ML Midterm Solution, Aug 2018

September 27, 2018

In this exam, you will demonstrate your understanding of the material from the lectures, tutorials, and problem sets.

For each question, insert your answer directly in this sheet. When complete, export the sheet as a PDF and upload to Gradescope.

Note that you have **2.5 hours** to do the exam. Also note that there are some short answer questions that you may be able to answer faster than the coding questions. You might consider answering those questions first to get as much credit as possible!

1 Question 1 (20 points)

Consider the data (X, $\mathbf{y}$) given in the cell below.

```
In [4]: import numpy as np

        X1 = np.array([[ 5.85811186e+00,  1.22514944e-01], [ 6.00000443e+00,  1.35309858e+00],
               [ 5.92284784e+00,  2.73055315e+00], [ 1.52096299e+00,  6.21018604e+00],
               [ 5.30181969e+00,  1.06006684e+00], [ 1.84827940e-02,  2.85281664e+00],
               [ 3.45604657e+00,  1.53032968e+00], [ 4.22908122e-03,  3.36294834e+00],
               [ 3.48985476e+00,  7.51884752e-01], [ 6.26725305e+00,  1.07496703e+00],
               [ 1.14301185e+00,  6.51237438e+00], [ 4.23216371e+00, -2.44093332e+00],
               [ 5.47550663e+00,  4.06333297e-01], [ 6.91545908e-01,  6.40202254e+00],
               [ 5.36323789e+00,  3.34195653e-01], [ 2.05562456e+00,  6.22292165e+00],
               [ 8.41550344e-01,  6.64205417e+00], [ 3.52365485e+00,  2.04313164e+00],
               [ 4.47711189e+00, -3.86246764e-02], [ 5.89528259e+00,  1.79006628e+00],
               [ 6.19166029e-01,  5.11662188e+00], [ 5.54388098e+00,  1.06447058e+00],
               [ 2.32147724e+00,  5.38777768e+00], [ 2.62141168e+00,  4.28575143e+00],
               [ 4.63535586e-02,  4.45755669e+00], [ 5.33931497e+00,  7.04464916e-01],
               [ 5.57510725e+00, -3.39830880e-02], [ 1.96161624e+00,  5.95425951e+00],
               [ 1.91163042e+00,  6.75226744e+00], [ 1.64362576e+00,  8.10340980e+00],
               [ 7.10795973e-01,  8.76085464e+00], [ 6.92524153e-01,  5.61415508e+00],
               [ 4.94898675e-01,  4.54839212e+00], [ 6.06103845e+00,  2.50633947e+00],
               [ 9.73818272e-02,  2.98938521e+00], [ 4.88362231e+00, -1.27782382e+00],
               [ 3.07408705e+00,  3.63515130e+00], [ 3.28864750e+00,  2.07462075e+00],
               [ 6.26354557e+00,  4.53934384e+00], [ 6.16816836e+00,  2.73690942e+00],
               [ 4.21560198e+00, -1.22336763e+00], [ 5.12756678e+00, -1.29691280e+00],
               [ 4.35537106e+00, -1.02344862e+00], [ 2.45336946e+00,  5.36145405e+00],
               [ 1.81532920e+00,  7.15020457e+00], [ 5.07147058e+00,  1.02288646e-01],
        [ 1.54560026e+00,  7.04755026e+00], [ 4.42717299e+00, -2.81259623e-01],
        [ 2.83503002e+00,  4.84934257e+00], [ 5.80106290e+00, -5.74634830e-02],
        [ 2.22759281e+00,  5.29488901e+00], [ 3.08219947e+00,  5.00507772e+00],
        [ 5.79711865e+00,  5.07300906e-01], [ 1.50372185e+00,  7.92150886e+00],
        [ 5.31174440e+00,  4.69530673e-01], [ 1.02455161e+00,  4.37343480e+00],
        [ 4.52702912e+00, -1.04276511e-01], [ 3.88224756e+00, -8.80499022e-01],
        [ 4.05922572e+00, -2.13265308e-01], [ 6.03892811e+00,  2.54764213e+00],
        [ 6.69758084e-01,  8.75729172e+00], [ 3.43081467e+00,  1.78550466e+00],
        [ 5.68295684e+00,  2.56567127e+00], [ 5.77236666e+00,  8.62633177e-01],
        [ 4.62600300e+00, -1.59433537e+00], [ 5.44977295e+00, -6.29484710e-01],
        [ 8.83574711e-01,  6.39890091e+00], [ 9.77493464e-01,  7.04576842e+00],
        [ 6.96010647e-01,  6.32483711e+00], [ 2.42501319e+00,  5.31169779e+00],
        [ 7.13900135e-01,  5.55653866e+00], [ 1.60388408e+00,  6.52536492e+00],
        [ 1.95919971e+00,  6.14051234e+00], [ 3.72469206e+00,  1.16211293e-01],
        [ 2.18109919e+00,  5.03391632e+00], [ 3.37992980e-01,  4.38666016e+00],
        [ 2.66255987e+00,  4.16248548e+00], [ 1.37563677e+00,  9.01655089e+00],
        [ 4.97070262e+00,  7.65748319e-01], [ 5.67177205e+00,  1.60946513e+00],
