta}_y^{(1)} = \frac{\sum_{i=1}^{n} x_{i1} \mathbb{1}(y_i=y)}{\sum_{i=1}^{n} \mathbb{1}(y_i=y)} \quad y \in \{0,1\}$$

(c)

$$\hat{\theta}_y^{(2)} = \arg\max_{\theta_y^2} \sum_{i=1}^{n} \ln p(x_{i2} | \theta_y^2)$$

$$= \arg\max_{\theta_y^2} \sum_{i=1}^{n} \ln \theta_{yi}^{(2)} - (1-\theta_{yi}^{(2)}) \ln x_{i2}$$

[illegible]

$\Rightarrow$ (we derive Pareto ML :

$$\hat{\theta}_{ML} = \frac{n}{\sum^n \ln x_i} \quad)$$

$$\frac{\partial \ell}{\partial \hat{\theta}_y^2} = 0 \Leftrightarrow \hat{\theta}_y^{(2)} = \frac{\sum_i^n \mathbb{1}(y_i = y)}{\sum_{i=1}^n \mathbb{1}(y_i = y) \ln x_i}$$

□

HW2

February 25, 2018

1 HOMEWORK 2

1.1 CODE

In [507]:
```
import matplotlib.pyplot as plt
import numpy as np
import math

X_train = np.array(pd.read_csv("./hw2-data/X_train.csv", header=None))
X_test = np.array(pd.read_csv("./hw2-data/X_test.csv", header=None))
y_train = np.array(pd.read_csv("./hw2-data/y_train.csv", header=None))
y_test = np.array(pd.read_csv("./hw2-data/y_test.csv", header=None))
# X_train.head()

# HELPER FUNCTIONS
def BLL(X, T):
    return np.sum(X*np.log(T) + (1-X)*np.log(1-T), axis=1)
def PLL(X, T):
    return np.sum(np.log(T) - (T+1)*np.log(X), axis=1)

# inpired from https://timvieira.github.io/blog/post/2014/02/11/exp-normalize-trick/
def sigmoid(x):
    res = []
    for i in x:
        if i >= 0:
            z = np.exp(-i)
            res.append(1 / (1 + z))
        else:
            z = np.exp(i)
            res.append(z / (1 + z))
    return np.array(res)

# BAYES_NAIVE_CLASSIFIER
class BNC():
    def fit(self, X, y):
        self.BNC_pi_0 = np.extract(y==0, y).size/y.size
        self.BNC_pi_1 = np.extract(y==1, y).size/y.size
        BX, PX = np.split(X, [54], axis =1)
```

```
        self.BNC_BT_0 = np.sum(BX[(y == 0).flatten(), :], axis=0)/np.extract(y==0, y).
        self.BNC_BT_1 = np.sum(BX[(y == 1).flatten(), :], axis=0)/np.extract(y==1, y).
        self.BNC_PT_0 = np.extract(y==0, y).size/ np.sum(np.log(PX[(y == 0).flatten(),
        self.BNC_PT_1 = np.extract(y==1, y).size/ np.sum(np.log(PX[(y == 1).flatten(),

    def forward_pass(self, X):
        BX, PX = np.split(X, [54], axis =1)
        y_pred_0 = np.log(self.BNC_pi_0) + BLL(BX, self.BNC_BT_0) + PLL(PX, self.BNC_P

        y_pred_1 = np.log(self.BNC_pi_1) + BLL(BX, self.BNC_BT_1) + PLL(PX, self.BNC_P
        return (y_pred_1>y_pred_0).astype(int).reshape((-1,1))

    def get_BT(self):
        return self.BNC_BT_0, self.BNC_BT_1

# K-Nearest Neighbour Classifier
class KNNC():
    def fit(self, X, y): #almost like init
        self.X = X
        self.y = y

    def ranked_NNs(self, X_t):
        return np.argsort(np.array([[np.sum(np.abs(j-i)) for j in self.X] for i in X_t

    def forward_pass(self, k, ranked_nns):
        return np.around(np.mean(self.y[ranked_nns[:, :k]], axis=1)).astype(dtype=np.i

# Logistic Regression

class LRC():
    def fit_train(self, X_l, y_l, n_iterations = 10000, LR_fun = "GA"):
        self.X = np.hstack((X_l, np.ones((X_l.shape[0], 1))))
        self.y = np.where(y_l == 1, 1, -1)
        self.w = np.zeros((self.X.shape[1], 1))
        self.iters = []
        self.Ls = []
        l_rate = 0
        for i in range(n_iterations):
            if LR_fun == "GA":
                l_rate = 1.0/(10**5*np.sqrt(i+1))
                L = np.sum(np.log(sigmoid(self.y*np.dot(self.X, self.w))+1e-10))
                self.w = self.w + l_rate*(np.sum((1-sigmoid(self.y*np.dot(self.X, self
                                                  )*self.y*self.X, axis=0).reshape(self
                self.iters.append(i)
                self.Ls.append(L)
            elif LR_fun == "N":
                l_rate = 1.0/np.sqrt(i+1)
                L = np.sum(np.log(sigmoid(self.y*np.dot(self.X, self.w))+1e-10))
```

```
                    grad = np.sum((1-sigmoid(self.y*np.dot(self.X, self.w)))*self.y*self.X
                    term = sigmoid(np.dot(self.X,self.w))
                    second = np.zeros((self.X.shape[0],self.X.shape[1],self.X.shape[1]))
                    for j in range(self.X.shape[0]):
                        arr=self.X[j].reshape([-1,1])
                        t3=np.dot(arr, arr.T)
                        second[j] = term[j]*(1-term[j])*t3
                    second = -np.sum(second, axis=0)
                    self.w += -np.dot(np.linalg.inv(second),grad)
                    self.iters.append(i)
                    self.Ls.append(L)
                else:
                    print("Invalid learning rate function")
            return np.array(self.iters), np.array(self.Ls)

        def forward_pass(self, Xt):
            return (sigmoid(np.dot(np.hstack((Xt, np.ones((Xt.shape[0], 1)))), self.w))>0.
```

