Important Note for question !

- Please **do not** change the default variable names in this problem, as we will use them in different parts.
- The default variables are initially set to "None".
- You only need to modify code in the "TODO" part. We added a lot of "assertions" to check your code. **Do not** modify them.

In [1]:
```
# load packages
import numpy as np
import pandas as pd
import time
from sklearn.naive_bayes import GaussianNB
```

P1. Load data and plot

TODO

- Load train and test data, and split them into inputs(trainX, testX) and labels(trainY, testY)

In [2]:
```
# Use pandas to load q1_train.csv and q1_test.csv
# Each data point has 200 features(X), followed by 1 label(Y)

#### TODO ####
trainX = pd.read_csv("q1_train.csv").iloc[:,1:201]
trainY = pd.read_csv("q1_train.csv").iloc[:,201]
testX = pd.read_csv("q1_test.csv").iloc[:,1:201]
testY = pd.read_csv("q1_test.csv").iloc[:,201]
##############

assert(len(trainX.shape) == 2)
assert(len(trainY.shape) == 1)
assert(trainX.shape[1] == 200)
```

P2. Write your Gaussian NB solver

TODO

- Finish the myNBSolver() function.
  - Compute P(y == 0) and P(y == 1), saved in "py0" and "py1"
  - Compute mean/variance of trainX for both y = 0 and y = 1, saved in "mean0", "var0", "mean1" and "var1"
    - Each of them should have shape (N_train, M), where N_train is number of train samples and M is number of features.
  - Compute P(xi | y == 0) and P(xi | y == 1), compare and save **binary** prediction in "train_pred" and "test_pred"
  - Compute train accuracy and test accuracy, saved in "train_acc" and "test_acc".
  - Return train accuracy and test accuracy.

In [101…
```
def myNBSolver(trainX, trainY, testX, testY):
    N_train = trainX.shape[0]
    N_test = testX.shape[0]
    M = trainX.shape[1]

    #### TODO ####
    # Compute P(y == 0) and P(y == 1)

    py0 = sum(trainY == 0)/N_train
    py1 = sum(trainY == 1)/N_train

    ##############
    print("Total probablity is %.2f. Should be equal to 1." %(py0 + py1))

    #### TODO ####
    # Compute mean/var for each label

    mean0 = np.mean(trainX.loc[trainY==0,:])
    mean1 = np.mean(trainX.loc[trainY==1,:])
    var0 = np.var(trainX.loc[trainY==0,:])
    var1 = np.var(trainX.loc[trainY==1,:])

    ##############
    assert(mean0.shape[0] == M)

    #### TODO ####
    # Compute P(xi|y == 0) and P(xi|y == 1), compare and make prediction
    # This part may spend 5 - 10 minutes or even more if you use for loop, so
    # feel free to print something (like step number) to check the progress
    def P_xi_y(x,mean,var):
```

```
        return np.log(1/np.sqrt(2*np.pi*var))-((x-mean)**2/2*var)

    train_pred = np.empty((N_train))
    test_pred = np.empty((N_test))
    p_0 = p_1 = p_0_test = p_1_test = 0
    for i in range(N_train):
        for j in range(M):
            p0 = P_xi_y(trainX.iloc[i,j],mean0[j],var0[j])
            p1 = P_xi_y(trainX.iloc[i,j],mean1[j],var1[j])
            #print("p0",p0,"p1",p1)
            p_0 = p0 + p_0
            p_1 = p1 + p_1
        y0 = np.log(py0) + p_0
        y1 = np.log(py1) + p_1
        #print("y0",y0,"y1",y1)
        if y0 > y1:
            train_pred[i]=0
        else:
            train_pred[i]=1

    for i in range(N_test):
        for j in range(M):
            p0_test = P_xi_y(testX.iloc[i,j],mean0[j],var0[j])
            p1_test = P_xi_y(testX.iloc[i,j],mean1[j],var1[j])
            p_0_test = p0_test + p_0_test
            p_1_test = p1_test + p_1_test
        y0_test = np.log(py0) + p_0_test
        y1_test = np.log(py1) + p_1_test
        if y0_test > y1_test:
            test_pred[i]=0
        else:
            test_pred[i]=1

    ##############
    assert(train_pred[0] == 0 or train_pred[0] == 1)
    assert(test_pred[0] == 0 or test_pred[0] == 1)

    #### TODO ####
    # Compute train accuracy and test accuracy
    m = 0
    for a in range(len(train_pred)):
        if train_pred[a] == trainY[a]:
            m = m + 1
    n = 0
    for b in range(len(test_pred)):
        if test_pred[b] == testY[b]:
            n = n + 1

    train_acc = m/N_train
    test_acc = n/N_test

    ##############

    return train_acc, test_acc
```

In [102…

```
# driver to test your NB solver
train_acc, test_acc = myNBSolver(trainX, trainY, testX, testY)
print("Train accuracy is %.2f" %(train_acc * 100))
print("Test accuracy is %.2f" %(test_acc * 100))
```

```
Total probablity is 1.00. Should be equal to 1.
Train accuracy is 90.04
Test accuracy is 89.77
```

P3. Test your result using sklearn

TODO

- Finish the skNBSolver() function.
    - fit model, make prediction and return accuracy for train and test sets.

In [103…
```
def skNBSolver(trainX, trainY, testX, testY):

    #### TODO ####
    # fit model
    # make prediction
    # compute accuracy
    clf = GaussianNB()
    clf.fit(trainX, trainY)
    sk_train_acc = clf.score(trainX, trainY)
    sk_test_acc = clf.score(testX, testY)

    ##############
    return sk_train_acc, sk_test_acc
```

In [104…
```
# driver to test skNBSolver
sk_train_acc, sk_test_acc = skNBSolver(trainX, trainY, testX, testY)
print("Train accuracy is %.2f" %(sk_train_acc * 100))
print("Test accuracy is %.2f" %(sk_test_acc * 100))
```

```
Train accuracy is 92.22
Test accuracy is 92.05
```

In [ ]:
```
```

Note for question2

- Please follow the template to complete q1
- You may create new cells to report your results and observations

