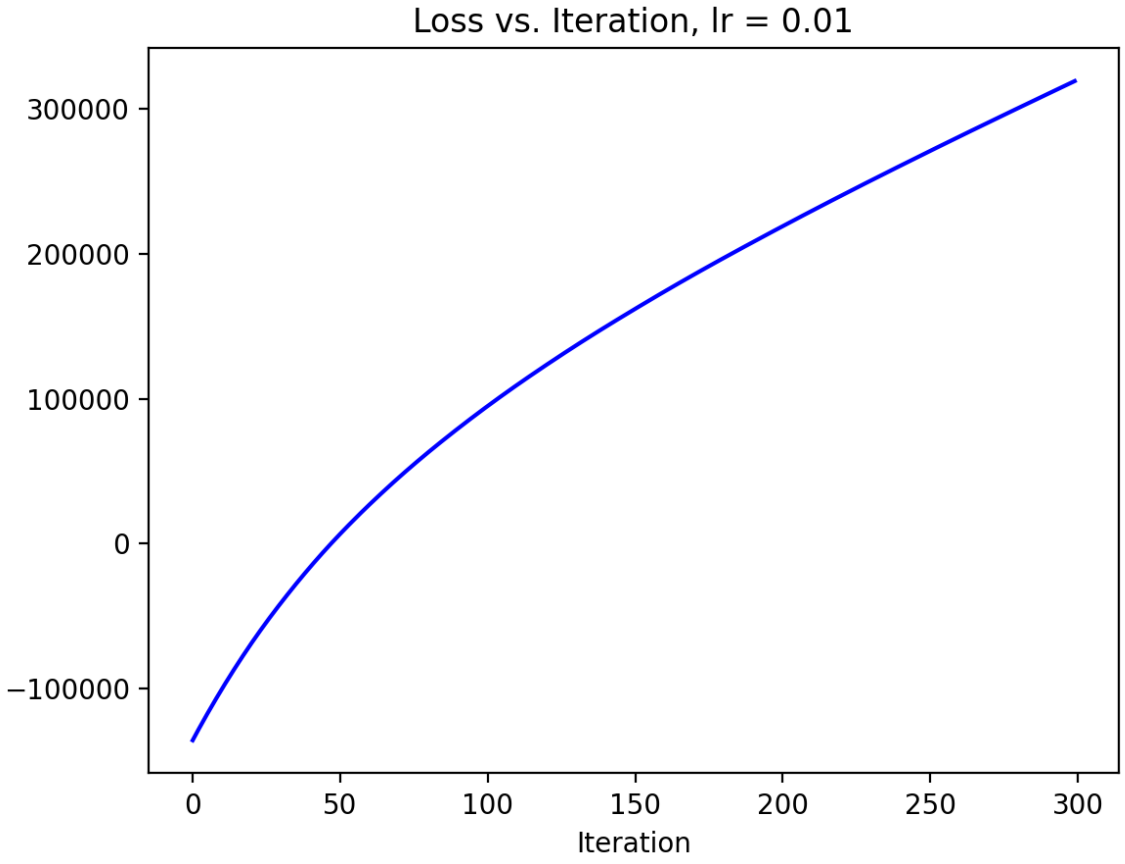
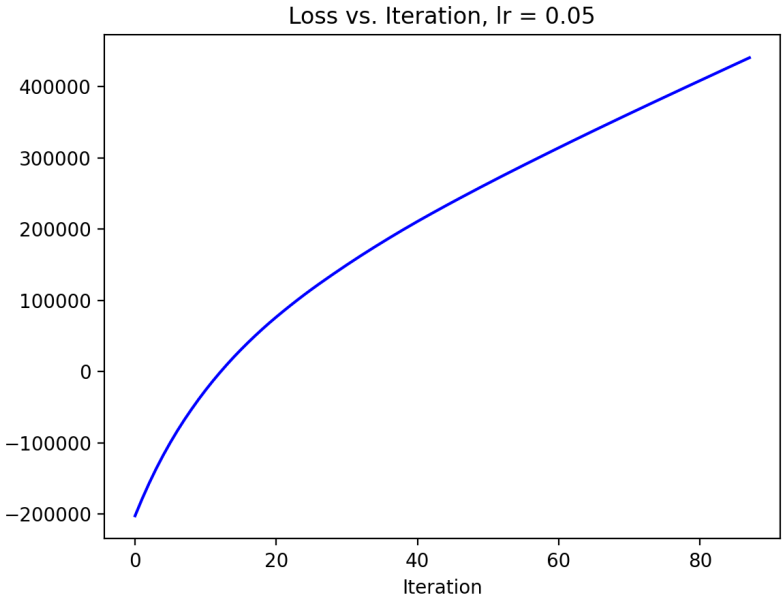
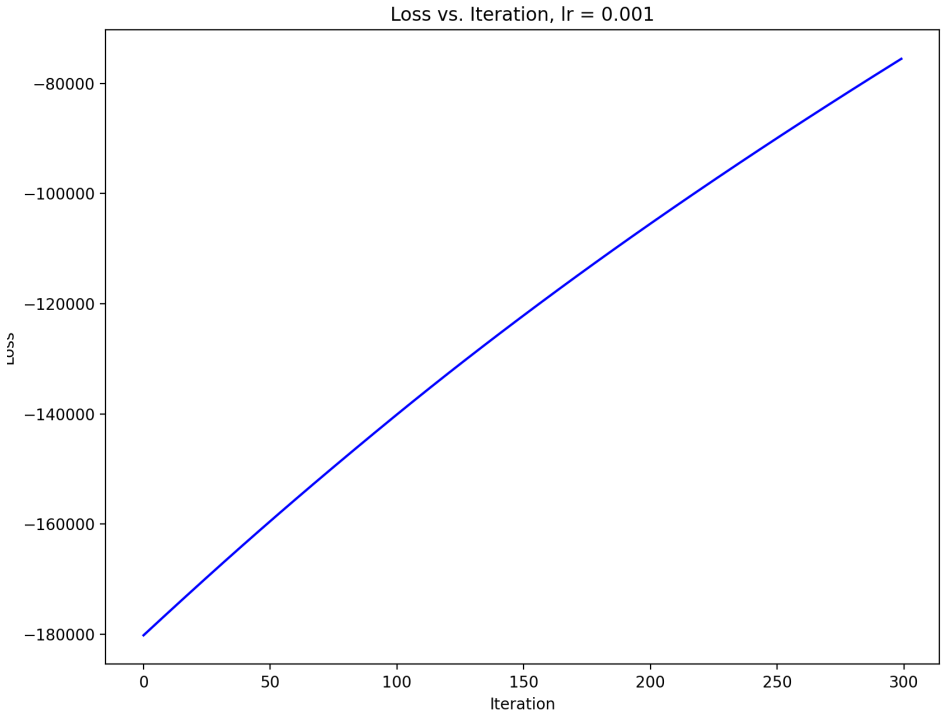