1.2 PROBLEMS

1.2.1 Problem 2 - a

```
In [513]: bnc = BNC()
          bnc.fit(X_train, y_train)
          results = bnc.forward_pass(X_test)
          print("ACCURACY: ", np.extract(y_test==results, results).size/y_test.size*100)

          OO = np.extract(np.logical_and(y_test == 0, results == 0), y_test).size
          OI = np.extract(np.logical_and(y_test == 0, results == 1), y_test).size
          IO = np.extract(np.logical_and(y_test == 1, results == 0), y_test).size
          II = np.extract(np.logical_and(y_test == 1, results == 1), y_test).size

          pd.DataFrame(np.array([[OO, OI],
                    [IO, II]]))
```

```
ACCURACY:  92.47311827956989
```

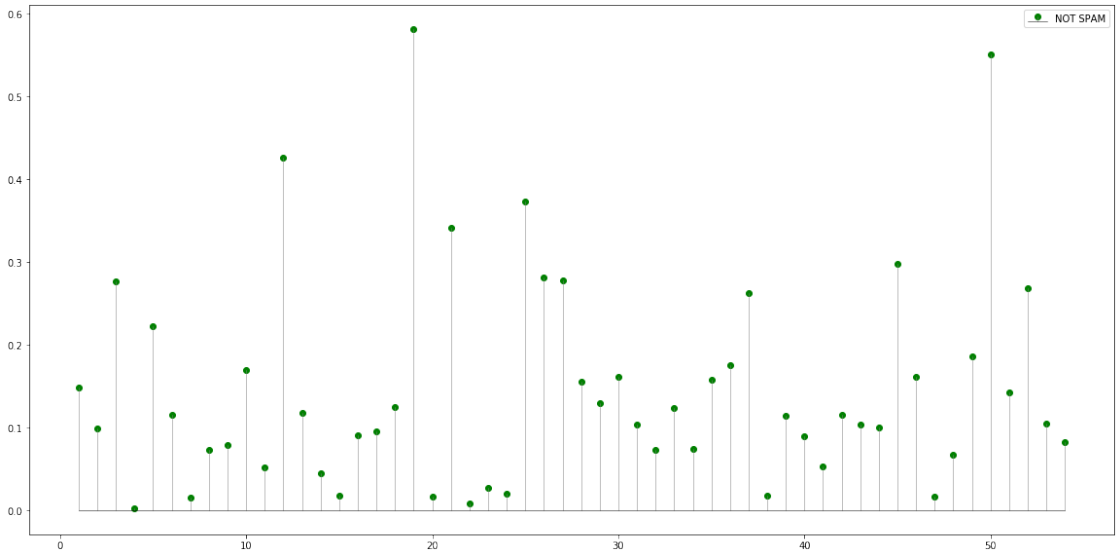
```
Out[513]:     0   1
          0  54   2
          1   5  32
```