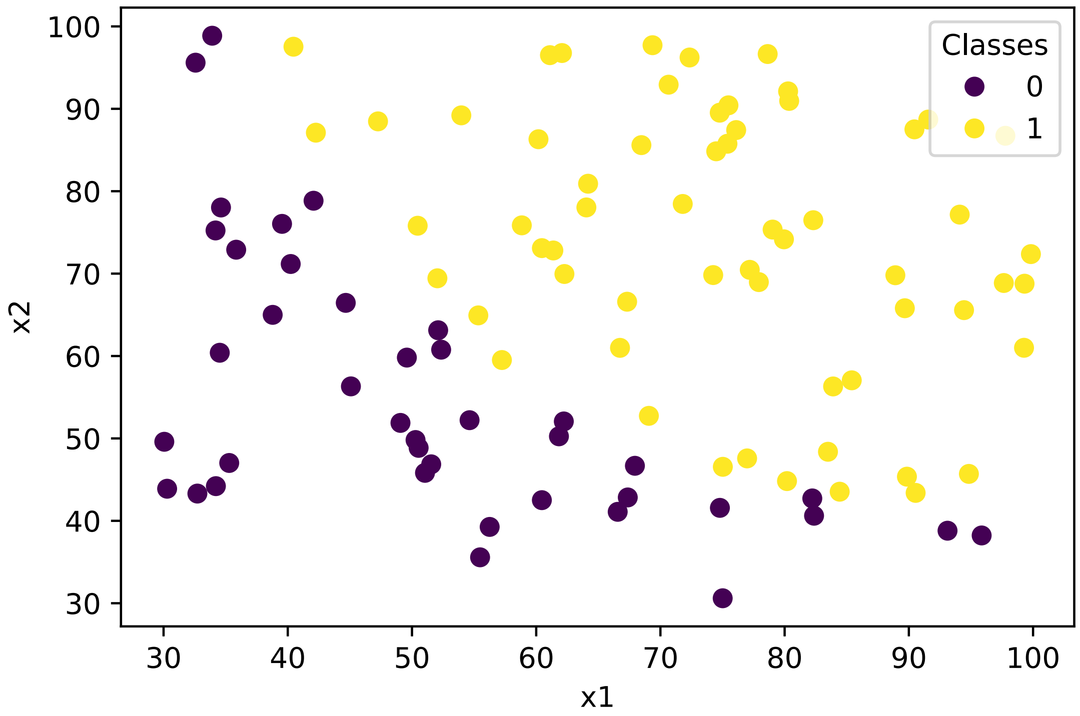
In [2]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
```

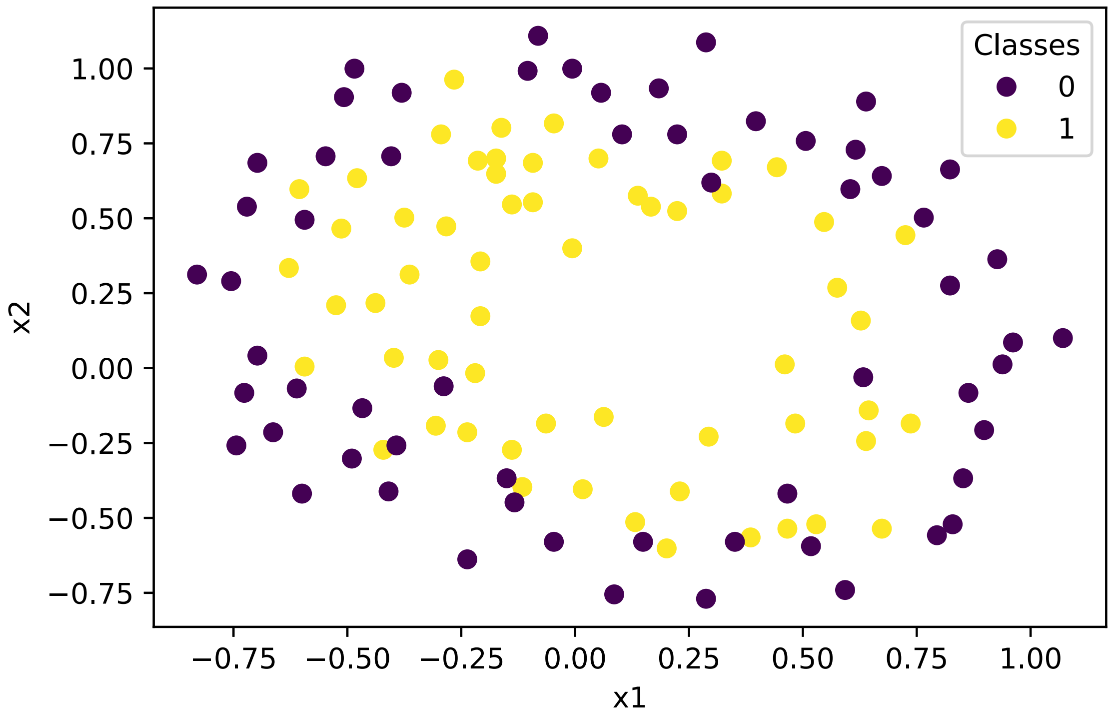
A. Load data and plot

TODO

- load data
- plot the points of different labels with different color

In [126…
```
# Load dataset
T1 = np.loadtxt('ex2data1.txt',delimiter=',')
T2 = np.loadtxt('ex2data2.txt',delimiter=',')

# Plot points
x1=T1[:,0]
y1=T1[:,1]
classes1=T1[:,2]
plt.figure(1)
scatter = plt.scatter(x1,y1,c=classes1)
plt.xlabel("x1")
plt.ylabel("x2")
legend1 = plt.legend(*scatter.legend_elements(),loc="upper right",
                        title="Classes")

x2=T2[:,0]
y2=T2[:,1]
classes2=T2[:,2]
plt.figure(2)
scatter = plt.scatter(x2,y2,c=classes2)
plt.xlabel("x1")
plt.ylabel("x2")
legend2 = plt.legend(*scatter.legend_elements(),loc="upper right",
                        title="Classes")
```





B. sigmoid function

TODO

- name the sigmoid function **sigmoid()**

In [128…

```
#Define sigmoid function
def sigmoid(h):
    return (1+np.exp(-h))**(-1)
```

C. loss function, gradient function

TODO

- Define loss function and name it **loss()**
- Define Gradient Function and name it **gradient()**

In [127…

```
#Define loss function
def loss(X, Y, w):
    J = np.mean(-Y*np.log(sigmoid((X@w))+ 1e-8)
                -(1-Y)*np.log(1-sigmoid((X@w))+ 1e-8))
    return J


#Define gradient function
def gradient(X, Y, w):
    g = np.mean((sigmoid((X@w))-Y)*X,axis=0)
    return g
```

D. prediction function, gradient descent and plot meshgrids

TODO

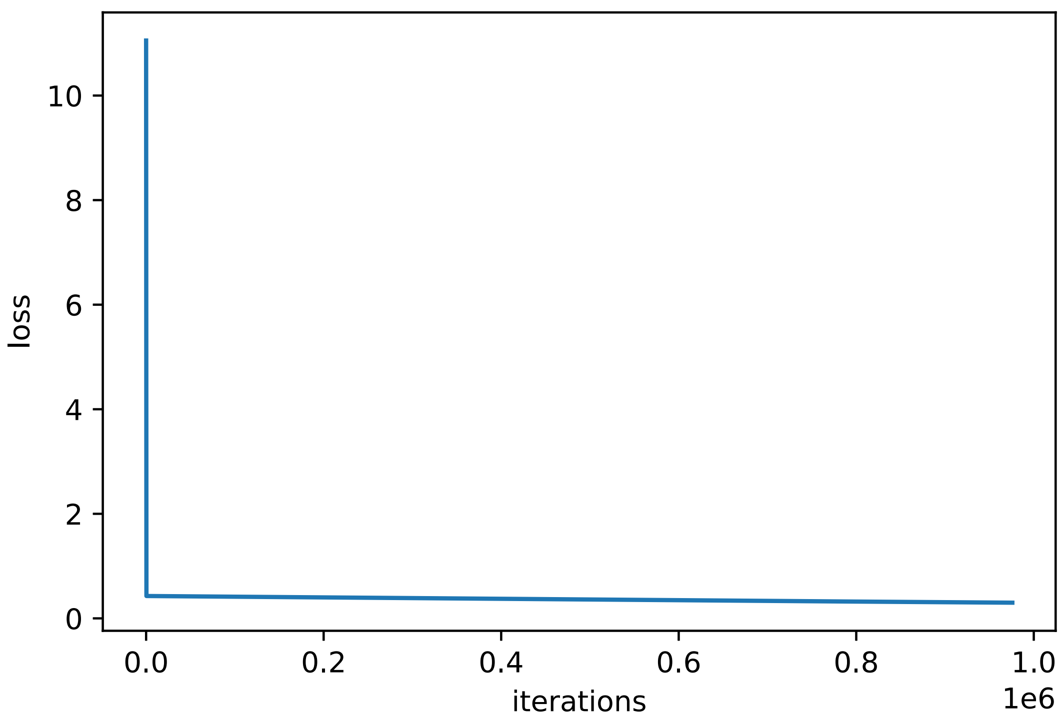
- Define a prediction function and name it **predict()**
- Using all above functions implement gradient descent with appropriate initialization, learning rates & # of initialization
- Use contourf/meshgrids or any other command to visualize the boundary conditions

In [129…