        [ 5.31404919e+00,  9.70842964e-02], [ 6.02115102e+00,  1.14662909e+00],
        [ 4.05204793e+00, -1.03418242e+00], [ 3.66918406e+00,  1.11617948e+00],
        [ 3.82708241e+00, -6.89595628e-01], [ 4.65955271e+00, -8.63810375e-01],
        [ 3.32919754e+00,  2.31647457e+00], [ 4.33734390e+00, -1.29535742e+00],
        [ 3.95943543e+00,  4.58081833e-01], [ 2.89422338e+00,  4.49606922e+00],
        [ 2.19510893e+00,  7.36292377e+00], [ 2.79133907e+00,  4.25746815e+00],
        [ 7.19221108e-01,  6.54004102e+00], [ 5.13548270e+00, -2.54674801e-01],
        [ 2.05933161e+00,  5.28710542e+00], [ 1.78202945e+00,  7.16853185e+00],
        [ 3.48957680e+00,  1.36969854e+00], [ 4.93781831e+00,  6.91655345e-01],
        [ 4.27909400e+00,  2.03346686e-02], [ 5.15336506e+00, -4.14053935e-01]])

X2 = np.array([[ 3.43323051, -4.68184123], [ 2.59748185, -0.61500731],
        [ 0.33480371, -1.47508942], [ 5.7825976 , -5.44824068],
        [ 0.87070876, -0.89526139], [ 0.01802905, -1.89299839],
        [ 2.74929135, -0.56312653], [ 4.22560236, -6.12770556],
        [ 2.78127731, -0.95800779], [ 3.47222918, -3.33378672],
        [ 0.9782014 , -1.00938521], [ 2.25445075, -0.90402158],
        [ 5.55216081, -6.18491739], [ 6.13910761, -3.93666989],
        [ 2.99043663, -2.89123684], [ 0.44884867, -1.45297558],
        [ 3.88912523, -5.06048717], [ 2.10025175, -1.1028566 ],
        [ 0.93182831,  1.75229369], [ 3.80362678, -4.04122195],
        [ 0.34639776, -0.87243213], [ 5.7116009 , -4.93209123],
        [ 5.05087165, -6.67367203], [ 1.3492558 ,  1.33394759],
        [ 2.95688016, -2.19674503], [ 4.96859322, -8.01640548],
        [ 5.87695275, -3.34649668], [ 1.94117582,  2.27797699],
        [ 3.62547526, -4.0217949 ], [ 0.82629384, -0.27290357],
        [ 3.6890367 , -5.82061241], [ 4.93797566, -6.52602515],
        [ 5.90841492, -5.27521196], [ 5.1879215 , -7.80427044],
        [ 2.09766098, -0.68048301], [ 5.76655061, -3.82518968],
        [ 0.97690188, -0.91069622], [ 2.26501337,  1.56056043],
        [ 3.13676069, -2.29446312], [ 5.29747136, -5.55385824],
       [ 2.00649857,  0.68266315], [ 0.77856457, -1.75844385],
       [ 3.54571209, -4.98304275], [ 0.97247461, -0.11610218],
       [ 4.31311129, -7.82947521], [ 3.48007352, -6.04086403],
       [ 1.50781401,  0.33381045], [ 2.43853928, -0.03145604],
       [ 0.25314147, -2.45908731], [ 1.35654226, -0.22865539],
       [ 5.06582758, -6.90535573], [ 3.41686466, -3.37239301],
       [ 5.61214807, -5.25701105], [ 5.97522182, -3.22566212],
       [ 3.18939015, -4.3188073 ], [ 5.46345617, -4.82657628],
       [ 5.41151575, -6.0126475 ], [ 5.347017  , -6.83984988],
       [ 5.7259852 , -3.72566258], [ 1.38905244, -0.13844681],
       [ 3.9562558 , -6.09691018], [ 1.25473486,  0.8640623 ],
       [ 0.08936814, -3.30723552], [ 3.22365241, -4.78721191],
       [ 3.2415687 , -4.24880257], [ 1.1302129 , -1.41426559],
       [ 4.12636748, -7.32837589], [ 2.15023019,  0.05432966],
       [ 4.55269902, -6.73020725], [ 1.63594817,  1.72806532],
       [ 0.099722  , -1.90546332], [ 4.61336823, -7.07377162],
       [ 3.55425568, -3.30089928], [ 4.52715899, -7.27897524],
       [ 5.30602493, -7.61995372], [ 5.44738276, -6.9766927 ],
       [ 4.34946148, -7.64515063], [ 4.04771578, -7.19386371],
       [ 1.85701731,  0.39977015], [ 3.04610896, -3.2246929 ],
       [ 2.96501443, -1.32784036], [ 1.15817849,  0.48576348],
       [ 1.11438724,  1.46252362], [ 0.44152831,  0.09032514],
       [ 1.95859258, -1.05902546], [ 2.63644257, -1.85839175],
       [ 1.93538744,  0.77671848], [ 4.12489761, -4.92407274],
       [ 3.93239509, -5.08468575], [ 3.22929697, -4.11585099],
       [ 4.94033586, -4.20813851], [ 2.77167364, -2.60322887],
       [ 2.05535528,  2.02195992], [ 0.97916579,  0.86646976],
       [ 1.84314531,  2.3764651 ], [ 4.92906354, -6.32746025],
       [ 1.72832737,  1.88550113], [ 2.83989839, -3.62635215],
       [ 4.17304198, -7.15506579], [ 0.47159292,  0.28458576]])
```

```
X = np.concatenate([X1, X2],0);
y = np.concatenate([-np.matrix(np.ones([100,1])),np.matrix(np.ones([100,1]))]);
```