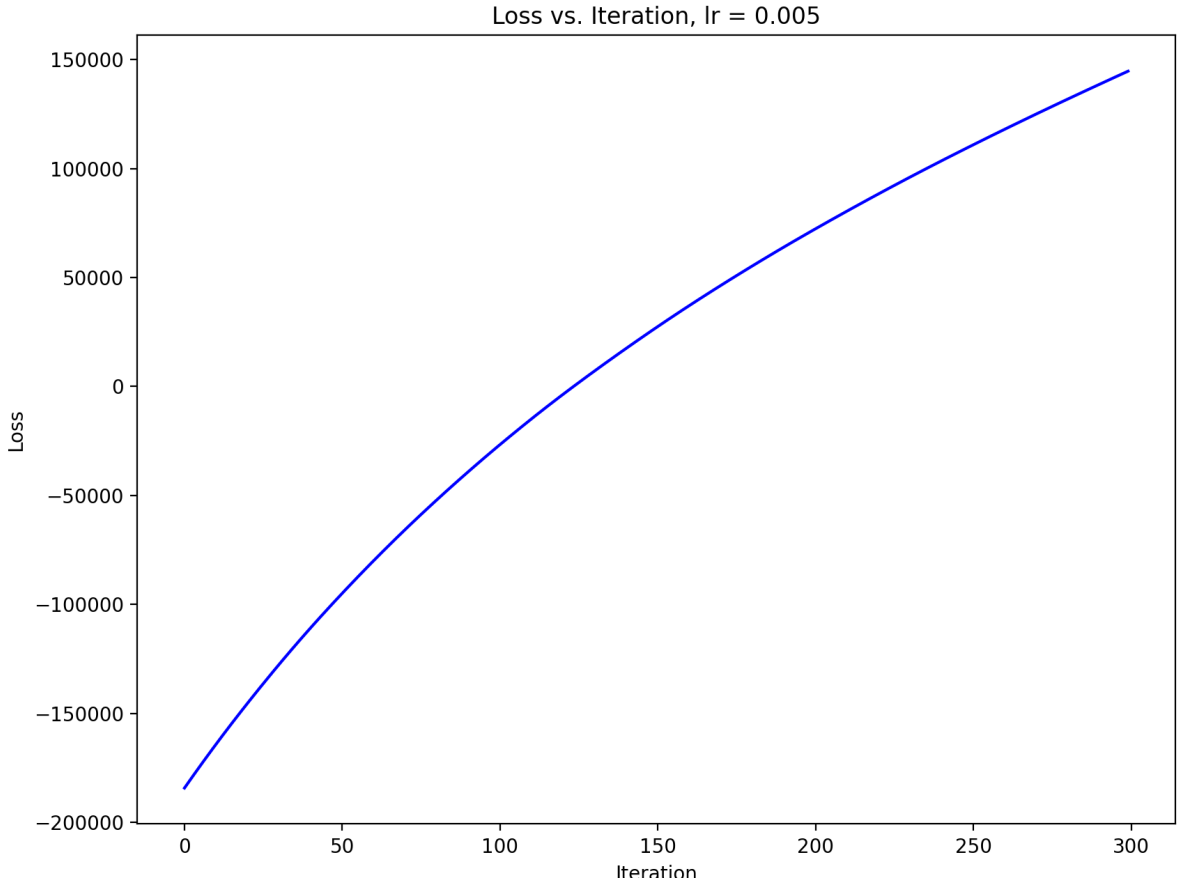
Jeret McCoy, Abelardo Riojas, Gustavo Flores
Part A

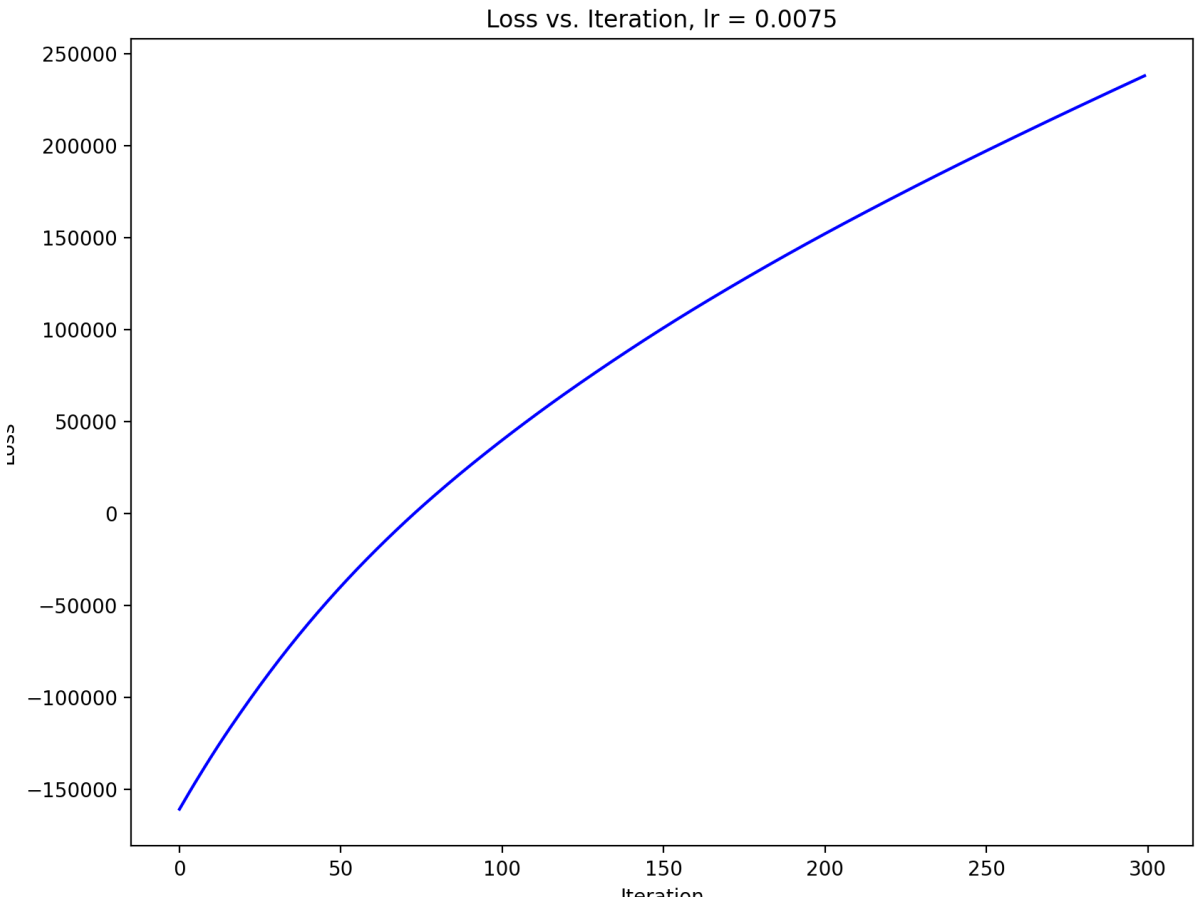
LearnRate	Train Error	Test Error
0.01	11.7%	25.7%
0.05	50%	50%
0.001	40.6%	48.5%
0.005	20.6%	23.3%
0.0075	14.9%	19.2%




Loss vs. Iteration, lr = 0.05
400000
300000
200000
100000
0
−100000
−200000
0
20
40
60
80
Iteration


Loss vs. Iteration, lr = 0.001
−80000
−100000
−120000
−140000
−160000
−180000
Loss
0
50
100
150
200
250
300
Iteration


Loss vs. Iteration, lr = 0.005
150000
100000
50000
0
−50000
−100000
−150000
−200000
Loss
0
50
100
150
200
250
300
Iteration



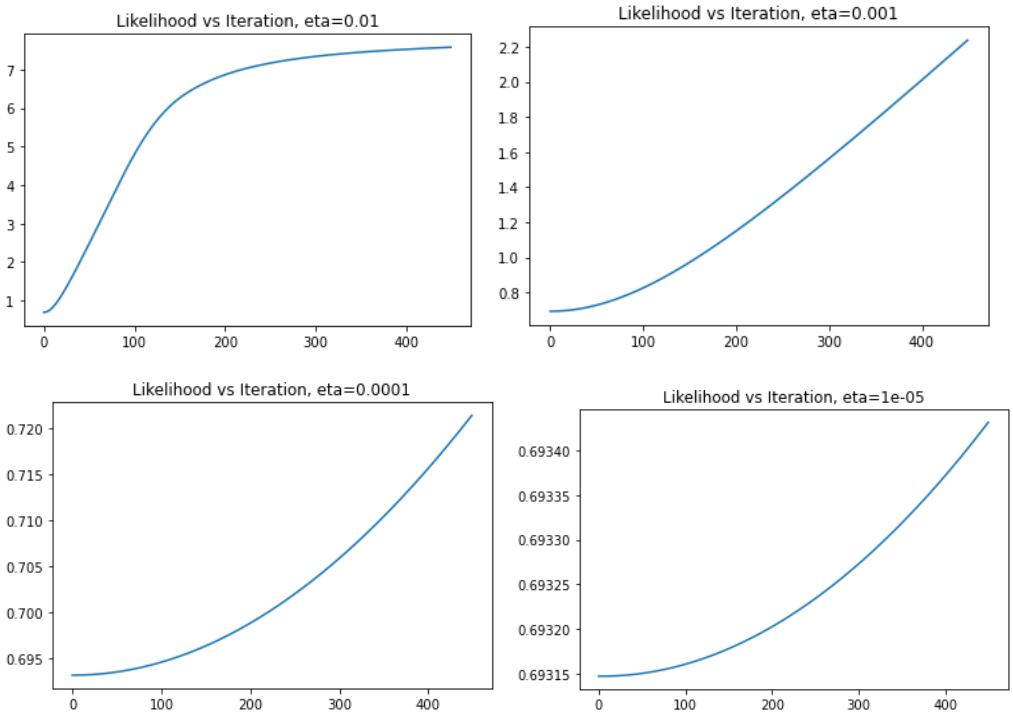

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


class LiteLR:
    def __init__(self, learnRate, X, y, iterations=300, shrinkage=0.0001):
        self.iterations = iterations
        self.shrinkage = shrinkage
        self.learnRate = learnRate
        self.X = X
        self.y = y
        self.weights = np.random.randn(X.shape[1])
    def L(self):
        loss = 0
        for j, obj in enumerate(self.X): #for each row in X
```

```
            loss = loss + (   (self.y[j] * np.dot(self.weights, obj))  -
np.log(1 + np.exp( np.dot(self.weights, obj)  ))  )
        return loss
    def dL(self):
        dloss = 0
        for j, obj in enumerate(self.X): #for each row in X
            dloss = dloss + (  obj * (self.y[j] -  (
(np.exp(np.dot(self.weights,obj)))/(1 + np.exp(np.dot(self.weights,obj)))
)    )  )
        return dloss
    def gradientAscent(self, plot=False): #trains model, returns loss
        losses = list()
        for i in range(self.iterations):
            self.weights = self.weights - (self.learnRate * self.shrinkage
* self.weights) + ((self.learnRate/self.X.shape[1])* self.dL())
            losses.append(self.L())
        if (plot == True):
            itrs = range(self.iterations)
            plt.plot(itrs, losses , c='b')
            plt.xlabel("Iteration")
            plt.ylabel("Loss")
            plt.title("Loss vs. Iteration, lr =
{}".format(self.learnRate))
            plt.show()
        return self.L()
    def error(self, X, y):
        ret = 0
        for j, obj in enumerate(X):
            hyp = 1 / (1 + np.exp(-1 * np.dot(self.weights, obj)))
            pred = -1
            if (hyp > .5):
                pred = 1
            if (pred != y[j]):
                ret = ret + 1
        return ret / X.shape[0]


trainX = pd.read_csv("gisette_train.data", sep=' ', header=None)
trainX = np.array(trainX)
trainX = trainX[:, 0:5000]
```

```
testX = pd.read_csv("gisette_valid.data", sep=' ', header=None)
testX = np.array(testX)
testX = testX[:, 0:5000]
trainy = pd.read_csv("gisette_train.labels", sep=' ', header=None)
trainy = np.array(trainy)
testy = pd.read_csv("gisette_valid.labels", sep=' ', header=None)
testy = np.array(testy)
for i in range(5000):
    if (np.std(trainX[:,i]) != 0):
        trainX[:,i] = (trainX[:,i] - np.mean(trainX[:,i])) /
np.std(trainX[:,i])
        testX[:,i] = (testX[:,i] - np.mean(trainX[:,i])) /
np.std(trainX[:,i])


clf = LiteLR(.0075, trainX, trainy)
loss = clf.gradientAscent(plot=True)
print(loss)
print("Train error: {} Test error: {}".format(     clf.error(trainX,
trainy) , clf.error(testX,testy)   ))
```

Part B:

Learning Rate	Train Misclass Error	Test Misclass Error
.01	15%	15%
.001	15%	15%
.0001	15%	15%
.00001	15%	15%




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

X = pd.read_csv('X.dat', sep = ' ', header=None)
Xtest = pd.read_csv('Xtest.dat', sep = ' ', header=None)
Y = pd.read_csv('Y.dat', sep = ' ', header=None)
Ytest = pd.read_csv('Ytest.dat', sep = ' ', header=None)

for i, obs in enumerate(X):
    X[i] = (obs - obs.mean())/(obs.std())
for i, obs in enumerate(Xtest):
    Xtest[i] = (obs - obs.mean())/(obs.std())

class MyLogisticRegression:
    def __init__(self, learning_rate=0.01, max_iterations=450,
shrinkage=.0001):

        # Initialising all the parameters
        self.learning_rate  = learning_rate
        self.max_iterations = max_iterations
        self.shrinkage = shrinkage
        self.likelihoods    = []
```

```
        # Define epsilon because log(0) is not defined
        self.eps = 1e-7

    def sigmoid(self, z):

        ### START CODE HERE
        sig_z = (1/(1+np.exp(-z)))
        ### END CODE HERE

        assert (z.shape==sig_z.shape), 'Error in sigmoid implementation. Check
carefully'
        return sig_z

    def log_likelihood(self, y_true, y_pred):

        # Fix 0/1 values in y_pred so that log is not undefined
        y_pred = np.maximum(np.full(y_pred.shape, self.eps),
np.minimum(np.full(y_pred.shape, 1-self.eps), y_pred))


        likelihood = (y_true*np.log(y_pred)+(1-y_true)*np.log(1-y_pred))
        #self.likelihoods.append(-1*np.mean(likelihood))


        return np.mean(likelihood)

    def fit(self, X, y):


        num_examples = X.shape[0]
        num_features = X.shape[1]

        ### START CODE HERE

        # Initialize weights with appropriate shape
        self.weights = np.zeros((X.shape[1]))
        #print("Z",self.weights.shape)
        # print(X.shape)


        # Perform gradient ascent
        for i in range(self.max_iterations):
```

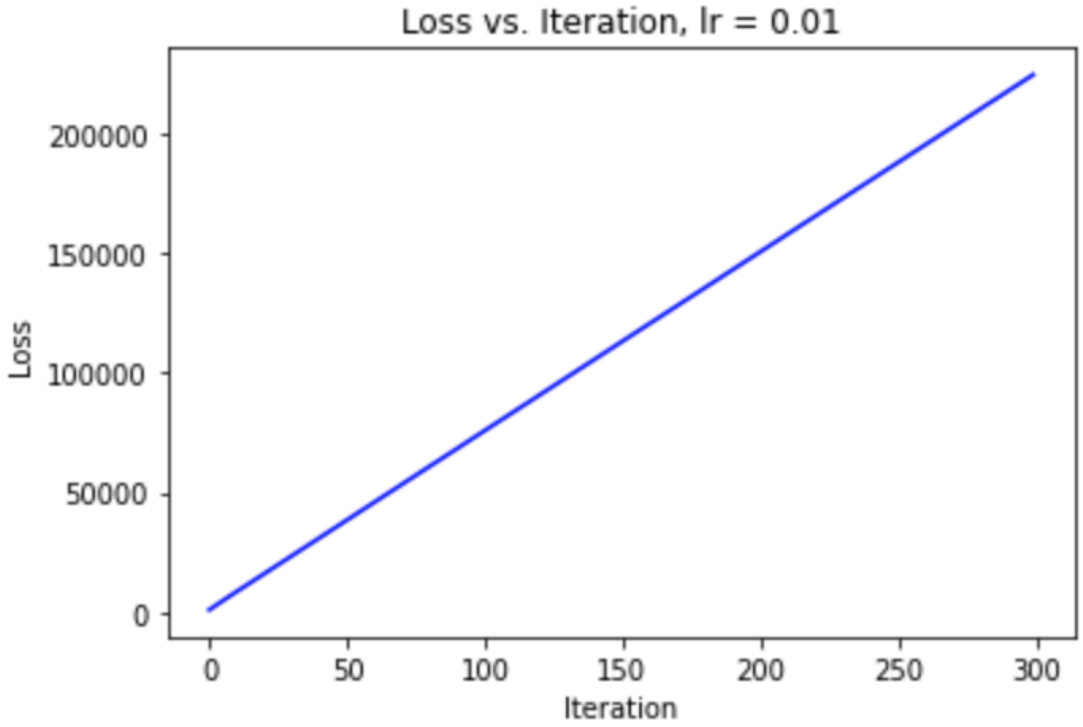
```
            z  = np.dot(X,self.weights)

            # Output probability value by appplying sigmoid on z
            y_pred = self.sigmoid(z)



            # Calculate the gradient values
            # This is just vectorized efficient way of implementing gradient.
            gradient = np.mean(np.dot(X.T, y-y_pred), axis=1)
            #print(gradient.shape)

            # Update the weights
            # It is gradient ASCENT not descent
            self.weights =  (self.weights) -
(self.learning_rate*self.shrinkage*self.weights) +
(self.learning_rate/X.shape[1])*gradient

            # Calculating log likelihood
            likelihood = self.log_likelihood(y,y_pred)

            self.likelihoods.append(likelihood)

        ### END CODE HERE

    def predict_proba(self,X):

        if self.weights is None:
            raise Exception("Fit the model before prediction")

        ### START CODE HERE

        z = np.dot(X,self.weights)
        probabilities = self.sigmoid(z)
        # probabilities.reshape(probabilities.shape[0],1)

        ### END CODE HERE

        return probabilities

    def predict(self, X, threshold=0.5):
```

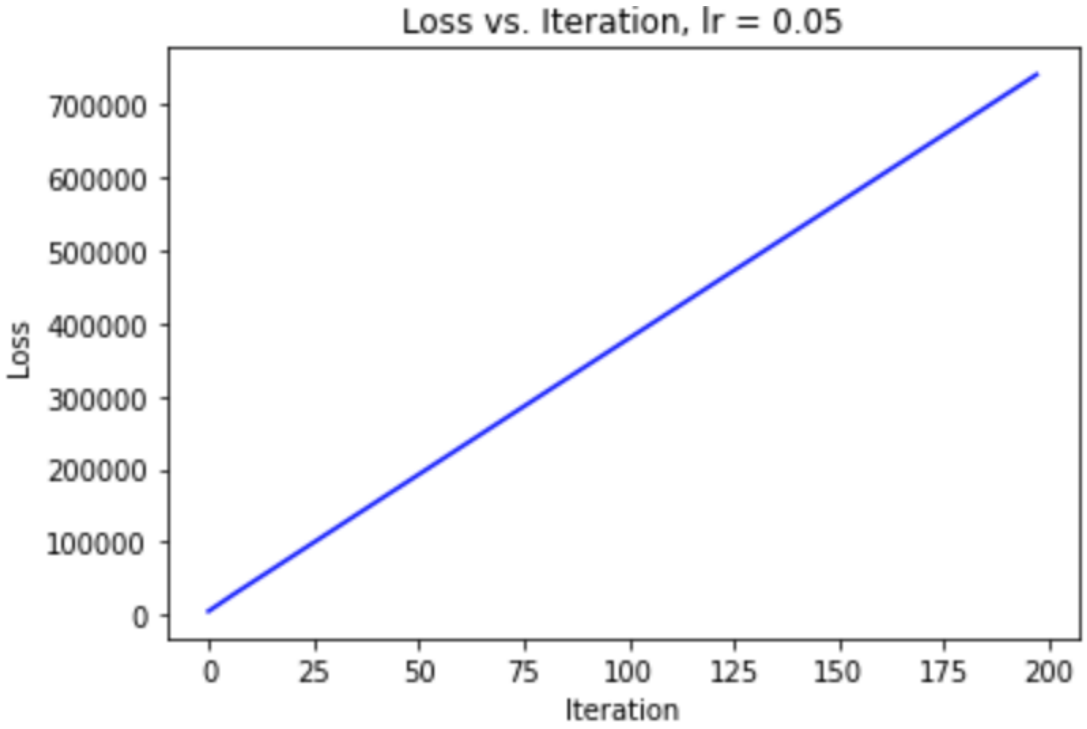
```
        # Thresholding probability to predict binary values

        binary_predictions = np.array(list(map(lambda x: 1 if x>threshold else
0, self.predict_proba(X))))

        return binary_predictions

clf = MyLogisticRegression(learning_rate=.01)

clf.fit(X,Y)

preds = clf.predict(X)

count_train = 0
for pred, y in zip(preds, Y):
    if pred == y:
        #print(pred, y)
        count_train+=1


preds = clf.predict(Xtest)

count_test = 0
for pred, y in zip(preds, Ytest):
    if pred == y:
        count_test+=1

print('Train Error: {:.2f}, Test Error
{:.2f}'.format(1-count_train/X.shape[0], 1-count_test/Xtest.shape[0]))

plt.title('Likelihood vs Iteration, eta={}'.format(clf.learning_rate))
plt.plot(np.abs(clf.likelihoods))
plt.show()
```

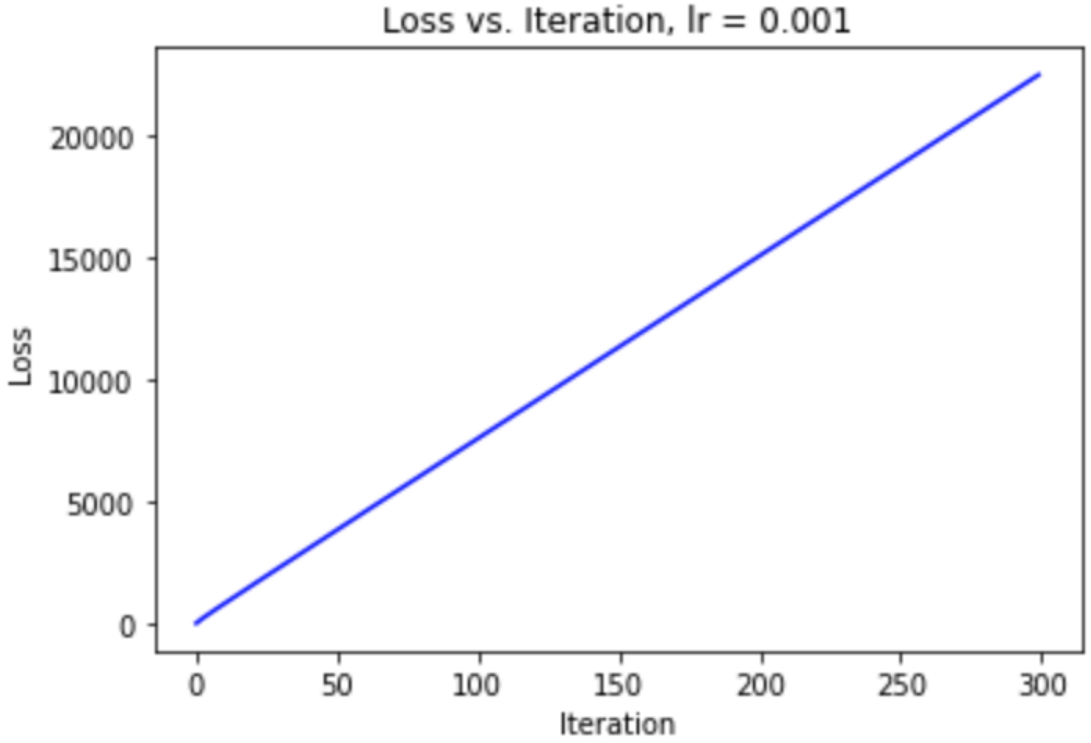
Part C.

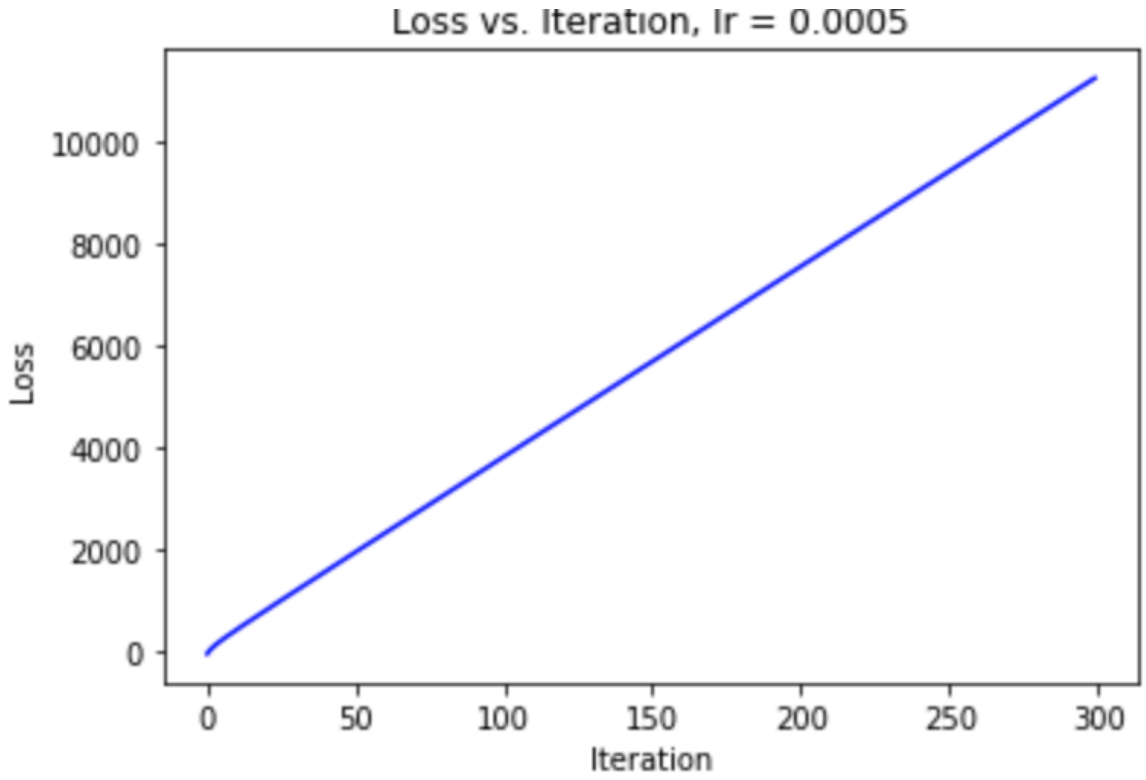
LearnRate	Train Error	Test Error
0.01	44.8%	47.5%
0.05	49.2%	49.8%

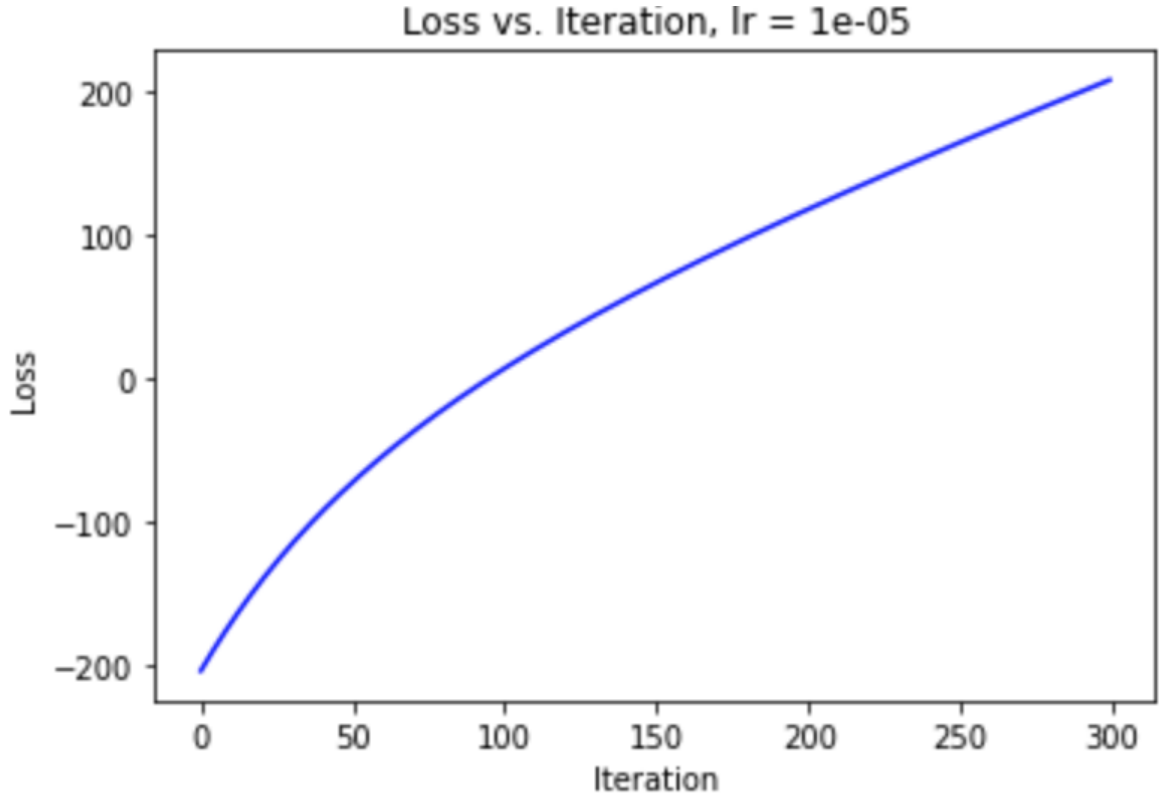
0.001	44.8%	47.8%
0.0005	44.8%	48.5%
0.00001	49.8%	49.8%