```
#Define prediction function
def predict(w,X):
    p = sigmoid(X@w)
    v = np.empty((len(X))).reshape(-1,1)
    v[p >= 0.5] = 1
    v[p < 0.5] = 0
    result = np.hstack((p,v))
    return result


#learned w
w = np.array([[-100.],[0.],[0.]])
one = np.ones((len(T1))).reshape(-1,1)
X = np.hstack((one, T1[:,0:2]))
Y = T1[:,2].reshape(-1,1)
J = []
iter = 0
while True:
    m = loss(X,Y,w)
    J.append(m)
    G = gradient(X, Y, w).reshape(-1,1)
    last_w = w
    w = w-0.01*G
    iter = iter +1
    if abs(loss(X,Y,w)-loss(X,Y,last_w)) < 1e-4 and loss(X,Y,w) < 0.3:
        break
print("w",w)
print("loss",loss(X,Y,w))
iter_times = np.arange(0,iter,1)
plt.plot(iter_times,J)
plt.xlabel("iterations")
plt.ylabel("loss")
```

```
w [[-64.72200748]
 [  0.52544198]
 [  0.51879292]]
loss 0.2999998972376585
```

Out[129… Text(0, 0.5, 'loss')



In [130…

```
#Call prediction function and Plot meshgrid to visualize
result = predict(w,X)
print("predict matrix",result)
```

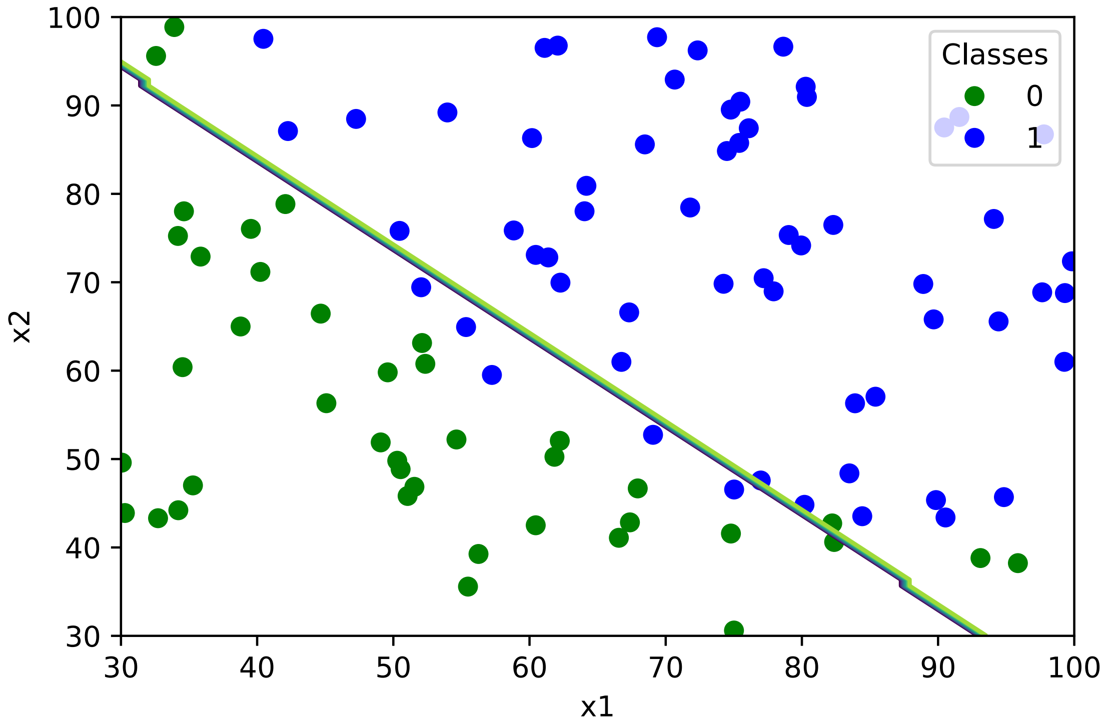
```
predict matrix [[2.35084987e-03 0.00000000e+00]
 [4.92998991e-12 0.00000000e+00]
 [3.14211605e-04 0.00000000e+00]
 [9.99991511e-01 1.00000000e+00]
 [9.99999875e-01 1.00000000e+00]
 [7.37928932e-06 0.00000000e+00]
 [9.99999974e-01 1.00000000e+00]
 [2.40678351e-01 0.00000000e+00]
```

```
[9.99999999e-01 1.00000000e+00]
[9.02679396e-01 1.00000000e+00]
[9.95842525e-01 1.00000000e+00]
[8.02786840e-05 0.00000000e+00]
[9.99999988e-01 1.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[1.09918886e-02 0.00000000e+00]
[9.99950663e-01 1.00000000e+00]
[2.55731850e-01 0.00000000e+00]
[8.13538556e-03 0.00000000e+00]
[9.99999999e-01 1.00000000e+00]
[6.00596247e-01 1.00000000e+00]
[8.26373092e-04 0.00000000e+00]
[9.99999934e-01 1.00000000e+00]
[2.69837234e-06 0.00000000e+00]
[4.56524470e-11 0.00000000e+00]
[9.99993886e-01 1.00000000e+00]
[9.86476023e-01 1.00000000e+00]
[6.60701864e-01 1.00000000e+00]
[9.87051141e-01 1.00000000e+00]
[2.10505031e-03 0.00000000e+00]
[2.43484342e-05 0.00000000e+00]
[9.95040431e-01 1.00000000e+00]
[9.99941783e-01 1.00000000e+00]
[1.00297117e-02 0.00000000e+00]
[2.05235039e-01 0.00000000e+00]
[1.28138650e-03 0.00000000e+00]
[1.32839632e-04 0.00000000e+00]
[9.87767895e-01 1.00000000e+00]
[9.99982855e-01 1.00000000e+00]
[2.07020960e-02 0.00000000e+00]
[4.40265992e-04 0.00000000e+00]
[9.99811467e-01 1.00000000e+00]
[1.62832083e-06 0.00000000e+00]
[9.99999994e-01 1.00000000e+00]
[4.08651369e-01 0.00000000e+00]
[7.32440787e-07 0.00000000e+00]
[6.56949274e-03 0.00000000e+00]
[9.99995847e-01 1.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[9.99999986e-01 1.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[9.99999857e-01 1.00000000e+00]
[9.99999995e-01 1.00000000e+00]
[9.95347902e-01 1.00000000e+00]
[2.38670050e-07 0.00000000e+00]
[3.87434824e-06 0.00000000e+00]
[4.81258229e-04 0.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[8.79877547e-01 1.00000000e+00]
[9.99972898e-01 1.00000000e+00]
[9.99998882e-01 1.00000000e+00]
[9.99999996e-01 1.00000000e+00]
[3.45063493e-10 0.00000000e+00]
[3.75383873e-07 0.00000000e+00]
[8.40712509e-11 0.00000000e+00]
[1.13771708e-03 0.00000000e+00]
[2.18211006e-04 0.00000000e+00]
[9.99205222e-01 1.00000000e+00]
[6.01902242e-06 0.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[8.79893700e-01 1.00000000e+00]
[1.30712959e-11 0.00000000e+00]
[9.99918042e-01 1.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[9.93079147e-01 1.00000000e+00]
[9.96128858e-01 1.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[9.97559908e-01 1.00000000e+00]
[7.54071523e-01 1.00000000e+00]
[1.84015279e-05 0.00000000e+00]
[6.56131700e-01 1.00000000e+00]
[9.99999988e-01 1.00000000e+00]
[9.99851371e-01 1.00000000e+00]
[9.94494251e-01 1.00000000e+00]
[2.25150238e-02 0.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[9.99999842e-01 1.00000000e+00]
[1.53205179e-01 0.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[3.37029354e-03 0.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[1.00000000e+00 1.00000000e+00]
[3.68827513e-08 0.00000000e+00]
[9.99999990e-01 1.00000000e+00]
[9.97572123e-01 1.00000000e+00]
[9.85881541e-01 1.00000000e+00]
[9.35410022e-01 1.00000000e+00]
[1.00000000e+00 1.00000000e+00]
```