1.2.2 Problem 2 - b

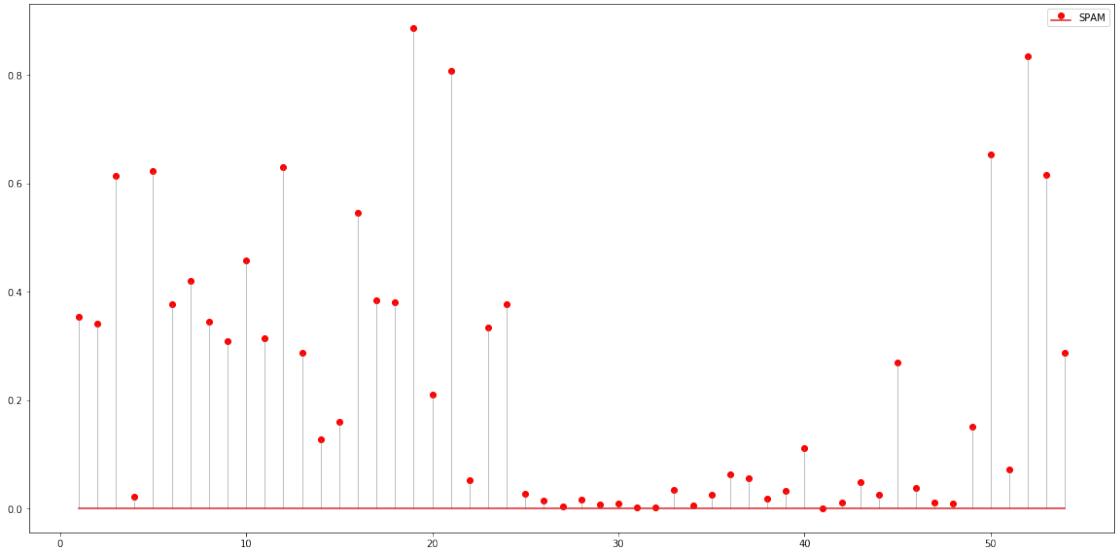
```
In [514]: import matplotlib.pyplot as plt

          indexes = np.arange(start=1, stop=55)
          bt_0, bt_1 = bnc.get_BT()
          plt.figure(figsize=(20, 10))
```

```
        markerline, stemlines, baseline = plt.stem(indexes, bt_0, label="NOT SPAM")
        plt.setp(baseline, color='black', linewidth=.5)
        plt.setp(markerline, color='g')
        plt.setp(stemlines, color='grey', linestyle='-', linewidth=.5)
        plt.legend()
        plt.show()
```



In [515]:
```
        plt.figure(figsize=(20, 10))
        markerline, stemlines, baseline = plt.stem(indexes, bt_1, label="SPAM")
        plt.setp(markerline, color='r')
        plt.setp(stemlines, color='grey', linestyle='-', linewidth=.5)
        plt.legend()
        plt.show()
```



dim 16 corresponds to the word "free". bernoulli's theta for spam case is high compared to that of not spam case meaning the existence of the word free in an email positively contributes to the probability of it being a spam compared to not spam.

same with dim 52 which corresponds to "!".

this does make sense because promotional spam offer "free" services and products and "!" can be used as a way to make the offers sounds exciting.