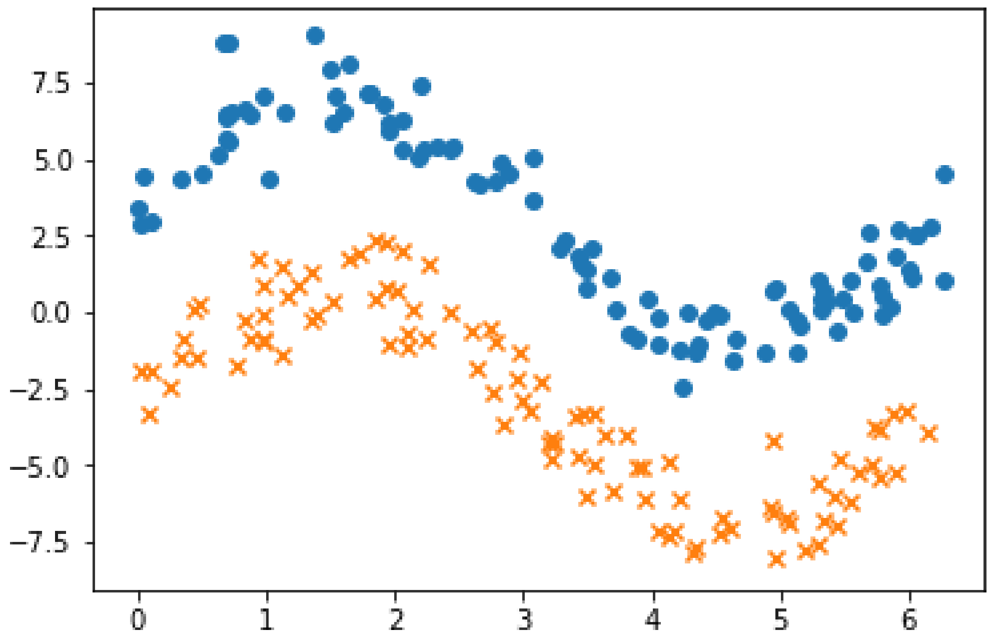
(Q1 continued) Do the following:

1. In the cell below, make a scatter plot of the data for class 1 and class 2 in different colors.
2. Answer the question, are the data linearly separable?

In [6]:
```
# Place code to plot the data here

import matplotlib.pyplot as plt

fig = plt.figure()
axis = fig.add_subplot(111)
axis.scatter(X1[:,0], X1[:,1], marker='o')
axis.scatter(X2[:,0], X2[:,1], marker='x')
plt.show()
```



No, the data are not linearly separable.

2 Question 2 (20 points)

Do the following:

1. **Build a logistic regression model for the data (X, y). Give the optimal parameter vector (θ) below:**

The optimal θ is $\begin{bmatrix}-2.115700 & -1.809640 & 6.605089\end{bmatrix}^\top$.

2. **Plot the data again, along with the decision boundary $\theta^\top \mathbf{x} = 0$.**

See below.

3. **What is the accuracy of this model on its training set?** (Note: in this problem there is no need to create separate training and test sets. Just train and test on the full data set.)

Best accuracy is 94.5% or 95% depending on random pattern presentation order, achieved after about 3000 iterations of stochastic gradient ascent with a learning rate of 0.01.

```
In [148]: ylr = y
          ylr[y==-1] = 0

          def h(x,theta):
              z = np.dot(x,theta)
              return 1/(1+np.exp(-z))
```

```
        m = X.shape[0]
        alpha = 0.001
        theta = np.array([[0],[1],[0]])
        converged = False
        XX = np.concatenate([X,np.ones([m,1])],1)
        llbest = -1e+6
        iter = 0
        acc = 0
        print('Running...')
        while not converged:
            perm = np.random.permutation(m)
            for i in range(0,m):
                ind = perm[i]
                x = XX[ind,:]
                x.shape = [1,3]
                yact = ylr[ind,0]
                ypred = h(x,theta)[0]
                theta = theta + alpha * (yact - ypred) * x.T
            ypred = h(XX,theta)
            ypred_crisp = ypred > 0.5
            acc = sum(ypred_crisp == ylr) / m
            loglike = np.sum(np.multiply(ylr, np.log(ypred)) + np.multiply((1-ylr), np.log(1-y
            if loglike < llbest + 1e-6:
                converged = True
            if loglike > llbest:
                llbest = loglike
            iter = iter + 1

        print('Best log likelihood %f at iteration %d' % (llbest, iter))
        print('Best theta %f %f %f' % (theta[0], theta[1], theta[2]))
        print("Iter %d accuracy %f" % (iter, acc))
```

```
Running...
Best log likelihood -25.465611 at iteration 3336
Best theta -2.091375 -1.796405 6.538327
Iter 3336 accuracy 0.945000
```

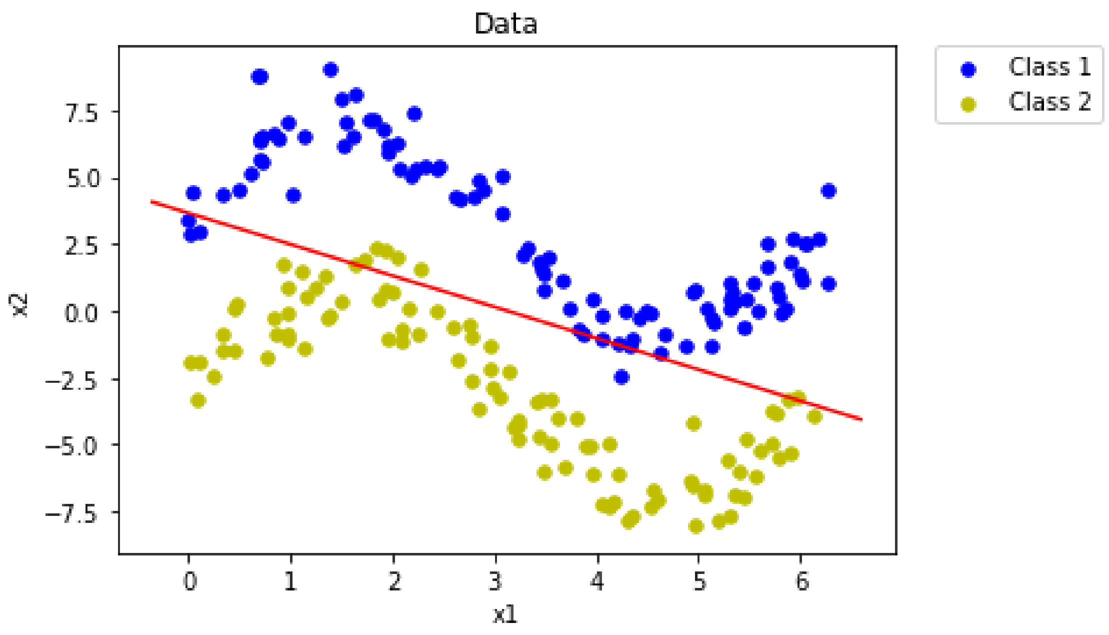
```
In [91]: # Place code to plot the data here

         fig = plt.figure()
         axes = fig.add_subplot(111)

         def plot_data(X1,X2,axis):
             series1 = axis.scatter(np.array(X1[:,0]), np.array(X1[:,1]), s=30, c='b', marker='o
             series2 = axis.scatter(np.array(X2[:,0]), np.array(X2[:,1]), s=30, c='y', marker='o
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('Data')
    plt.legend(handles=[series1, series2], bbox_to_anchor=(1.05, 1), loc=2, borderaxesp

def plot_w(w,b,axis):
    ylim = axis.get_ylim()
    xlim = axis.get_xlim()
    p1 = (xlim[0], - (w[0,0] * xlim[0] + b) / w[1,0])
    p2 = (xlim[1], - (w[0,0] * xlim[1] + b) / w[1,0])
    plt.plot((p1[0],p2[0]), (p1[1],p2[1]), 'r-')

plot_data(X1,X2,axes)
w = theta[0:2,0]
w.shape = [2,1]
b = theta[2,0]
plot_w(w,b,axes)
```



3 Question 3 (20 points)

Now suppose that we'd like to train a SVM to classify these data, but also suppose that you don't know how to modify the SVM optimization to handle data that are not linearly separable (I promised that wouldn't be on the midterm, didn't I?).

Then your goal is to come up with a feature mapping $\mathbf{Œ}(\mathbf{x})$ that will transform the data so that they are linearly separable.

Do the following:

1. Explain in general what kind of transformation we should use. (*Hint: think about trigonometric functions*)

The data look like shifted sine functions. If we leave x_1 alone and subtact $\sin(x_1)$ from x_2, perhaps scaled appropriately, the data should become linearly separable.

2. Experiment to find a mapping that transforms the data to another 2-dimensional feature space until you have one that works. In the cell below, add the code for the transformation and code to plot the transformed data set.

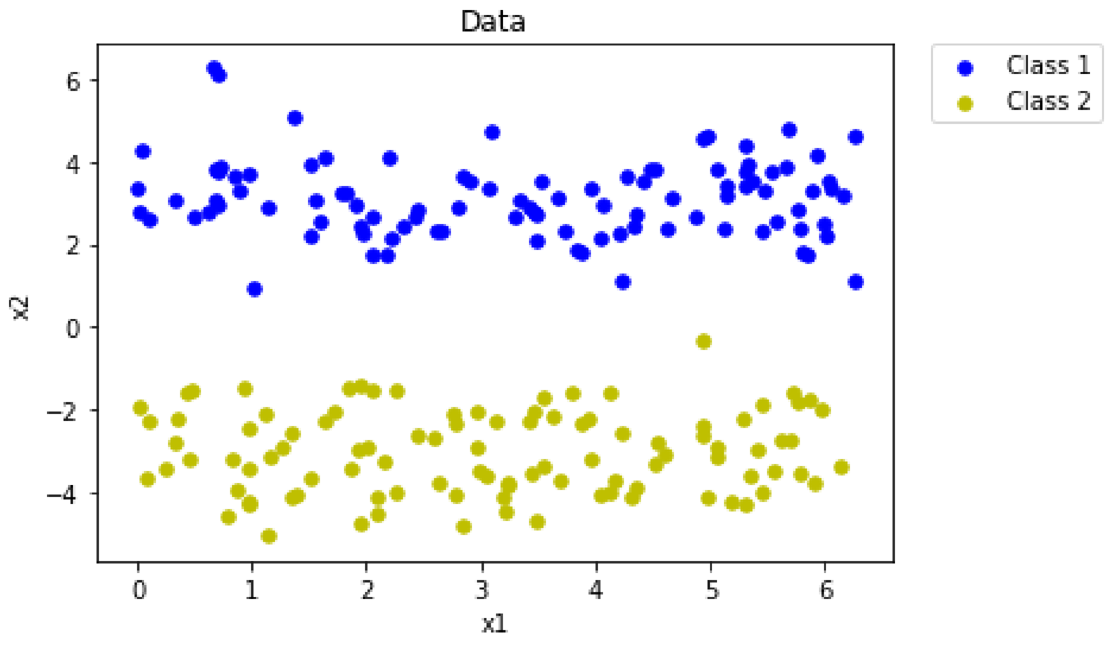
```
In [97]: # Fill in your mapping function here:

         def phi_q3(X):
             # Transform X
             m = X.shape[0]
             x1_new = X[:,0]                       # Copy x1
             x2_new = X[:,1] - 4 * np.sin(X[:,0])  # Subtract 4*sin(x1) from x2
             x1_new.shape = [m,1]
             x2_new.shape = [m,1]
             return np.concatenate((x1_new,x2_new),1)

         # Place code to plot the data here

         fig = plt.figure()
         axis = fig.add_subplot(111)

         plot_data(phi_q3(X1),phi_q3(X2),axis)
```



4 Question 4 (20 points)

Next we use the cvxopt library to train a SVM on the data transformed according to your answer in Question 3.

Do the following:

1. Place your code to convert X, $\mathbf{y}$ to the primal SVM problem (minimizing $\|\mathbf{w}\|^2$) below:

```
In [143]: import cvxopt

          def cvxopt_solve_qp(Q, c, A=None, b=None, E=None, d=None):
              Q = .5 * (Q + Q.T)  # make sure Q is symmetric
              args = [cvxopt.matrix(Q), cvxopt.matrix(c)]
              if A is not None:
                  args.extend([cvxopt.matrix(A), cvxopt.matrix(b)])
                  if E is not None:
                      args.extend([cvxopt.matrix(E), cvxopt.matrix(d)])
              sol = cvxopt.solvers.qp(*args)
              if 'optimal' not in sol['status']:
                  return None
              return np.array(sol['x']).reshape((Q.shape[1],))

          # Transform inputs to a QP problem

          XX = phi_q3(X)
          yy = y
          yy[yy==0] = -1

          Q = np.array([[1.,0,0],[0,1,0],[0,0,0]])
          c = np.zeros([3,1])
          A = np.multiply(np.tile(-yy,[1, 3]), np.concatenate([XX, np.ones([m,1])],1))
          b = -np.ones([m,1])

          x = cvxopt_solve_qp(Q, c, A, b);
          w = x[0:2]
          w.shape = [2,1]
          b = x[2]
          print('Optimal w: [%f %f] b: %f' % (w[0], w[1], b))
```

```
     pcost       dcost       gap    pres   dres
 0:  4.3750e-02  5.0456e+01  7e+02  2e+00  1e+03
 1:  2.6609e-01 -1.5574e+02  3e+02  7e-01  4e+02
 2:  6.4005e-01 -1.3605e+02  2e+02  4e-01  2e+02
 3:  1.2138e+00 -1.8659e+01  2e+01  4e-02  2e+01
 4:  1.2316e+00  7.6195e-01  5e-01  1e-15  4e-14
 5:  1.0235e+00  9.9584e-01  3e-02  9e-16  2e-14
 6:  1.0160e+00  1.0157e+00  3e-04  9e-16  7e-15
 7:  1.0159e+00  1.0159e+00  3e-06  1e-15  8e-15
 8:  1.0159e+00  1.0159e+00  3e-08  9e-16  8e-15
```

```
Optimal solution found.
Optimal w: [0.049775 -1.424564] b: 0.310187
```

2) What are the optimal **w** and b in the feature space? Write your answer here.

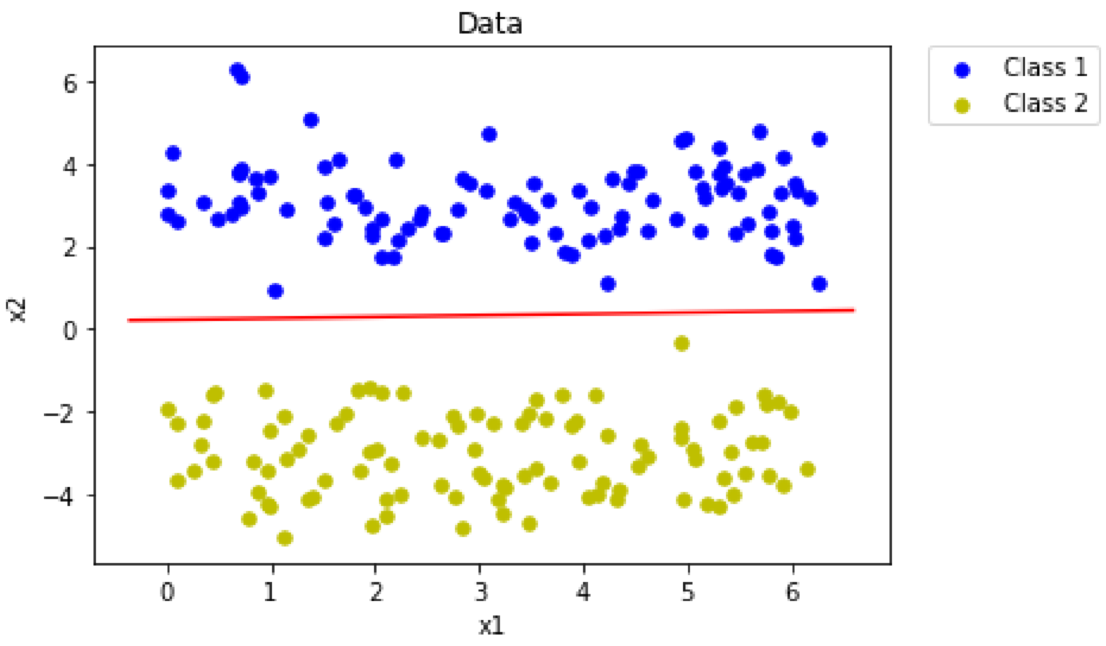
The optimal w is $\begin{bmatrix}0.049775 & -1.424564\end{bmatrix}^\top$**, and the optimal** b **is 0.310187.**

3) Plot the transformed data with the decision boundary. Add code below.

```
In [142]: # Code to plot transformed data and the SVM decision boundary here

          fig = plt.figure()
          axis = fig.add_subplot(111)

          plot_data(XX[0:100,:], XX[100:,:], axis)
          plot_w(w, b, axis)
```



5 Question 5 (20 points)

Suppose you had reason to believe that the data for each class given in Question 1 were generated by a random process in which, for each example, variable x_1 was sampled from a uniform distribution over the range $[0..2\pi]$ then x_2 was sampled from a Gaussian distribution with mean $\alpha \cos(x_1 + \phi) + \gamma$ and variance 1.

The data for each class thus has three parameters: α, ϕ, and γ.

Supposing that we would like to construct a classifier based on this generative model, start by writing the likelihood function

$$\begin{aligned}
L(\alpha,\phi,\gamma) &= p(\mathtt{X};\alpha,\phi,\gamma) \\
&= \textstyle\prod_{i=1}^{m} p(\mathbf{x}^{(i)};\alpha,\phi,\gamma) \\
&= \textstyle\prod_{i=1}^{m} p(x_1^{(i)};\alpha,\phi,\gamma)p(x_2^{(i)} \mid x_1^{(i)};\alpha,\phi,\gamma) \\
&= \textstyle\prod_{i=1}^{m} \frac{1}{2\pi} p(x_2^{(i)} \mid x_1^{(i)};\alpha,\phi,\gamma) \\
&= \textstyle\prod_{i=1}^{m} \frac{1}{2\pi}\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_2^{(i)}-\alpha\cos(x_1^{(i)}+\phi)-\gamma)^2}{2\sigma^2}} \\
&= \textstyle\prod_{i=1}^{m} \frac{1}{(2\pi)^{3/2}} e^{-\frac{1}{2}(x_2^{(i)}-\alpha\cos(x_1^{(i)}+\phi)-\gamma)^2}
\end{aligned}$$