```
[1.23704331e-01 0.00000000e+00]
 [9.99999999e-01 1.00000000e+00]]
```

In [131…

```
#Vistualize
x1 = np.linspace(30,100,100)
x2 = np.linspace(30,100,100)
x1_new, x2_new = np.meshgrid(x1,x2)
one_new = np.ones((len(x1_new.ravel()))).reshape(-1,1)
X_new = np.c_[one_new, x1_new.ravel(),x2_new.ravel()]
result_new = predict(w,X_new)
x0=T1[:,0]
y0=T1[:,1]
colors = ['green','blue']
plt.contour(x1_new,x2_new,result_new[:,1].reshape(x1_new.shape))
#plt.scatter(x1_new.ravel(),x2_new.ravel(),c=result_new[:,1])
scatter = plt.scatter(x0,y0,c=classes1,cmap= mpl.colors.ListedColormap(colors))
plt.xlabel("x1")
plt.ylabel("x2")
legend1 = plt.legend(*scatter.legend_elements(),loc="upper right",
                        title="Classes")
```



E. Feature mapping, regularized Cost function, gradient function and gradient descent

TODO

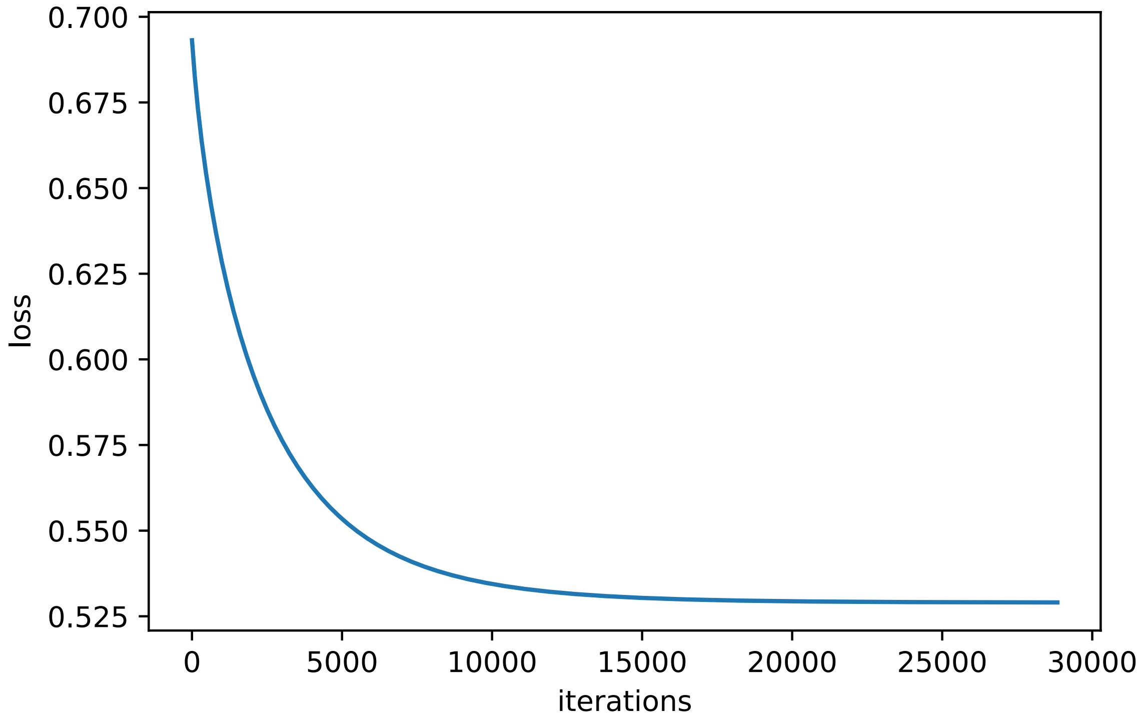
- implement function **map_feature()** to transform data from original space to the 28D space specified in the write-up
- Create a regularized loss function & gradient function and name it **loss_reg()** and **gradient_reg()**
- Using both these functions implement gradient descent with appropriate initialization, learning rates & # of initialization
- Use contourf/meshgrids or any other command to visualize the boundary conditions

In [132…

```
# Transform points to 28D space
def map_feature(x1,x2,m):
    A = np.empty(shape=(len(x1),0))
    for i in range(m+1):
        for j in range(i+1):
            x = (x1**j)*(x2**(i-j))
            A = np.hstack((A,x))
    return A


#Define cost function
def loss_reg(X, Y, w, lamda):
    w_new = w.copy()
    w_new[0,0] = 0
    J = np.mean(-Y*np.log(sigmoid(X@w)+1e-8)-(1-Y)*np.log(1-sigmoid(X@w)+1e-8))
    +((w_new.T)@w_new*(lamda/(2*len(X))))[0][0]
    return J


#Define gradient function
def gradient_reg(X, Y, w, lamda):
    w_new = w.copy()
    w_new[0,0] = 0
    g = np.mean((sigmoid((X@w))-Y)*X,axis=0).reshape(-1,1)+lamda * w_new/len(X)
    return g


#Define prediction function which implements regularized logistic regression
#learned w
def w_learn(w,X,Y,max_iter,tol,lamda,learning_rate):
    J = []
    iter = 0
    while iter<max_iter:
        m = loss_reg(X,Y,w,lamda)
        J.append(m)
        G = gradient_reg(X, Y, w,lamda).reshape(-1,1)
        last_w = w
        w = w-learning_rate*G
        iter = iter +1
        if abs(loss_reg(X,Y,w,lamda)-loss_reg(X,Y,last_w,lamda)) < tol:
            break
    return w,iter,J