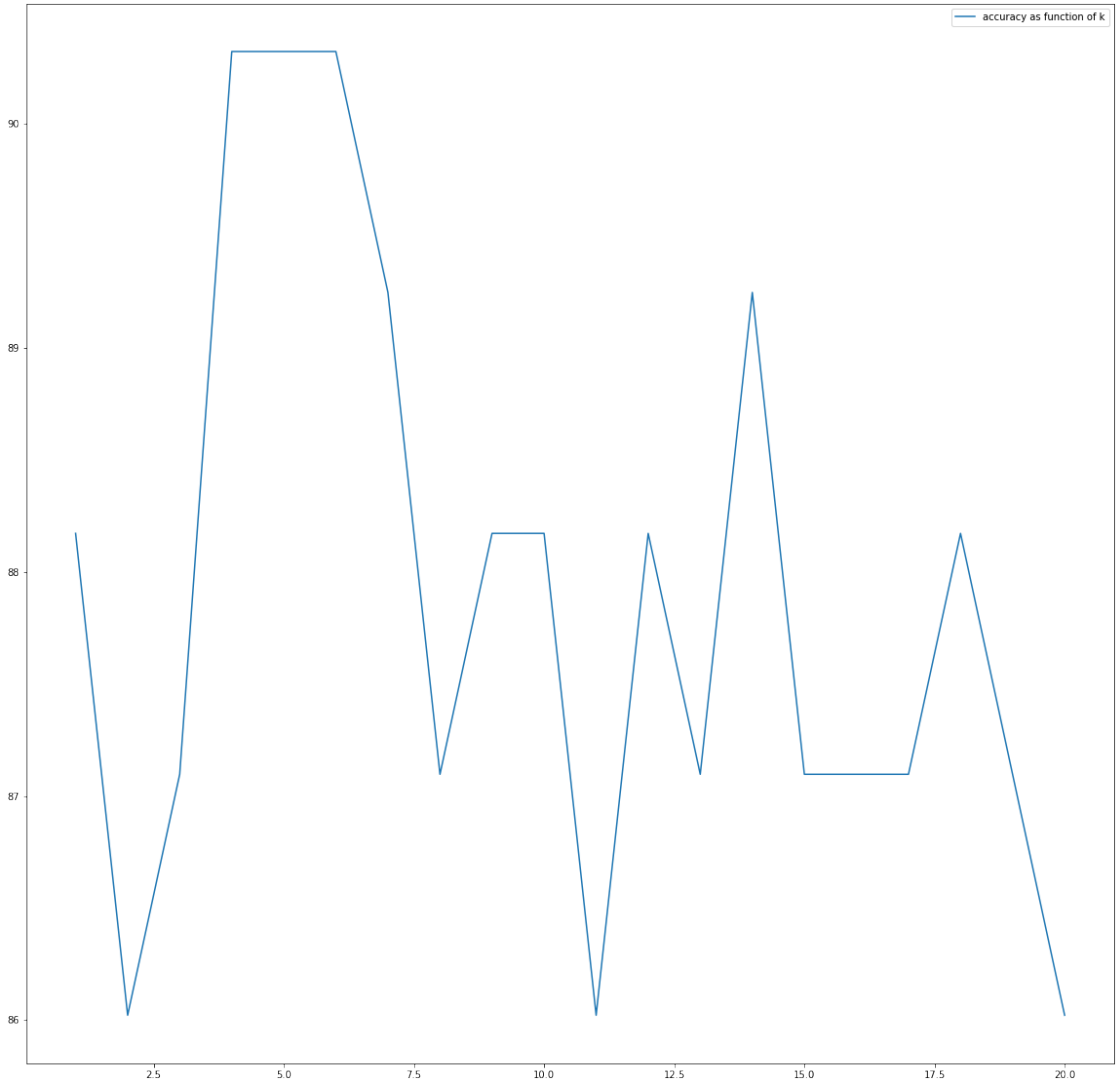
1.2.3 Problem 2 - c

In [516]:
```
knnc = KNNC()
knnc.fit(X_train, y_train)
RNNs = knnc.ranked_NNs(X_test)
preds= []
accuracies = []
for k in range(1,21):
    pred = knnc.forward_pass(k, RNNs).flatten()
    preds.append(pred)

preds= np.array(preds)


for k in range(1,21):
    count = 0
    for i, p in enumerate(preds[k-1, :]):
        if y_test.flatten()[i] == p:
            count += 1
    accur = count/y_test.size*100
    accuracies.append(accur)

plt.figure(figsize=(20, 20))
plt.plot(range(1,21), accuracies, label= "accuracy as function of k")
plt.legend()
plt.show()
```

