Important Note for question4 !

- 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.

```
# 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)

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




#### TODO ####
trainX = q1_train[:,:-1]
trainY = q1_train[:,-1]
testX = q1_test[:,:-1]
testY = q1_test[:,-1]
##############

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.

```
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)
    y0 = np.argwhere(trainY == 0)
    y1 = np.argwhere(trainY == 1)
    x0 = trainX[y0]

    x1 = trainX[y1]
    x0 = x0.squeeze()
    x1 = x1.squeeze()

    py0 = len(y0)/N_train
    py1 = len(y1)/N_train

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

    #### TODO ####
    # Compute mean/var for each label
    mean0 = np.mean(x0,axis=0)
    # print(mean0.shape)
```

```
    mean1 = np.mean(x1,axis=0)
    var0 = np.mean((x0-mean0)**2,axis=0)
    var1 = np.mean((x1-mean1)**2,axis=0)

    ##############
    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,
    # print something (like step number) to check the progress
    p_x_y0 = (2*np.pi*var0)**(-0.5)*np.exp(-((trainX-mean0)**2)/(2*var0))
    p_x_y1 = (2*np.pi*var1)**(-0.5)*np.exp(-((trainX-mean1)**2)/(2*var1))
    prod0 = py0* np.prod(p_x_y0,axis=1)
    prod1 = py1* np.prod(p_x_y1,axis=1)
    train_ans = np.ones(N_train)
    pos0 = np.argwhere(prod0>prod1)
    train_ans[pos0] = 0

    p_x_y0_test = (2*np.pi*var0)**(-0.5)*np.exp(-((testX-mean0)**2)/(2*var0
    p_x_y1_test = (2*np.pi*var1)**(-0.5)*np.exp(-((testX-mean1)**2)/(2*var1
    prod0_test = py0* np.prod(p_x_y0_test,axis=1)
    prod1_test = py1* np.prod(p_x_y1_test,axis=1)
    test_ans = np.ones(N_test)
    pos0_test = np.argwhere(prod0_test>prod1_test)
    test_ans[pos0_test] = 0


    train_pred = train_ans
    test_pred = test_ans

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

    train_acc = len(np.argwhere(train_pred==trainY))/N_train
    test_acc = len(np.argwhere(test_pred==testY))/N_test

    ##############()

    return train_acc, test_acc
```

```
# 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 92.22
Test accuracy is 92.05
```

P3. Test your result using sklearn

TODO

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

```
def skNBSolver(trainX, trainY, testX, testY):

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

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

```
# 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
```

```
# from google.colab import drive
# drive.mount('/content/drive')
```

```
# %cd /content/drive/MyDrive/24787/hw3/data_and_code
```

```
[WinError 3] The system cannot find the path specified: '/content/drive/M
yDrive/24787/hw3/data_and_code'
e:\drive\24787\hw3\data_and_code
```