#lambda = 1
w= np.zeros((28,1))
x1 = T2[:,0].reshape(-1,1)
x2 = T2[:,1].reshape(-1,1)
Y = T2[:,2].reshape(-1,1)
X = map_feature(x1,x2,6)
w,iter,J  = w_learn(w,X,Y,1e6,1e-8,1,0.01)
print("w",w)
print("loss",loss_reg(X,Y,w,1))
iter_times = np.arange(0,iter,1)
plt.plot(iter_times,J)
plt.xlabel("iterations")
plt.ylabel("loss")
```

```
w [[ 1.25007918]
 [ 1.16700997]
 [ 0.61229231]
 [-1.37466796]
 [-0.88693251]
 [-1.98702051]
 [-0.17608361]
 [-0.35638243]
 [-0.36004496]
 [ 0.11859625]
 [-1.18063384]
 [-0.26517307]
 [-0.60561199]
 [-0.06092944]
 [-1.44614744]
 [-0.47375078]
 [-0.2887387 ]
 [-0.27133961]
 [-0.05330495]
 [-0.20694506]
 [-0.24131083]
 [-0.93878943]
 [-0.13775362]
 [-0.32232059]
 [ 0.00944034]
 [-0.28990499]
 [ 0.01881184]
 [-1.03770964]]
loss 0.5290456351387117
```

Out[132… Text(0, 0.5, 'loss')



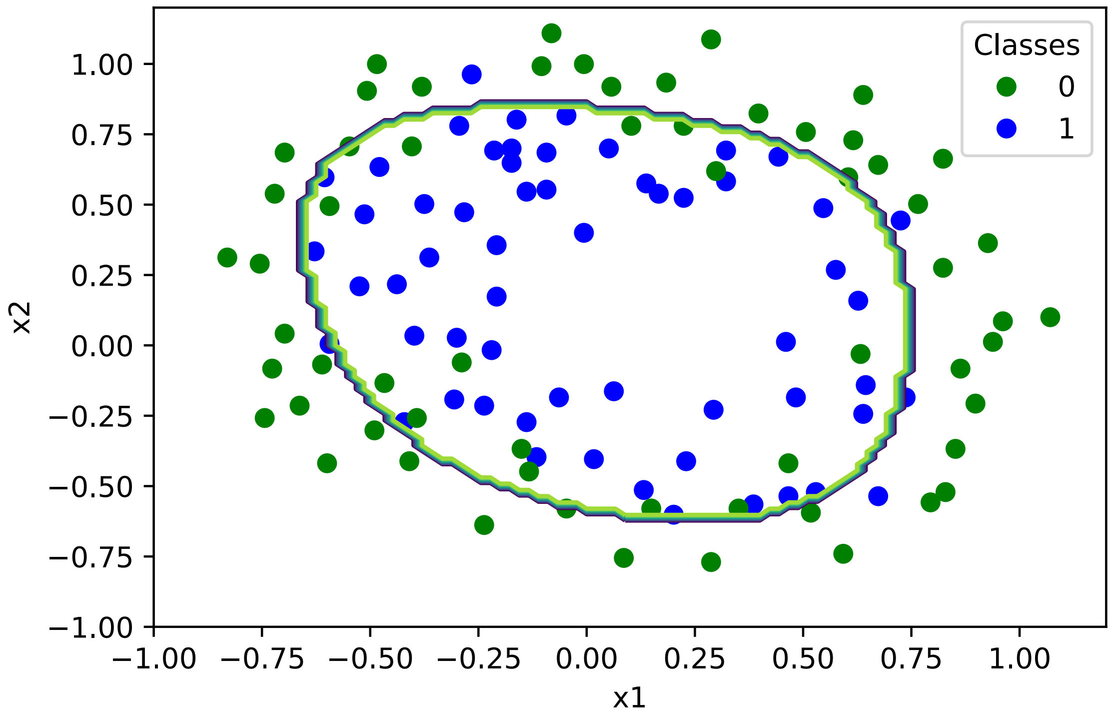
In [133…
```
##Call prediction function and Plot meshgrid to visualize
result2 = predict(w,X)
print("predict matrix",result2)
```

```
predict matrix [[0.69750133 1.        ]
 [0.71652821 1.        ]
 [0.69815464 1.        ]
 [0.7276679  1.        ]
 [0.64921396 1.        ]
 [0.62224356 1.        ]
 [0.66700788 1.        ]
 [0.6320653  1.        ]
 [0.6340684  1.        ]
 [0.58073212 1.        ]
 [0.53365981 1.        ]
 [0.51593079 1.        ]
 [0.57783758 1.        ]
 [0.47812562 0.        ]
 [0.62921892 1.        ]
 [0.71282963 1.        ]
 [0.77439891 1.        ]
 [0.55984888 1.        ]
 [0.69878235 1.        ]
 [0.62372569 1.        ]
 [0.52150304 1.        ]
 [0.53178757 1.        ]
 [0.49026906 0.        ]
 [0.5194833  1.        ]
 [0.60354071 1.        ]
 [0.51411079 1.        ]
 [0.53368128 1.        ]
 [0.39409606 0.        ]
 [0.77827278 1.        ]
 [0.5886177  1.        ]
```

```
 [0.2403284  0.        ]
 [0.58414054 1.        ]
 [0.73332811 1.        ]
 [0.76438728 1.        ]
 [0.73834355 1.        ]
 [0.71282564 1.        ]
 [0.65367432 1.        ]
 [0.71709532 1.        ]
 [0.74319058 1.        ]
 [0.65140068 1.        ]
 [0.72507712 1.        ]
 [0.69451971 1.        ]
 [0.60028004 1.        ]
 [0.74093544 1.        ]
 [0.64283276 1.        ]
 [0.67764245 1.        ]
 [0.42148704 0.        ]
 [0.76915895 1.        ]
 [0.56196774 1.        ]
 [0.61800261 1.        ]
 [0.76451608 1.        ]
 [0.80932233 1.        ]
 [0.7803684  1.        ]
 [0.78782595 1.        ]
 [0.77257389 1.        ]
 [0.68740369 1.        ]
 [0.72956359 1.        ]
 [0.66229683 1.        ]
 [0.23799616 0.        ]
 [0.54459857 1.        ]
 [0.69758534 1.        ]
 [0.36856734 0.        ]
 [0.28835218 0.        ]
 [0.47606583 0.        ]
 [0.3083455  0.        ]
 [0.10574042 0.        ]
 [0.31484739 0.        ]
 [0.10110443 0.        ]
 [0.13766613 0.        ]
 [0.27074025 0.        ]
 [0.19388935 0.        ]
 [0.23519971 0.        ]
 [0.2087152  0.        ]
 [0.23541221 0.        ]
 [0.27570082 0.        ]
 [0.46333321 0.        ]
 [0.60923665 1.        ]
 [0.52750164 1.        ]
 [0.32865761 0.        ]
 [0.33211651 0.        ]
 [0.529559   1.        ]
 [0.56426766 1.        ]
 [0.44055096 0.        ]
 [0.5482156  1.        ]
 [0.19339534 0.        ]
 [0.3674551  0.        ]
 [0.33606584 0.        ]
 [0.3408364  0.        ]
 [0.62005065 1.        ]
 [0.30464531 0.        ]
 [0.2646558  0.        ]
 [0.51047745 1.        ]
 [0.59047893 1.        ]
 [0.31869617 0.        ]
 [0.1804288  0.        ]
 [0.02689858 0.        ]
 [0.01662624 0.        ]
 [0.34681036 0.        ]
 [0.06515862 0.        ]
 [0.10151949 0.        ]
 [0.32671282 0.        ]
 [0.01604602 0.        ]
 [0.48886585 0.        ]
 [0.35698797 0.        ]
 [0.60902761 1.        ]
 [0.44573065 0.        ]
 [0.55460258 1.        ]
 [0.69115378 1.        ]
 [0.43999788 0.        ]
 [0.30901285 0.        ]
 [0.2694946  0.        ]
 [0.28176519 0.        ]
 [0.21330079 0.        ]
 [0.37941101 0.        ]
 [0.5674137  1.        ]
 [0.11665882 0.        ]
 [0.15344892 0.        ]
 [0.63065781 1.        ]]
```

In [134…