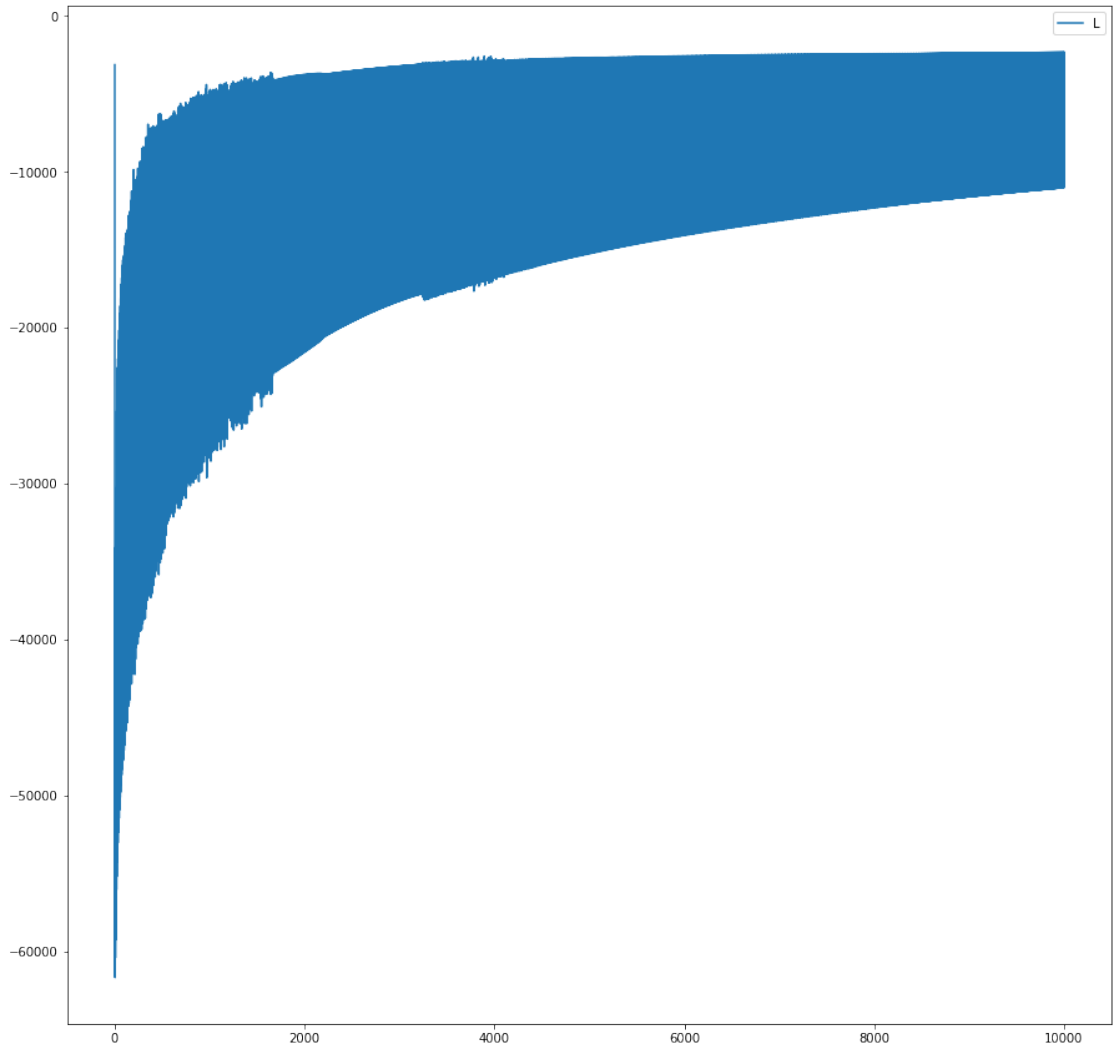


1.2.4 Problem 2 - d

```
In [517]: lrcGA = LRC()
          itersGA, LsGA = lrc.fit_train(X_train, y_train, n_iterations = 10000, LR_fun = "GA")
```