Loss vs. Iteration, lr = 0.05
Loss
700000
600000
500000
400000
300000
200000
100000
0
0
25
50
75
100
125
150
175
200
Iteration


Loss vs. Iteration, lr = 0.001
Loss
20000
15000
10000
5000
0
0
50
100
150
200
250
300
Iteration






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

```
import pandas as pd
import csv

class LiteLR:
    def __init__(self, learnRate, X, y, iterations=300, shrinkage=0.0001):
        self.iterations = iterations
        self.shrinkage = shrinkage
        self.learnRate = learnRate
        self.X = X
        self.y = y
        self.weights = np.zeros(X.shape[1])
        #display(self.weights)
    def L(self):
        loss = 0
        for j, obj in enumerate(self.X): #for each row in X
            loss = loss + (   (self.y[j] * np.dot(self.weights, obj))  -
np.log(1 + np.exp( np.dot(self.weights, obj)  ))  )
            #display(self.weights)
        return loss
    def dL(self):
        dloss = 0
        for j, obj in enumerate(self.X): #for each row in X
            #self.weights = map(float,self.weights)
            #obj = map(float,obj)
            dloss = dloss + (  obj * (self.y[j] -  (
(np.exp(np.dot(self.weights,obj)))/(1 + np.exp(np.dot(self.weights,obj)))
)    )  )
        return dloss
    def gradientAscent(self, plot=False): #trains model, returns loss
        losses = list()
        for i in range(self.iterations):
            #display(self.weights - (self.learnRate * self.shrinkage *
self.weights))
            self.weights = self.weights - (self.learnRate * self.shrinkage *
self.weights) + ((self.learnRate/self.X.shape[1])* self.dL())
            #display(self.weights)
            losses.append(self.L())

        if (plot == True):
            itrs = range(self.iterations)
            plt.plot(itrs, losses , c='b')
            plt.xlabel("Iteration")
            plt.ylabel("Loss")
```

```
            plt.title("Loss vs. Iteration, lr = {}".format(self.learnRate))
            plt.show()
        return self.L()
    def error(self, X, y):
        ret = 0
        for j, obj in enumerate(X):
            hyp = 1 / (1 + np.exp(-1 * np.dot(self.weights, obj)))
            pred = -1
            if (hyp > .5):
                pred = 1
            if (pred != y[j]):
                ret = ret + 1
        return ret / X.shape[0]

def read_csv(filename):
    with open(filename, newline='') as f_input:
        return [list(map(float, row)) for row in csv.reader(f_input)]


trainX = pd.read_csv('dexter_train.data')

trainX = np.array(trainX)
trainX = trainX.astype(str)
#trainX = trainX[:, 0:5000]
testX = pd.read_csv("dexter_valid.data")
testX = np.array(testX)
#testX = testX[:, 0:5000]
trainy = pd.read_csv("dexter_train.labels")
trainy = np.array(trainy)
testy = pd.read_csv("dexter_valid.labels")
testy = np.array(testy)




trainX_l = []
testX_l = []

for i in range(0,299):
    trainX_d = np.zeros(20000)
    #display(len(trainX_d))
    testX_d = np.zeros([20000])
    #trainX[i,0] = trainX[i,0].split(' ')
```

```
    d1 = trainX[i,0].split(' ')
    for j in range(len(d1)):
        q = d1[j].split(':')
        if len(q) > 1:
            q1 = int(q[0])
            q2 = int(q[1])

            trainX_d[int(q1)] = q2
    trainX_l.append(trainX_d)

    d2 = testX[i,0].split(' ')
    for j in range(len(d2)):
        q = d2[j].split(':')
        if len(q) > 1:
            q1 = int(q[0])
            q2 = int(q[1])

            testX_d[q1] = q2


    testX_l.append(testX_d)

trainX_l = np.array(trainX_l)
testX_l = np.array(testX_l)
trainX_l = np.array(trainX_l).astype(int)
testX_l = np.array(testX_l).astype(int)
trainy = np.array(trainy).astype(int)
testy = np.array(testy).astype(int)


for i in range(299):
    if trainX_l[i][:].all() != 0:
        trainX_l[i][:] = (trainX_l[i][:] - np.mean(trainX_l[i][:])) /
trainX_l[i][:]
        testX_l[i][:] = (testX_l[i][:] - np.mean(trainX_l[i][:])) /
trainX_l[i][:]


clf = LiteLR(.0005, trainX_l, trainy)
loss = clf.gradientAscent(plot=True)
print(loss)
```

```
print("Train error: {} Test error: {}".format(     clf.error(trainX_l, trainy)
, clf.error(testX_l,testy)   ))
```