```
#Vistualize lambda = 1
x1 = np.linspace(-1,1.2,100)
x2 = np.linspace(-1,1.2,100)
x1_new, x2_new = np.meshgrid(x1,x2)
X_new = map_feature((x1_new.ravel()).reshape(-1,1),
                        (x2_new.ravel()).reshape(-1,1),6)
result_new = predict(w,X_new)
plt.contour(x1_new,x2_new,result_new[:,1].reshape(x1_new.shape))
#plt.scatter(x1_new.ravel(),x2_new.ravel(),c=result_new[:,1])
x0=T2[:,0]
y0=T2[:,1]
colors = ['green','blue']
scatter = plt.scatter(x0,y0,c=classes2,cmap= mpl.colors.ListedColormap(colors))
plt.xlabel("x1")
plt.ylabel("x2")
legend1 = plt.legend(*scatter.legend_elements(),loc="upper right",
                        title="Classes")
```



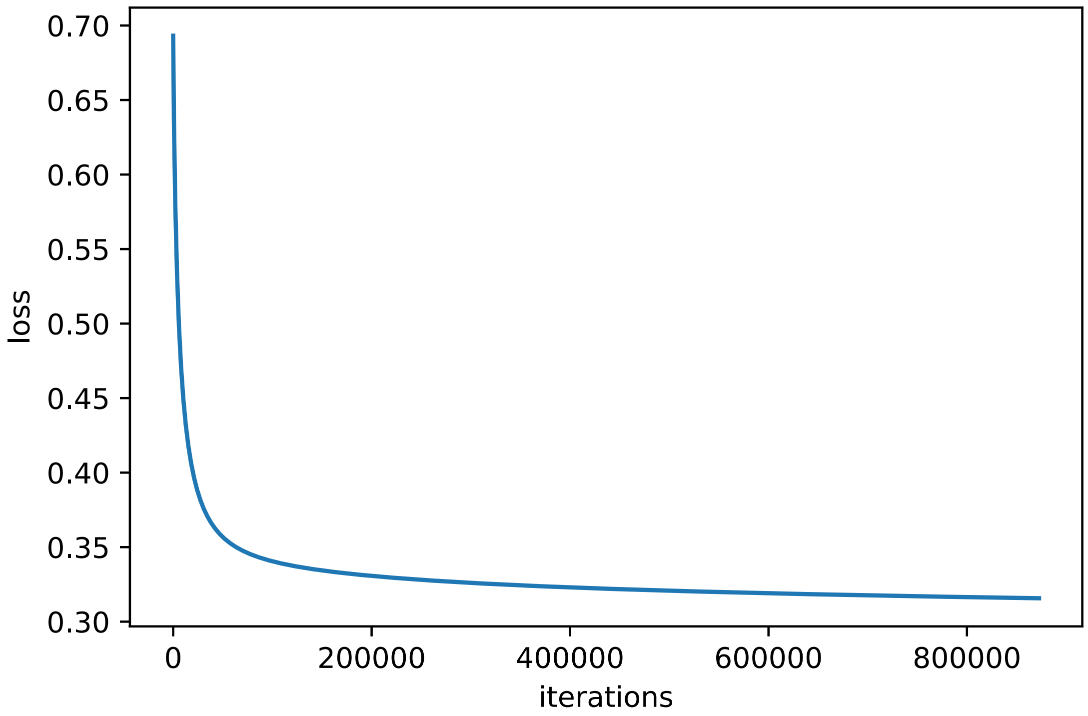
F. Tune the strength of regularization

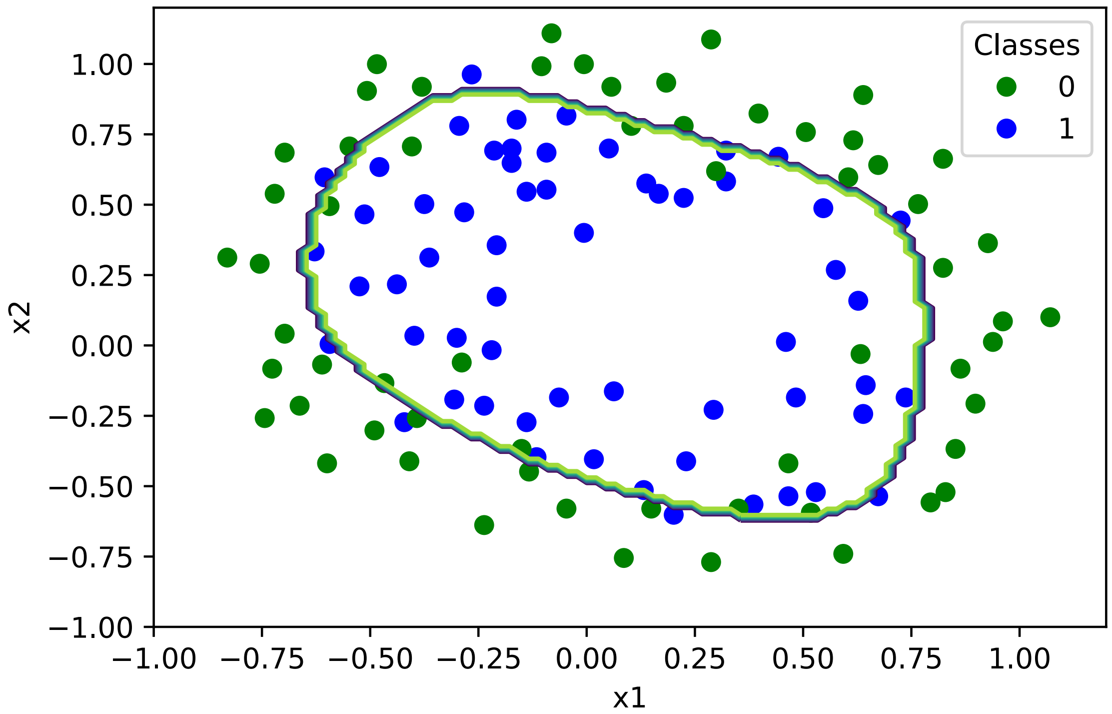
TODO

- tweak the hyper-parameter λ to be $[0, 1, 100]$
- draw the decision boundaries

In [136…

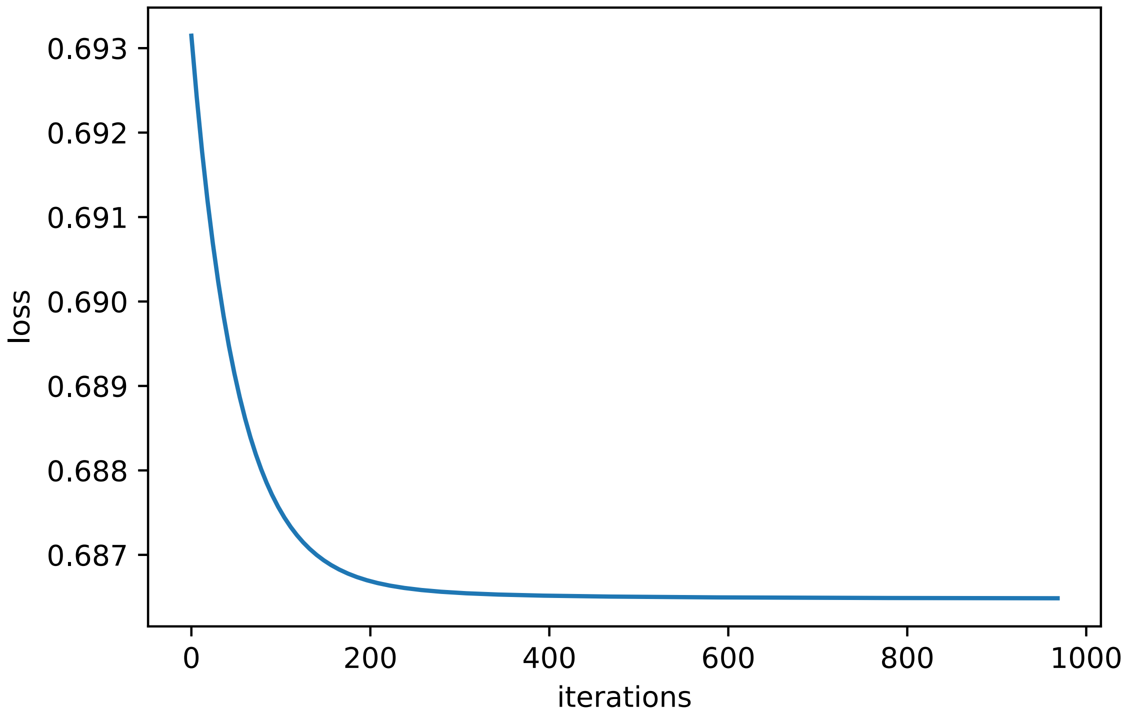
```
# lambda = 0.0001
w = np.zeros((28,1))
x1 = T2[:,0].reshape(-1,1)
x2 = T2[:,1].reshape(-1,1)
Y = T2[:,2].reshape(-1,1)
X = map_feature(x1,x2,6)
w,iter,J  = w_learn(w,X,Y,1e6,1e-8,0.0001,0.01)
iter_times = np.arange(0,iter,1)
plt.figure(1)
plt.plot(iter_times,J)
plt.xlabel("iterations")
plt.ylabel("loss")