Then derive the log likelihood function

$$\begin{aligned}
\updownarrow(\alpha,\phi,\gamma) &= \log L(\alpha,\phi,\gamma) \\
&= \textstyle\sum_{i=1}^{m} \log(\frac{1}{(2\pi)^{3/2}}) - \sum_{i=1}^{m} \frac{1}{2}(x_2^{(i)} - \alpha\cos(x_1^{(i)}+\phi) - \gamma)^2 \\
&= -\frac{3m}{2}\log(2\pi) - \frac{1}{2}\textstyle\sum_{i=1}^{m}(x_2^{(i)} - \alpha\cos(x_1^{(i)}+\phi) - \gamma)^2
\end{aligned}$$

Towards finding maximum likelihood estimates of α, ϕ, and γ, first find the partial derivatives

$$\frac{\partial\updownarrow}{\partial\alpha} = \sum_{i=1}^{m}(x_2^{(i)} - \alpha\cos(x_1^{(i)}+\phi) - \gamma)\cos(x_1^{(i)}+\phi)$$

$$\frac{\partial\updownarrow}{\partial\phi} = -\sum_{i=1}^{m}(x_2^{(i)} - \alpha\cos(x_1^{(i)}+\phi) - \gamma)\alpha\sin(x_1^{(i)}+\phi)$$

$$\frac{\partial\updownarrow}{\partial\gamma} = \sum_{i=1}^{m}(x_2^{(i)} - \alpha\cos(x_1^{(i)}+\phi) - \gamma)$$

Considering the forms of the partial derivatives of the likelihood function, explain how we could go about finding the optimal parameters

$$\alpha^*,\phi^*,\gamma^* = \text{argmax}_{\alpha,\phi,\gamma}\updownarrow(\alpha,\phi,\gamma).$$

Well, we would set each of the partial derivatives above to 0 then try to solve for the parameters. We can clearly use the third equation to eliminate γ from the other two equations, but then we have two nonlinear equations in two unknowns and no obvious way to solve them. In this situation, we would probably have to begin with an initial guess as to the parameters then use Newton's method to find the zeros. If we were allowed to assume $\phi = -\pi/2$ (which appears to be the case by inspecting the graphs of the two training set distributions) we would have a simpler solution.

Finally, supposing you were able to obtain optimal parameters α, ϕ, and γ for each of the two classes in Question 1, and assuming $p(y=1) = p(y=-1) = \frac{1}{2}$, explain how you would use these optimal parameters to predict the label of a new input $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\top$.

Clearly, we should predict

$$\begin{aligned}
y^* &= \text{argmax}_{\hat{y}\in\{-1,1\}} p(y=\hat{y} \mid \mathbf{x}) \\
&= \text{argmax}_{\hat{y}\in\{-1,1\}} \tfrac{p(\mathbf{x}|y=\hat{y})p(y=\hat{y})}{p(\mathbf{x})} \\
&= \text{argmax}_{\hat{y}\in\{-1,1\}} p(\mathbf{x} \mid y=\hat{y}) \\
&= \text{argmax}_{\hat{y}\in\{-1,1\}} e^{-\frac{1}{2}(x_2-\alpha_{\hat{y}}\cos(x_1+\phi_{\hat{y}})-\gamma_{\hat{y}})^2} \\
&= \text{argmin}_{\hat{y}\in\{-1,1\}} (x_2 - \alpha_{\hat{y}}\cos(x_1+\phi_{\hat{y}}) - \gamma_{\hat{y}})^2
\end{aligned}$$

We simply evaluate the expression for the two possible values of $\hat{y}$ and choose the one that gives us the smallest distance to the curve.

```
In [ ]:
```