Note for question1

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

```
# Import libraries
# load packages
import numpy as np
import pandas as pd
import time
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

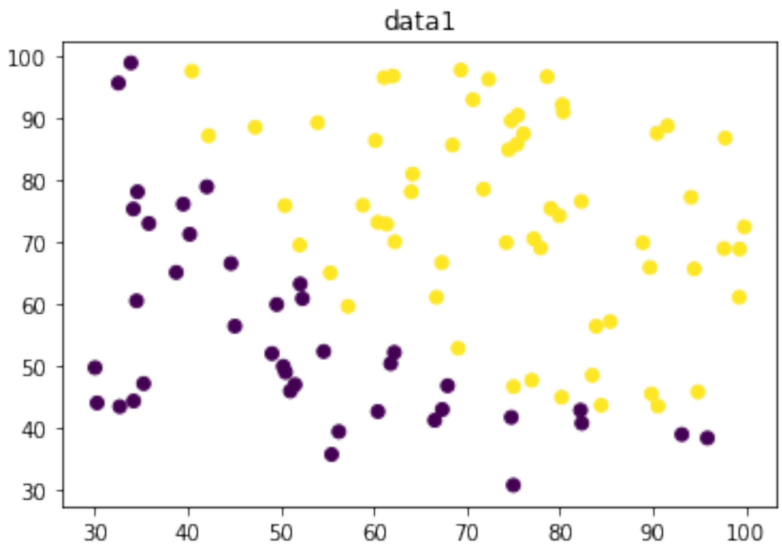
A. Load data and plot

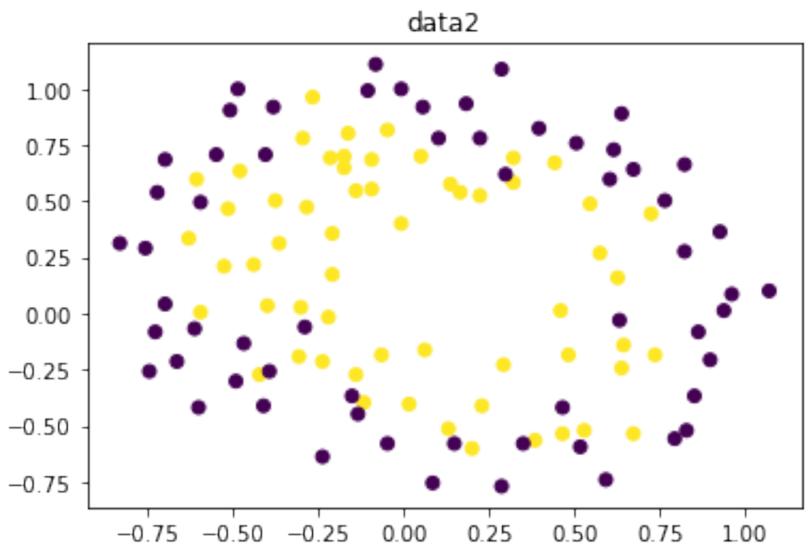
TODO

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

Q2 (a)

```
# Load dataset


data1 = np.loadtxt("ex2data1.txt",delimiter=',')
data2 = np.loadtxt("ex2data2.txt",delimiter=',')

def readdata(filename):
    data = np.loadtxt(filename,delimiter=',')
    X = data[:,:-1]
    Y = data[:,-1].reshape(-1,1)
    return X,Y
def plotscatter(X,Y,title=None):
    plt.scatter(X[:,0],X[:,1],c=Y)
    plt.title(title)
    plt.show()
    # return fig


X1,Y1 = readdata("ex2data1.txt")
X2,Y2 = readdata("ex2data2.txt")

plotscatter(X1,Y1,title="data1")
plotscatter(X2,Y2,title="data2")

# print(Y1.shape)
# X1 = data1[:,:-1]
# Y1 = data1[:,-1]
# X2 = data2[:,:-1]
# Y2 = data2[:,-1]
# print(X1.shape)



# fig =plt.figure()
# plt.scatter(X1[:,0],X1[:,1],c=Y1)
# fig =plt.figure()
# plt.scatter(X2[:,0],X2[:,1],c=Y2)
# Y1_reshape = Y1.reshape(-1,1)






# Plot points
```





B. sigmoid function

TODO

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

Q2 (b)

```
#Define sigmoid function

def sigmoid(mat):
    return (1+np.exp(-mat))**(-1)
```

C. loss function, gradient function

TODO

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

Q2 (c)

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

#Define gradient function
def gradient(X,w,Y):
    return np.mean((sigmoid(X @ w)-Y)*X,axis= 0).reshape(-1,1)
```

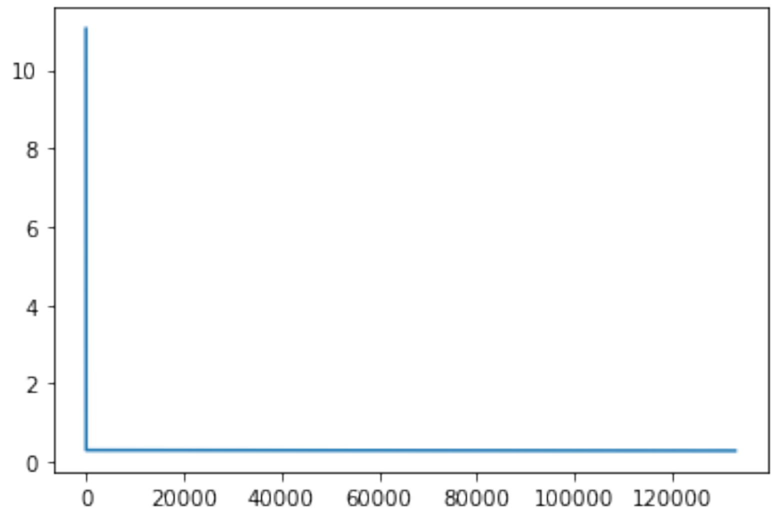
D. prediction function, gradient descent and plot meshgrids

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

Q2 (d)

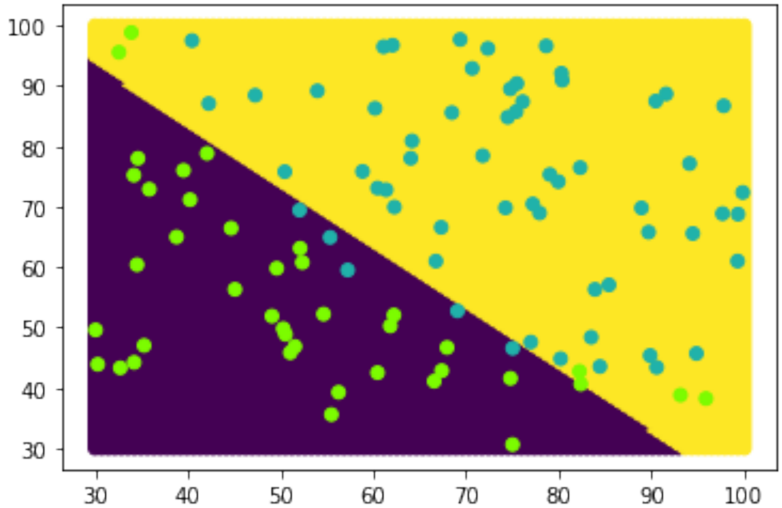
```
#Define prediction function
def predict(w,X):
    # print(f"{w.shape} {X.shape}")
    vector = X@w
    p = sigmoid(vector)
    pred = np.empty(vector.shape)

    pred[p>0.5] =1
    pred[p<=0.5] = 0

    return np.hstack((p, pred))
def plotloss(losslst):
    plt.plot(np.arange(len(losslst)), losslst)
    plt.show()

def train(X,Y,step=0.1,iter=1e6,threshold=1e-8,e=1e-8):
    new_X = np.ones((X.shape[0],X.shape[1]+1))
    new_X[:,1:] = X
    error  = 1e5
    count = 0
    prev = 0
    J_lst = []
    w = np.array([[-65],[0],[0]],dtype=float)
    while error > threshold and count<iter:
        J = loss(new_X,w,Y1,e)
        G = gradient(new_X,w,Y1)
        w -= step*G
        error = np.abs(J-prev)
        prev = J
        J_lst.append(J)
        count+=1
    plotloss(J_lst)

    return w




w = train(X1,Y1,step=0.01,iter=1e6,threshold=1e-8,e=1e-8)
```



Q2 (d)

```
def drawmesh(xrange,yrange,w,X,Y):
    xx = np.linspace(xrange[0], xrange[1], 100)
    yy = np.linspace(yrange[0], yrange[1], 100)
    xv, yv = np.meshgrid(xx, yy)
    xr= xv.ravel()
    yr = yv.ravel()
    new_X = np.ones((len(xr),3))
    new_X[:,1] = xr
    new_X[:,2] = yr
    soft = predict(w,new_X)
    # ListedColormap(["darkorange", "gold", "lawngreen", "lightseagreen"])
    print(f"predict results:\n {soft}")
    plt.scatter(xr,yr,c=soft[:,1])
    plt.scatter(X[:,0],X[:,1],c=Y,cmap=ListedColormap([ "lawngreen", "light
    plt.show()


drawmesh([30,100],[30,100],w,X1,Y1)
```

```
predict results:
 [[2.73278232e-14 0.00000000e+00]
 [3.86740529e-14 0.00000000e+00]
 [5.47311199e-14 0.00000000e+00]
 ...
 [1.00000000e+00 1.00000000e+00]
 [1.00000000e+00 1.00000000e+00]
 [1.00000000e+00 1.00000000e+00]]
```



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

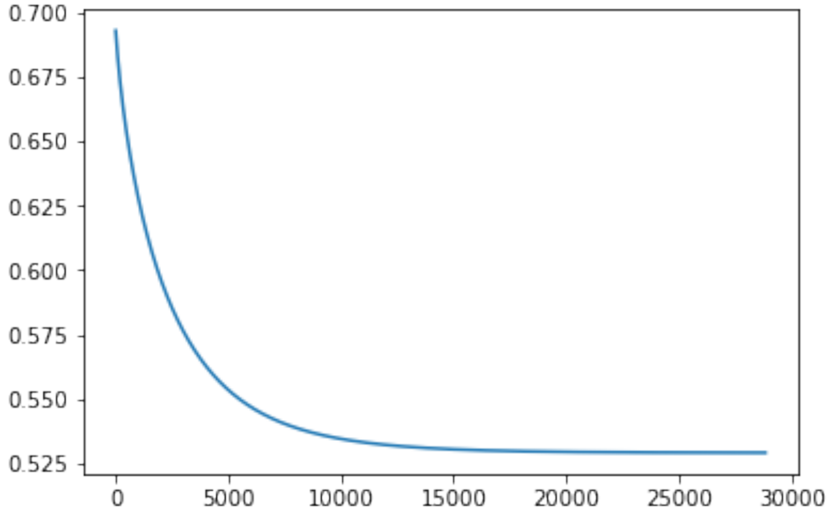
TODO

- 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

Q3 (a)

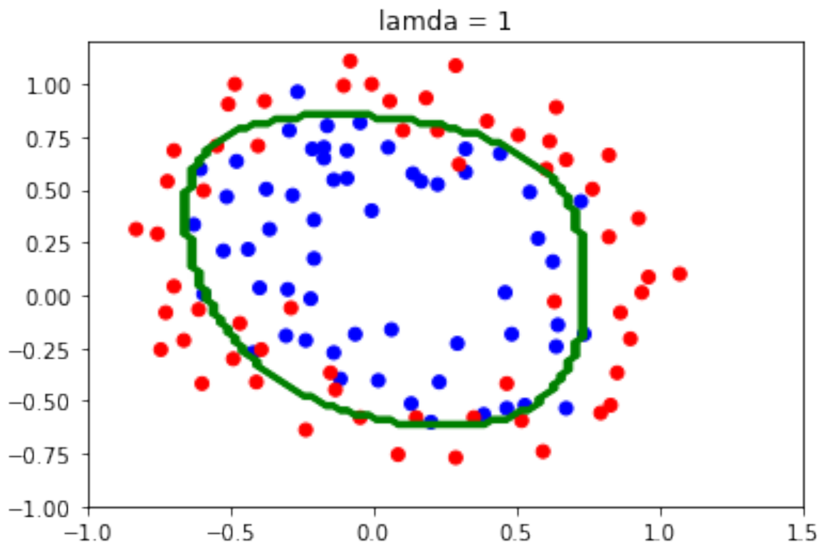
```
# Transform points to 28D space
# Y2_reshape = Y2.reshape(-1,1)
def map_feature_reg(X,feature_num=6):
  x1 = X[:,0].reshape(-1,1)
  x2 = X[:,1].reshape(-1,1)
  count = 0
  for j in range(feature_num+1):
    for k in range(j+1):
      if count ==0:
        feature_lst = (x1**(k))*(x2**(j-k))
      else:
```

```
        feature_lst= np.hstack((feature_lst,(x1**(k))*(x2**(j-k))))
      count +=1

  return feature_lst


def loss_reg(X,Y,w,lamda, n,e):
    # just for the last part w, it will not take the bias term
    w_partial = w.copy()
    w_partial[0,0] = 0

    return (np.mean(-Y*np.log(sigmoid(X @ w)+e)-(1-Y)*np.log(1-sigmoid(X @ 

def gradient_reg(X, Y, w,lamda,n,e):

    w_partial = w.copy()
    w_partial[0,0] = 0
    g = np.mean((sigmoid(X @ w)-Y)*X, axis= 0).reshape((-1,1))+lamda * w_pa
    return g

def train_reg(X,Y,step=0.1,iter=1e6,threshold=1e-8,e=1e-8,lamda=1,loss_pl
ot=True):
      new_X = map_feature_reg(X,6)
      error  = 1e5
      count = 0
      prev = 0
      J_lst = []
      w = np.zeros((28,1))
      n = new_X.shape[0]
      # print(n)
      while error > threshold and count<iter:
        J = loss_reg(new_X,Y,w,lamda,n,e)
        G = gradient_reg(new_X,Y,w,lamda,n,e)
        w -= step*G
        error = np.abs(J-prev)
        prev = J
        J_lst.append(J)
        count+=1
      if loss_plot:
        plotloss(J_lst)
      return w

w = train_reg(X2,Y2,step=0.01,iter=1e6,threshold=1e-8,e=1e-8,lamda=1)
```



Here I define some functions to draw the boundry in both my method and sklearn.

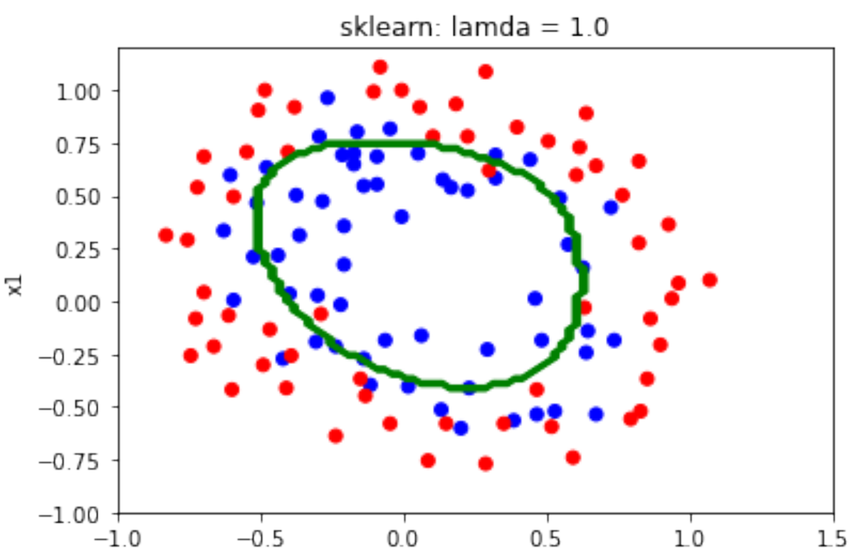
```
def drawmesh_reg(xrange,yrange,w,X,Y,title=None,subplot=None):
    xx = np.linspace(xrange[0], xrange[1], 100)
    yy = np.linspace(yrange[0], yrange[1], 100)
    xv, yv = np.meshgrid(xx, yy)
    xr= xv.ravel()
    yr = yv.ravel()

    xc = xr.reshape(-1,1)
    yc = yr.reshape(-1,1)
    fake_x = np.hstack((xc,yc))

    new_X = map_feature_reg(fake_x)
    # print(new_X.shape)
    soft = predict(w,new_X)
    # ListedColormap(["darkorange", "gold", "lawngreen", "lightseagreen"])
    if subplot==None:
        plt.contour(xv,yv,soft[:,1].reshape((-1,100)),colors ='green')
        # plt.scatter(xr,yr,c=soft[:,1])
        plt.scatter(X[:,0],X[:,1],c=Y,cmap=ListedColormap([ "red", "blue"])
        plt.title(title)
        plt.show()
    else:
        subplot.contour(xv,yv,soft[:,1].reshape((-1,100)),colors ='green')
        # subplot.scatter(xr,yr,c=soft[:,1])
        subplot.scatter(X[:,0],X[:,1],c=Y,cmap=ListedColormap([ "red", "blu
        subplot.set_title(title)

# fig, axs = plt.subplots(1,2,figsize=(16,8))

drawmesh_reg([-1,1.5],[-1,1.2],w,X2,Y2,title="lamda = 1",subplot=None)
```




```
from sklearn import linear_model

def sk_mesh(xrange,yrange,X,Y,c=1,title=None,subplot=None):



    xx = np.linspace(xrange[0], xrange[1], 100)
    yy = np.linspace(yrange[0], yrange[1], 100)
    xv, yv = np.meshgrid(xx, yy)
    # print(xv.shape)

    xr= xv.ravel()
    yr = yv.ravel()

    xc = xr.reshape(-1,1)
    yc = yr.reshape(-1,1)

    fake_x = np.hstack((xc,yc))
    fake_X = map_feature_reg(fake_x)

    new_X = map_feature_reg(X)
    # print(f"{new_X.shape}  {Y2.shape}")
    clf = linear_model.LogisticRegression(penalty="l2", solver="liblinear",
    clf.set_params(C=c)
    clf.fit(new_X,Y2)
    w = clf.coef_.reshape(-1,1)
    soft = predict(w,fake_X)
    # print(soft.shape)

    # ListedColormap(["darkorange", "gold", "lawngreen", "lightseagreen"])
```

```
    if subplot==None:
        plt.contour(xv,yv,soft[:,1].reshape((-1,100)),colors ='green')
        # plt.scatter(xr,yr,c=soft[:,1])
        plt.scatter(X[:,0],X[:,1],c=Y,cmap=ListedColormap([ "red", "blue"])
        plt.title(f"sklearn: lamda = {c**(-1)}")
        plt.ylabel("x2")
        plt.ylabel("x1")
        plt.show()
    else:
        subplot.contour(xv,yv,soft[:,1].reshape((-1,100)),colors ='green')
        # subplot.scatter(xr,yr,c=soft[:,1])
        subplot.scatter(X[:,0],X[:,1],c=Y,cmap=ListedColormap([ "red", "blu
        subplot.set_title(f"sklearn: lamda = {c**(-1)}")
        # subplot.set_y


sk_mesh([-1,1.5],[-1,1.2],X2,Y2,c=1)
```

```
D:\anaconda\envs\piptorch\lib\site-packages\sklearn\utils\validation.py:9
93: DataConversionWarning: A column-vector y was passed when a 1d array w
as expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  y = column_or_1d(y, warn=True)
```



F. Tune the strength of regularization

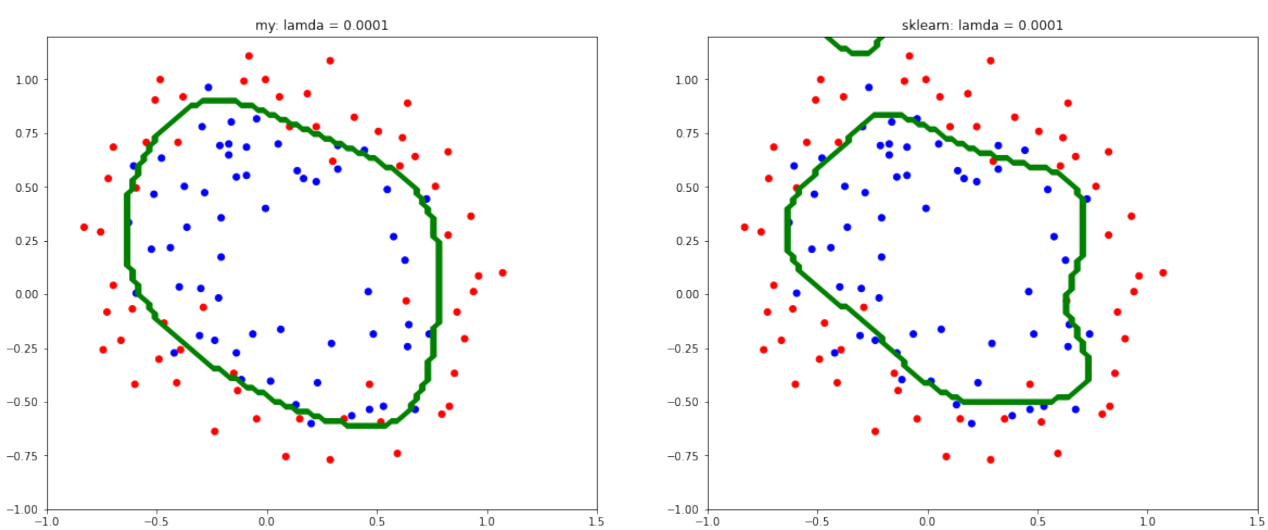
TODO

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

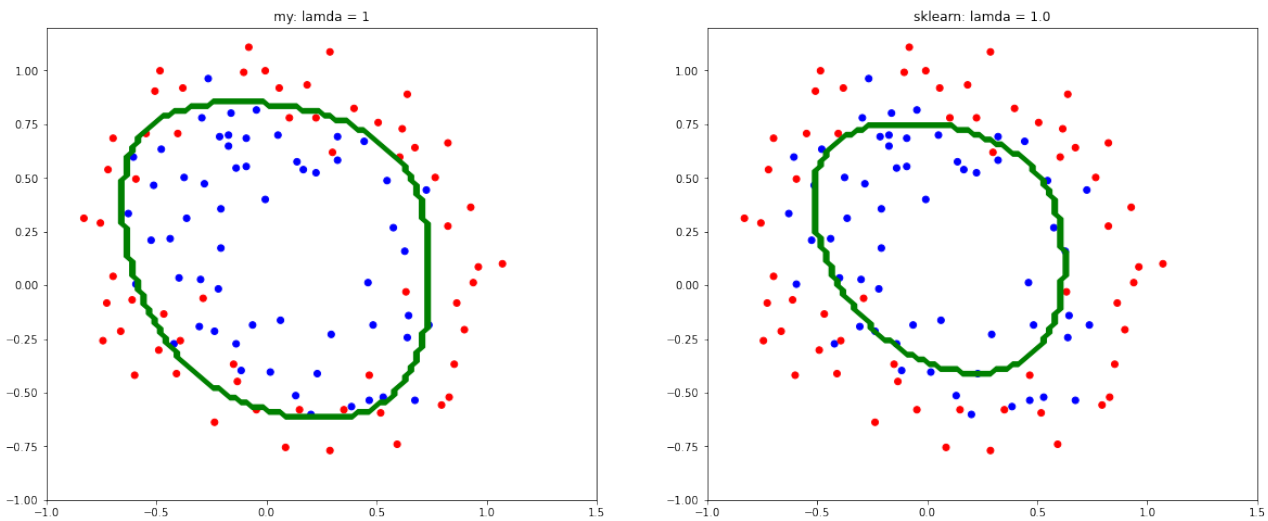
Here are the graphs for use my method and sklearn to draw the boundry in condition lamda = [0.0001,1,1000]

```
for i in [0.0001,1,100]:
    fig, axs = plt.subplots(1,2,figsize=(20,8))

    w = train_reg(X2,Y2,step=0.01,iter=1e6,threshold=1e-8,e=1e-8,lamda=i,lo
    drawmesh_reg([-1,1.5],[-1,1.2],w,X2,Y2,title=f"my: lamda = {i}",subplot
    sk_mesh([-1,1.5],[-1,1.2],X2,Y2,c=i**(-1),subplot=axs[1])
    plt.show()
```

```
D:\anaconda\envs\piptorch\lib\site-packages\sklearn\utils\validation.py:9
93: DataConversionWarning: A column-vector y was passed when a 1d array w
as expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  y = column_or_1d(y, warn=True)
C:\Users\joeww\AppData\Local\Temp/ipykernel_36808/570009302.py:4: Runtime
Warning: overflow encountered in exp
  return (1+np.exp(-mat))**(-1)
```



```
D:\anaconda\envs\piptorch\lib\site-packages\sklearn\utils\validation.py:9
93: DataConversionWarning: A column-vector y was passed when a 1d array w
as expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  y = column_or_1d(y, warn=True)
```




```
D:\anaconda\envs\piptorch\lib\site-packages\sklearn\utils\validation.py:9
93: DataConversionWarning: A column-vector y was passed when a 1d array w
as expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  y = column_or_1d(y, warn=True)
```