#Vistualize lambda = 0.0001
plt.figure(2)
x1 = np.linspace(-1,1.2,100)
x2 = np.linspace(-1,1.2,100)
x1_new, x2_new = np.meshgrid(x1,x2)
X_new = map_feature((x1_new.ravel()).reshape(-1,1),
                        (x2_new.ravel()).reshape(-1,1),6)
result_new = predict(w,X_new)
plt.contour(x1_new,x2_new,result_new[:,1].reshape(x1_new.shape))
#plt.scatter(x1_new.ravel(),x2_new.ravel(),c=result_new[:,1])
x0=T2[:,0]
y0=T2[:,1]
colors = ['green','blue']
scatter = plt.scatter(x0,y0,c=classes2,cmap= mpl.colors.ListedColormap(colors))
plt.xlabel("x1")
plt.ylabel("x2")
legend1 = plt.legend(*scatter.legend_elements(),loc="upper right",
                        title="Classes")
```

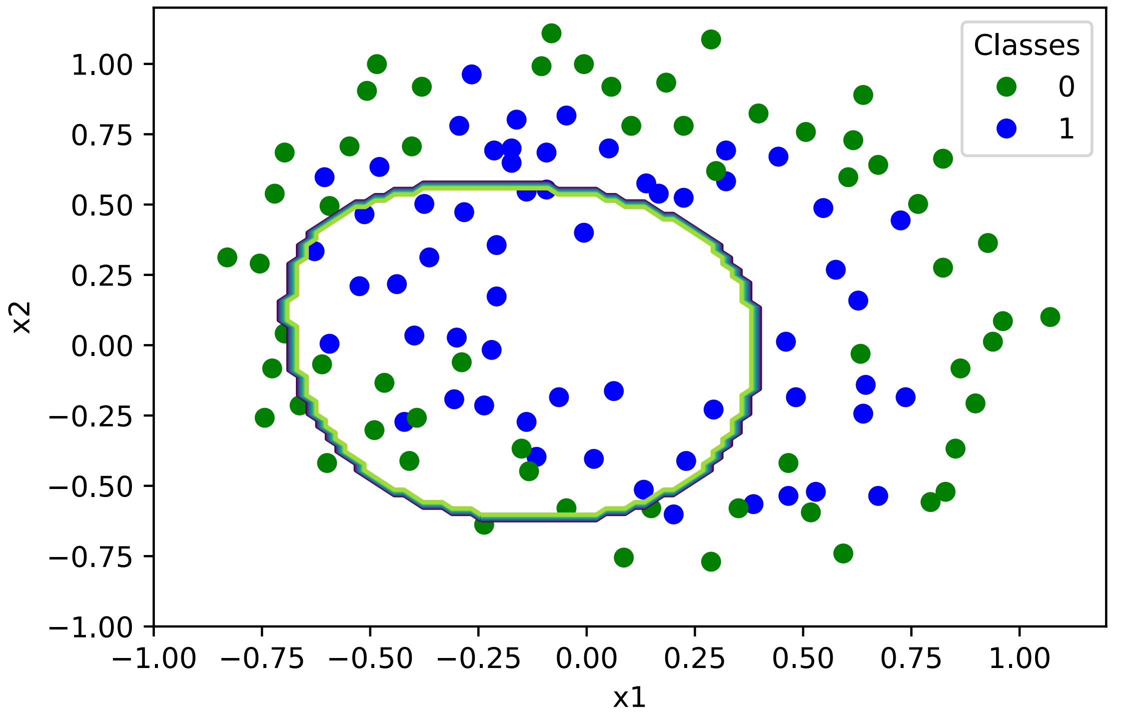




In [137…

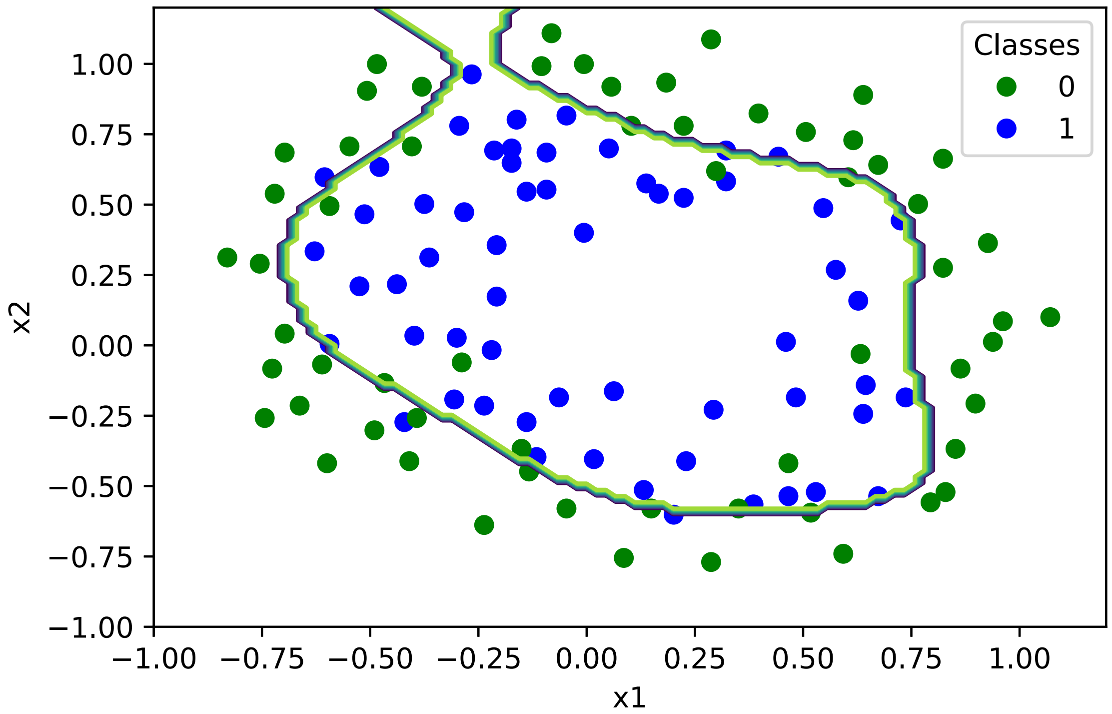
```
# lambda = 100
w = np.zeros((28,1))
x1 = T2[:,0].reshape(-1,1)
x2 = T2[:,1].reshape(-1,1)
Y = T2[:,2].reshape(-1,1)
X = map_feature(x1,x2,6)
w,iter,J  = w_learn(w,X,Y,1e6,1e-8,100,0.01)
iter_times = np.arange(0,iter,1)
plt.figure(1)
plt.plot(iter_times,J)
plt.xlabel("iterations")
plt.ylabel("loss")