The previous step is super long :(

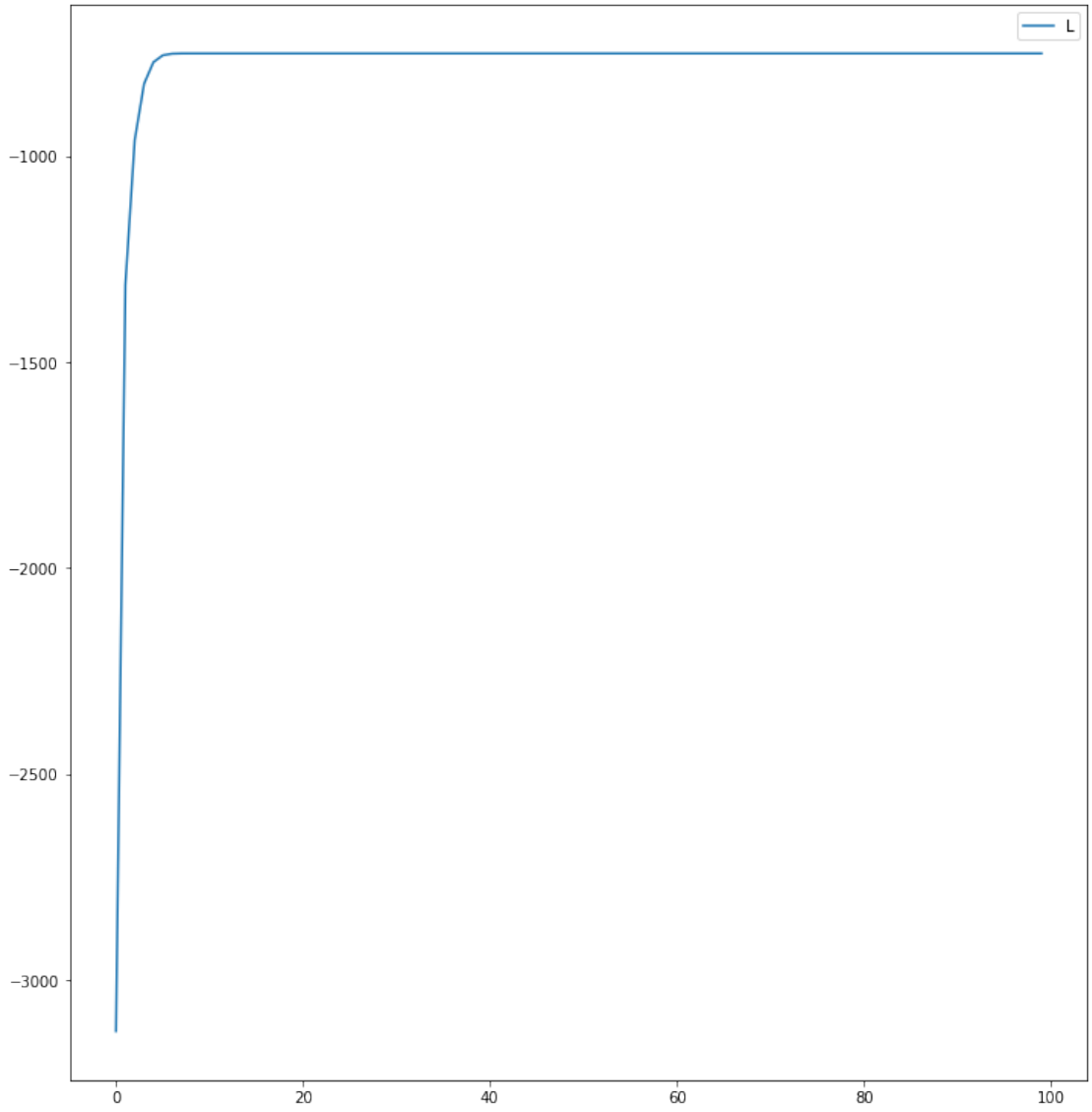
```
In [518]: plt.figure(figsize=(15, 15))
          plt.plot(itersGA, LsGA, label='L')
          plt.legend()
          plt.show()
```



1.2.5 Problem 2 - e

```
In [519]: lrcN = LRC()
          itersN, LsN = lrcN.fit_train(X_train, y_train, n_iterations = 100, LR_fun = "N")

In [520]: plt.figure(figsize=(12, 13))
          plt.plot(itersN, LsN, label='L')
          plt.legend()
          plt.show()
```




```
In [521]: print("Accuracy: ", np.extract(y_test==lrcN.forward_pass(X_test), y_test).size/y_test.

Accuracy:  91.39784946236558
```