#Vistualize lambda = 100
plt.figure(2)
x1 = np.linspace(-1,1.2,100)
x2 = np.linspace(-1,1.2,100)
x1_new, x2_new = np.meshgrid(x1,x2)
X_new = map_feature((x1_new.ravel()).reshape(-1,1),
                        (x2_new.ravel()).reshape(-1,1),6)
result_new = predict(w,X_new)
plt.contour(x1_new,x2_new,result_new[:,1].reshape(x1_new.shape))
#plt.scatter(x1_new.ravel(),x2_new.ravel(),c=result_new[:,1])
x0=T2[:,0]
y0=T2[:,1]
colors = ['green','blue']
scatter = plt.scatter(x0,y0,c=classes2,cmap= mpl.colors.ListedColormap(colors))
plt.xlabel("x1")
plt.ylabel("x2")
legend1 = plt.legend(*scatter.legend_elements(),loc="upper right",
                        title="Classes")
```





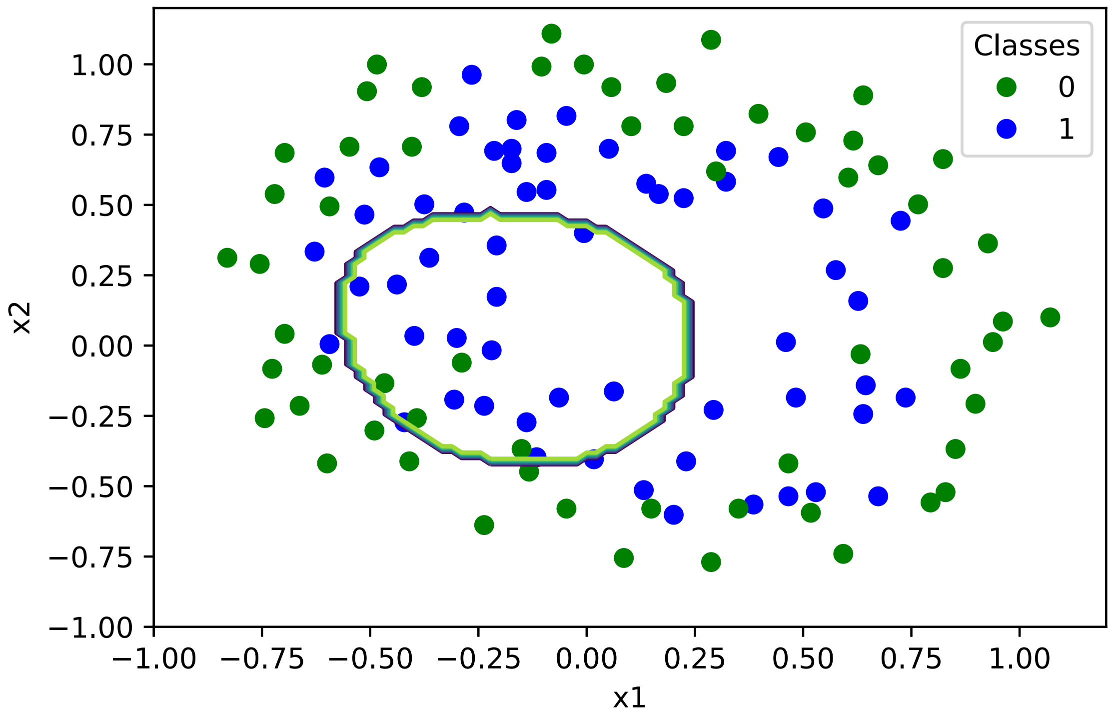
In [140…

```
#Verify models
#lambda = 0.0001
from sklearn import linear_model
x1 = T2[:,0].reshape(-1,1)
x2 = T2[:,1].reshape(-1,1)
X = map_feature(x1,x2,6)
Y = T2[:,2]
clf = linear_model.LogisticRegression(penalty = "l2", solver = "liblinear",
                    tol = 1e-10, max_iter = int(1e6),C=10000)
clf.fit(X,Y)
pred_test= clf.predict(X_new)
plt.contour(x1_new,x2_new,pred_test.reshape(x1_new.shape))
#plt.scatter(x1_new.ravel(),x2_new.ravel(),c=pred_test)
x0=T2[:,0]
y0=T2[:,1]
colors = ['green','blue']
scatter = plt.scatter(x0,y0,c=classes2,cmap= mpl.colors.ListedColormap(colors))
plt.xlabel("x1")
plt.ylabel("x2")
legend1 = plt.legend(*scatter.legend_elements(),loc="upper right",
                        title="Classes")
```



In [141…

```
#Verify models
#lambda = 100
clf = linear_model.LogisticRegression(penalty = "l2", solver = "liblinear",
                        tol = 1e-10, max_iter = int(1e6),C=0.01)
clf.fit(X,Y)
pred_test= clf.predict(X_new)
plt.contour(x1_new,x2_new,pred_test.reshape(x1_new.shape))
x0=T2[:,0]
y0=T2[:,1]
colors = ['green','blue']
scatter = plt.scatter(x0,y0,c=classes2,cmap= mpl.colors.ListedColormap(colors))
plt.xlabel("x1")
plt.ylabel("x2")
legend1 = plt.legend(*scatter.legend_elements(),loc="upper right",
                        title="Classes")
```



In [ ]: