Assignment2

November 1, 2017

0.0.1 Exercise 1.

```
In [1]: import matplotlib.pylab as plt
        import numpy as np
        import pandas as pd
        import scipy as sp
        import seaborn as sns
        import time
        import pdb
        import math
        from sklearn.metrics import accuracy_score
        %matplotlib inline
```

```
In [2]: # Import data
        data = pd.read_csv('applesOranges.csv')
        data.head()
```

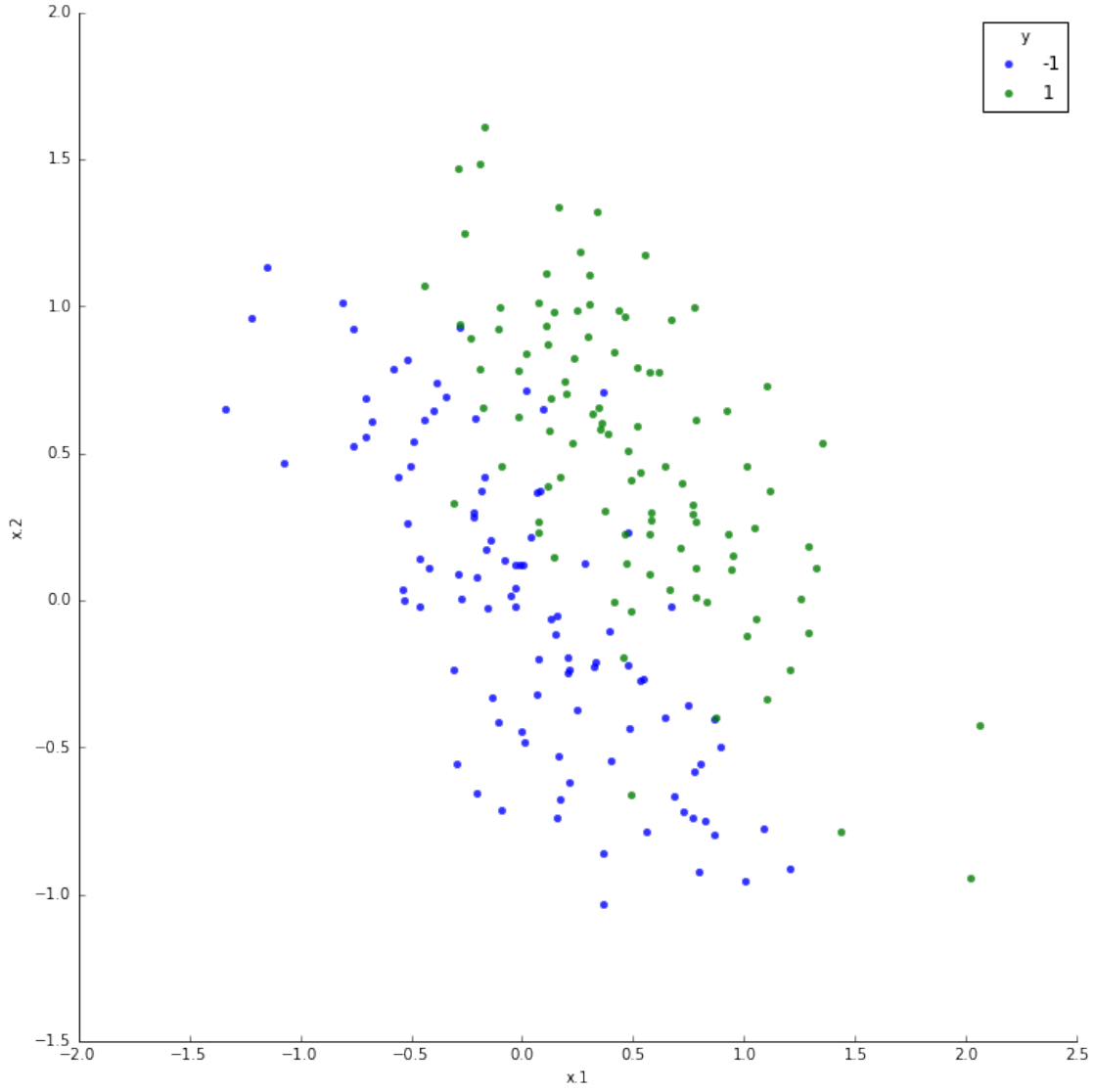
```
Out[2]:      x.1    x.2  y
        0  0.365  0.708  0
        1  0.543 -0.268  0
        2 -0.401  0.643  0
        3  0.866 -0.796  0
        4 -0.386  0.742  0
```

```
In [3]: data.loc[data.y == 0,['y']] = -1
        data.head()
```

```
Out[3]:      x.1    x.2  y
        0  0.365  0.708 -1
        1  0.543 -0.268 -1
        2 -0.401  0.643 -1
        3  0.866 -0.796 -1
        4 -0.386  0.742 -1
```

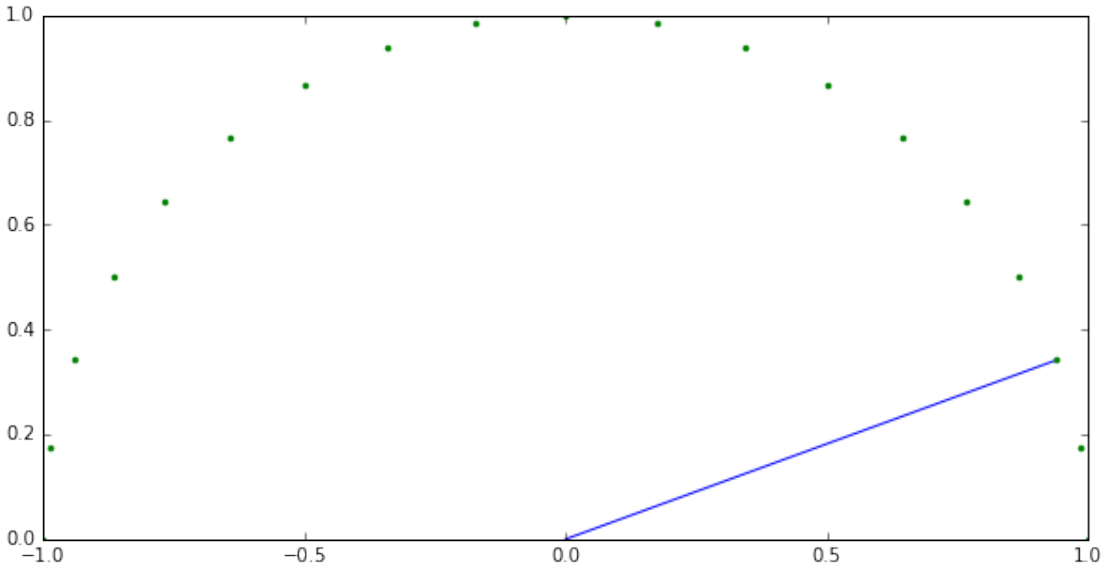
a)

```
In [4]: # Plot the data in a scatter plot
        sns.lmplot('x.1', 'x.2', data=data, hue='y', fit_reg=False, size=10, legend
```



b)

```
In [5]: phi = np.linspace(0,np.pi,19)
        w1, w2 = np.cos(phi), np.sin(phi)
        w = np.vstack((w1,w2))
        plt.figure(figsize=(10,5))
        plt.plot((0,w1[2]), (0,w2[2]))
        plt.plot(w1,w2, '.');
```



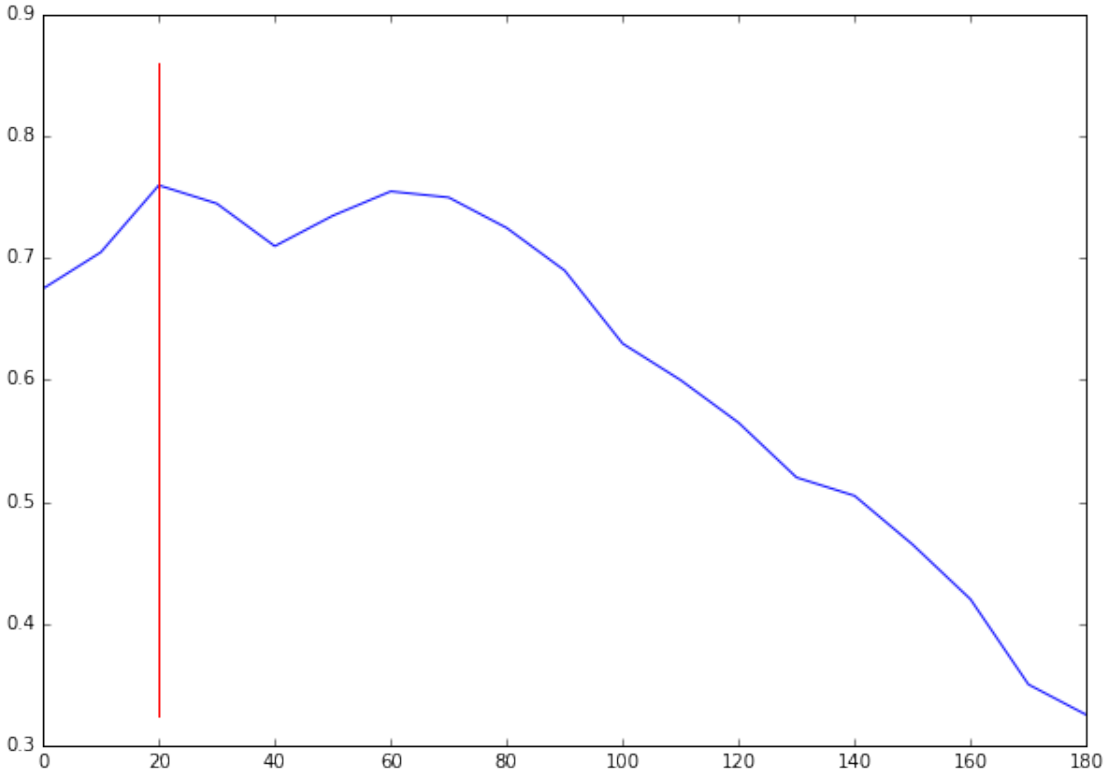

```
In [6]: def get_scalar_product(data, weights):
            scalar_product = np.dot(w.T, data.iloc[:,:2].T)
            return scalar_product
```

```
In [7]: theta = 0
        y = np.sign(get_scalar_product(data, w)-theta)
        p = [accuracy_score(data.y, y[x]) for x in range(len(y))]
        alpha = [np.round(math.degrees(phi[x]),2) for x in range(len(phi))]
```

```
In [8]: alpha
```

```
Out[8]: [0.0,
         10.0,
         20.0,
         30.0,
         40.0,
         50.0,
         60.0,
         70.0,
         80.0,
         90.0,
         100.0,
         110.0,
         120.0,
         130.0,
         140.0,
         150.0,
         160.0,
         170.0,
         180.0]
```

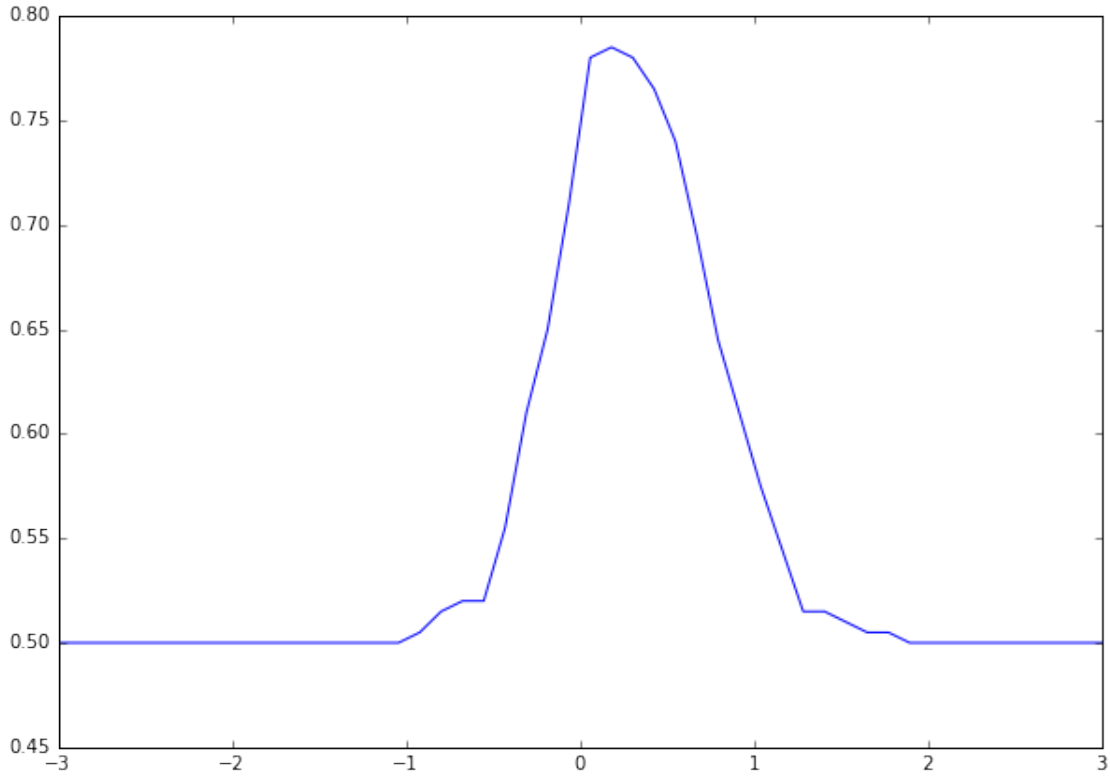
```
In [9]: plt.figure(figsize=(10,7))
        plt.plot(alpha, p)
        plt.plot((alpha[np.argmax(p)],alpha[np.argmax(p)]),(min(p),(max(p)+0.1)),
```



c)

From these weight vectors, pick the w yielding best performance. Now vary the $\theta \in [-3, 3]$ and pick the value of θ giving the best performance.

```
In [11]: theta = np.linspace(-3,3,50)
         y = [np.sign(np.dot(w[:,np.argmax(p)].T,data.iloc[:,:2].T)-theta[x]) for x
         p = [accuracy_score(data.y, y[x]) for x in range(len(y))]

In [12]: plt.figure(figsize=(10,7))
         plt.plot(theta, p);
```




```
In [72]: np.argmax(p)
```

```
Out[72]: 26
```

```
In [14]: y_pred = np.sign(np.dot(w[:,2].T,data.iloc[:,:2].T)-theta[np.argmax(p)])
         y_pred
```

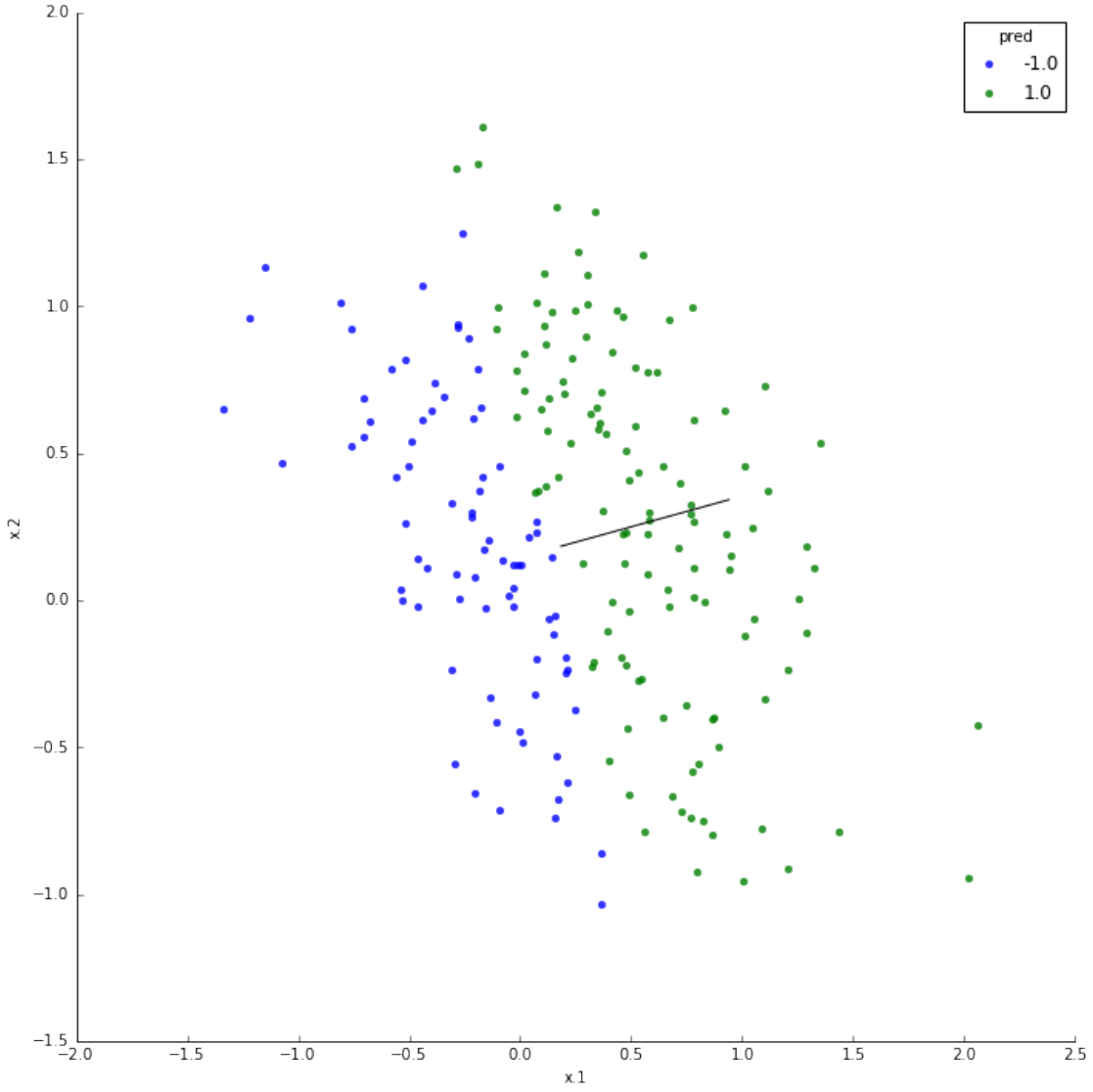
```
Out[14]: array([ 1.,  1., -1.,  1., -1., -1., -1.,  1., -1., -1., -1., -1., -1.,
                -1., -1.,  1., -1.,  1., -1., -1.,  1., -1., -1.,  1., -1.,  1.,
                 1., -1.,  1., -1.,  1., -1.,  1., -1., -1., -1., -1., -1., -1.,
                -1., -1., -1., -1., -1., -1.,  1.,  1.,  1.,  1., -1., -1., -1.,
                 1., -1., -1.,  1., -1., -1., -1., -1.,  1.,  1.,  1., -1., -1.,
                 1., -1.,  1., -1., -1., -1., -1.,  1., -1.,  1., -1., -1., -1.,
                 1.,  1., -1.,  1., -1., -1.,  1., -1., -1., -1.,  1., -1., -1.,
                -1., -1., -1.,  1., -1., -1., -1., -1., -1., -1., -1.,  1.,  1.,
                 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
                -1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                 1.,  1., -1.,  1.,  1.,  1.,  1.,  1., -1.,  1., -1.,  1., -1.,
                 1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,
                 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,
                 1., -1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                 1.,  1.,  1.,  1.,  1.])
```

```
In [15]: data.y.values

Out[15]: array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1], dtype=int64)

In [16]: data['pred'] = y_pred
```

d)

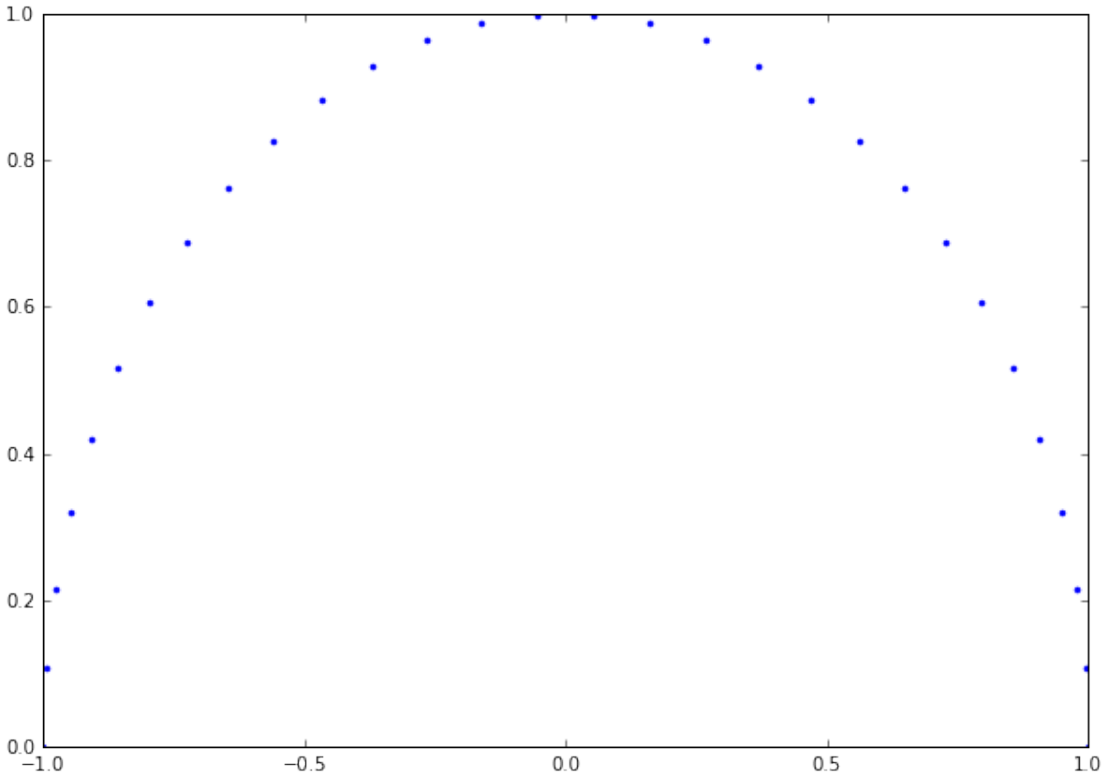
Plot the data points,colored according to the classification corresponding to these parameter values. Plot the weight vector w in the same plot. How do you interpret your results?

```
In [17]: w2[2]

Out[17]: 0.34202014332566871

In [18]: sns.lmplot('x.1', 'x.2', data=data, hue='pred', fit_reg=False, size=10, le
         plt.plot((theta[26],w1[2]), (theta[26],w2[2]), 'K');
         #plt.arrow(theta[26],theta[26], w1[2],w2[2], head_width=0.05, head_length=
```

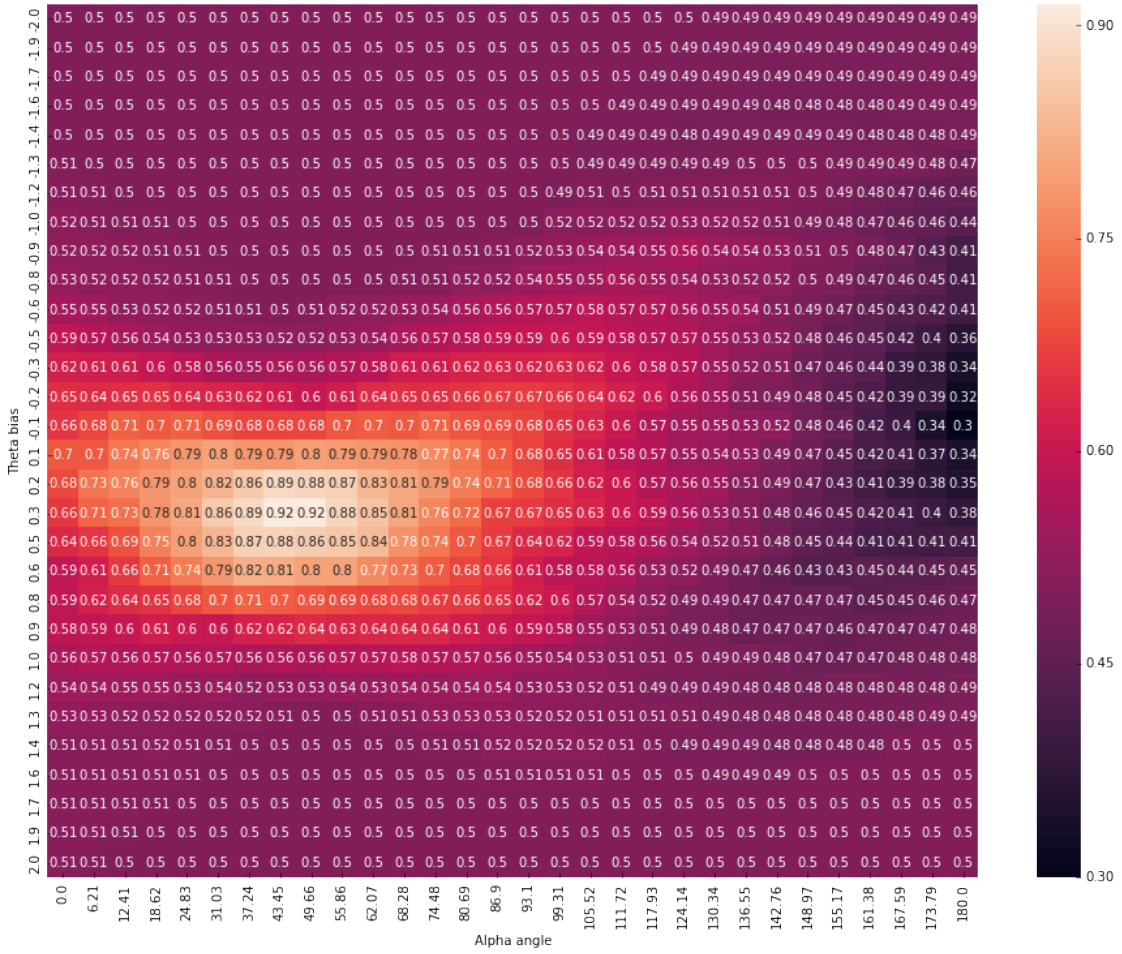


1. e) Find the best combination of w and θ by exploring all combinations of α and θ (within a sensible range and with sensible precision) and plotting the performance of all combinations in a heatmap.

```
In [19]: phi = np.linspace(0,np.pi,30)
         w1, w2 = np.cos(phi), np.sin(phi)
         w = np.vstack((w1,w2))
         alpha = [np.round(math.degrees(phi[x]),2) for x in range(len(phi))]

         plt.figure(figsize=(10,7))
         plt.plot(w1,w2, '.');
```




```
In [20]: theta = np.linspace(-2,2,30)
         p_matrix = np.zeros((len(alpha),len(theta)))
         for i in range(w.shape[1]):
             y = [np.sign(np.dot(w[:,i].T,data.iloc[:,:2].T)-theta[x]) for x in ran
             p_matrix[:,i] = [accuracy_score(data.y, y[x]) for x in range(len(y))]

In [21]: p_df = pd.DataFrame(p_matrix)
         p_df.columns = np.round(alpha,2)
         p_df.index = np.round(theta,1)

In [22]: fig, ax = plt.subplots(figsize=(16,12))
         sns.heatmap(p_df, annot=True)
         ax.set_xlabel('Alpha angle')
         ax.set_ylabel('Theta bias');
```



1. f) Can the optimization method (e) be applied to any classification problem? Discuss potential problems and give an application example in which the above method must fail.

Classification problems which do not have a global but several local minima

0.0.2 Exercise 2 - Ajla's Version

a)

```
In [58]: X = np.arange(-2, 2.25, 0.25)

In [59]: res = []

         for i in range(50):
             a = np.random.normal(0, 2, 10)
             w = np.random.normal(0, 1, 10)
             b = np.random.uniform(-2, 2, 10)
```

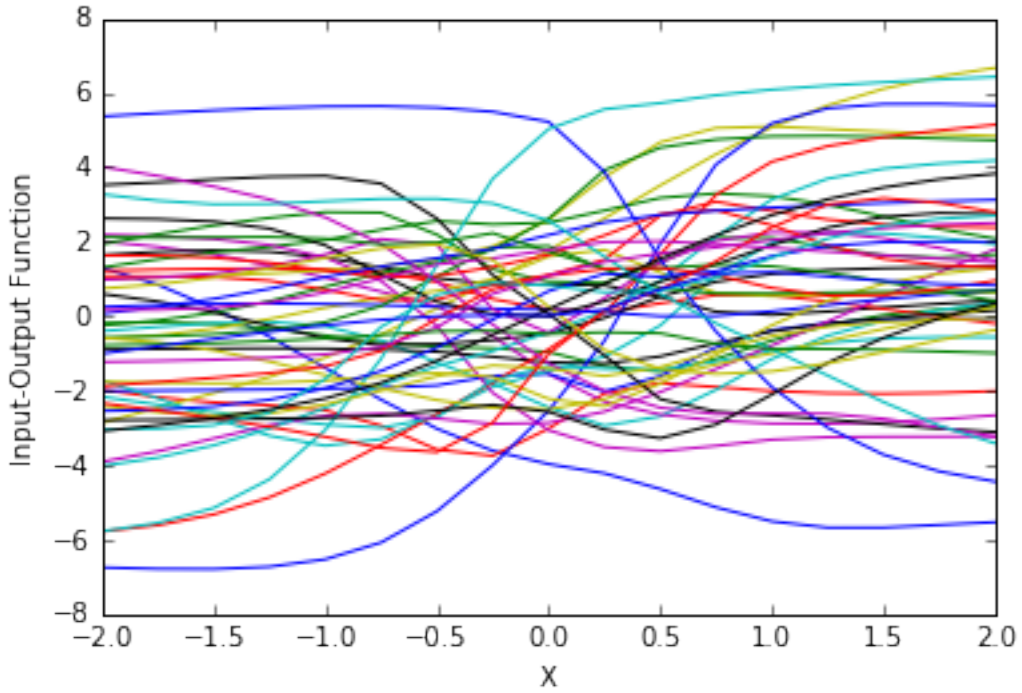
```
        f = []
        for i in range(10):
            f.append([np.sum(a[i]*x - b[i]) for x in X])

        s = np.dot(w.transpose(), np.tanh(f))
        res.append(s)
```

In [60]:
```
results = np.asarray(res)
results.shape
```

Out[60]: (50L, 17L)

In [61]:
```
plt.plot(np.arange(-2, 2.25, 0.25), results.transpose())
plt.xlabel('X')
plt.ylabel('Input-Output Function')
```

Out[61]: <matplotlib.text.Text at 0x10c97d68>



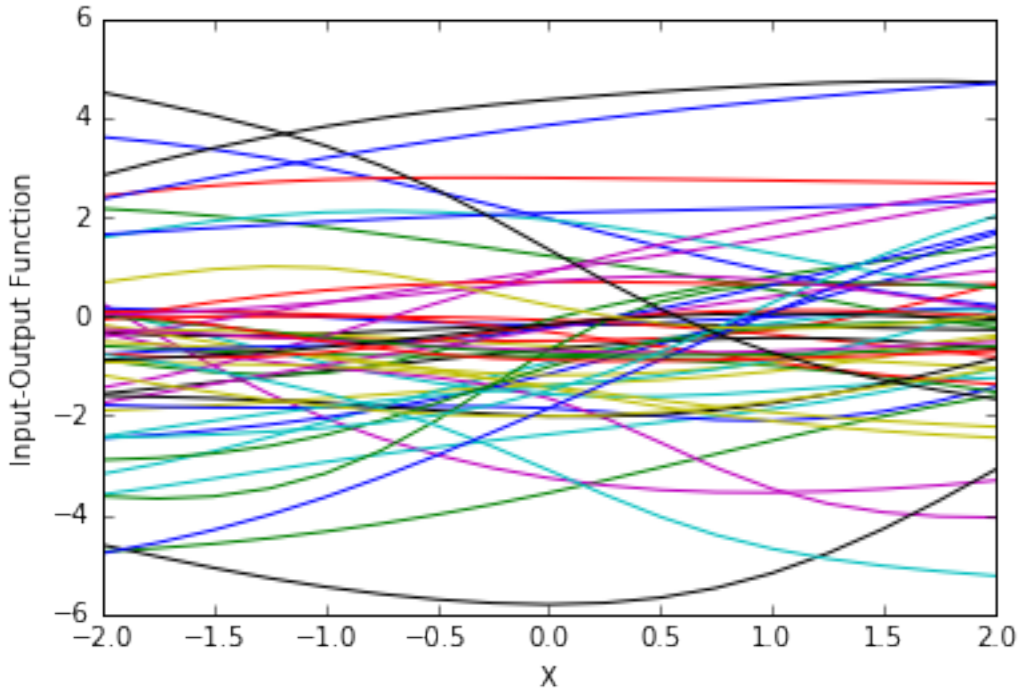
b)

In [62]:
```
res1 = []

for i in range(50):
    a = np.random.normal(0, 0.5, 10)
```

```
    w = np.random.normal(0, 1, 10)
    b = np.random.uniform(-2, 2, 10)


    f = []
    for i in range(10):
        f.append([np.sum(a[i]*x - b[i]) for x in X])
    s = np.dot(w.transpose(), np.tanh(f))
    res1.append(s)
```

In [63]:
```
results1 = np.asarray(res1)
plt.plot(np.arange(-2, 2.25, 0.25), results1.transpose())
plt.xlabel('X')
plt.ylabel('Input-Output Function')
```

Out[63]: <matplotlib.text.Text at 0x1127d6a0>



c)

In [64]:
```
g = -X
g
```

Out[64]:
```
array([ 2.  ,  1.75,  1.5 ,  1.25,  1.  ,  0.75,  0.5 ,  0.25, -0.  ,
       -0.25, -0.5 , -0.75, -1.  , -1.25, -1.5 , -1.75, -2.  ])
```

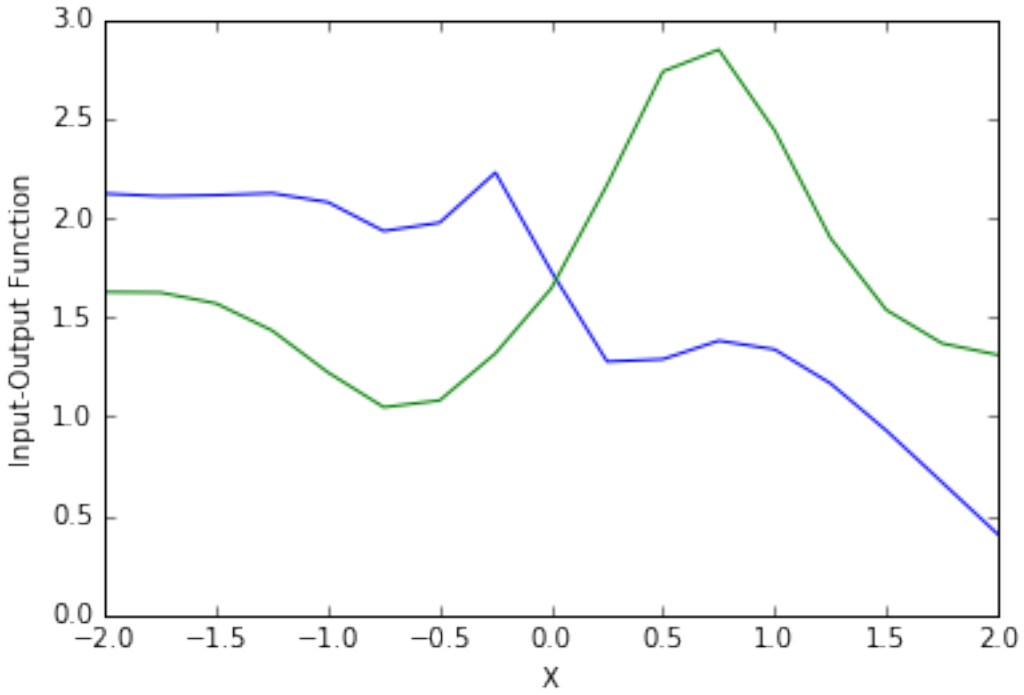
In [65]:
```
mse1 = 1./len(X)*np.sum(np.square(results-g), axis = 1)
mse1
```

```
Out[65]: array([ 11.79715565,   2.16213079,   7.95684112,   3.34294273,
                  1.50798872,  23.90636866,   1.26731989,   5.54277975,
                  8.57884526,   1.07579178,  11.15825783,   3.55974994,
                 17.48718309,   2.92934744,  42.46167948,  17.21713454,
                 27.95057481,   7.01685472,  11.41593127,   3.08234082,
                  3.42737618,   1.7241042 ,   4.80046993,   6.63447773,
                 16.92216568,   4.63248346,   6.29110819,   3.18194286,
                 13.38034831,   3.2825365 ,   5.02267868,  39.20816993,
                  2.42378558,   4.40563827,   5.47933913,   8.82602611,
                  3.04999822,   5.26093736,   2.90020625,   7.58848663,
                  6.49360444,   9.21416531,   8.78047812,   1.79195034,
                 14.1652363 ,  10.08060254,   4.82759514,   0.82552557,
                 13.80506469,   3.62022166])

In [66]: mse2 = 1./len(X)*np.sum(np.square(results1-g), axis = 1)
         mse2

Out[66]: array([  3.75991974,  16.57757954,   2.20932356,   2.11005022,
                  7.09975445,   4.43618827,   1.70102816,   1.52760176,
                  1.48601815,   1.04958046,   9.4795466 ,   5.09078917,
                  1.3176637 ,   3.13084797,   4.9926524 ,   4.51113969,
                  2.22041734,   6.87851688,   4.55392075,   2.79619756,
                 20.41410686,   6.80481723,   1.84567407,   0.75511993,
                  3.28793392,   4.30666512,   2.09887617,   4.75977881,
                  6.14419117,  10.25022137,   8.90457122,   5.44508653,
                  6.87380403,   2.45073442,  28.53000493,   3.9591072 ,
                  6.86252697,   2.5863297 ,   9.68650871,   1.59358288,
                  4.46818787,   2.47877218,  14.11879169,   1.38686771,
                  1.42740929,   8.14048015,   2.08263494,   0.803235  ,
                  2.77680878,  17.59544006])

In [67]: min(mse1)

Out[67]: 0.82552557363279622

In [68]: min(mse2)

Out[68]: 0.75511992973874498

In [69]: f1_best = np.argmin(mse1)
         f2_best = np.argmin(mse2)

         print("best f1 is ", f1_best, " and best f2 is ", f2_best)

('best f1 is ', 47, ' and best f2 is ', 23)


In [70]: f1 = results[36]
         f2 = results[30]

         combined = np.vstack((f1, f2)).T
         combined
```

```
Out[70]: array([[ 2.12157901,  1.62619313],
                [ 2.10868687,  1.62398672],
                [ 2.11362223,  1.56875772],
                [ 2.12319365,  1.43257664],
                [ 2.07834288,  1.22215202],
                [ 1.93306247,  1.04680245],
                [ 1.97517639,  1.08062846],
                [ 2.2272805 ,  1.31745148],
                [ 1.73080859,  1.6446705 ],
                [ 1.2762696 ,  2.16582423],
                [ 1.28836947,  2.73600117],
                [ 1.38154595,  2.84576673],
                [ 1.337534  ,  2.43927438],
                [ 1.16668709,  1.89807956],
                [ 0.9297704 ,  1.53625165],
                [ 0.66869864,  1.36800894],
                [ 0.40707482,  1.30992942]])
```

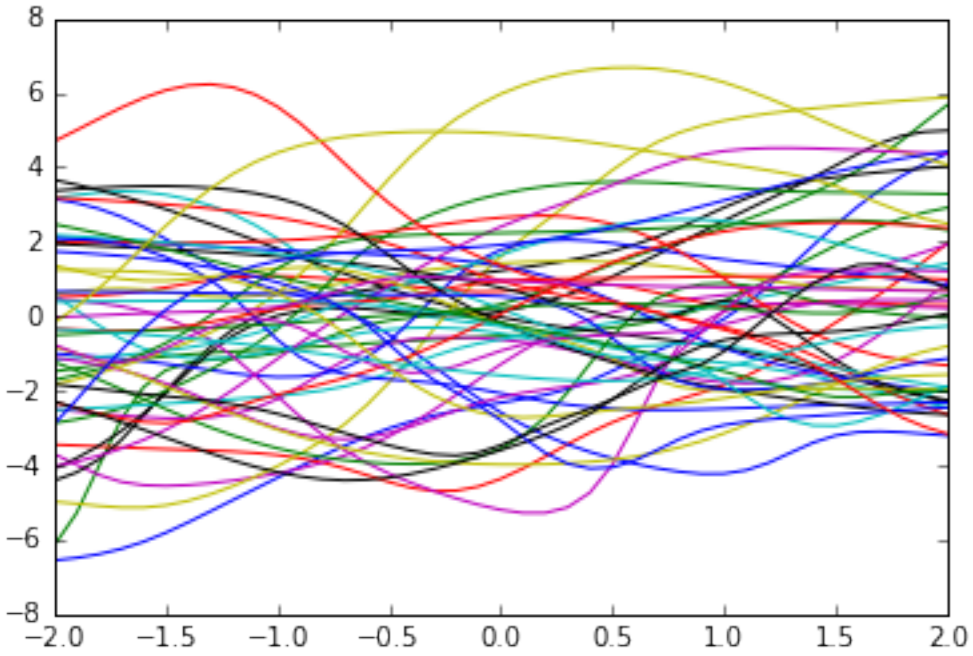
```
In [71]: plt.plot(np.arange(-2, 2.25, 0.25), combined)
         plt.xlabel('X')
         plt.ylabel('Input-Output Function')
```

```
Out[71]: <matplotlib.text.Text at 0x108efe80>
```

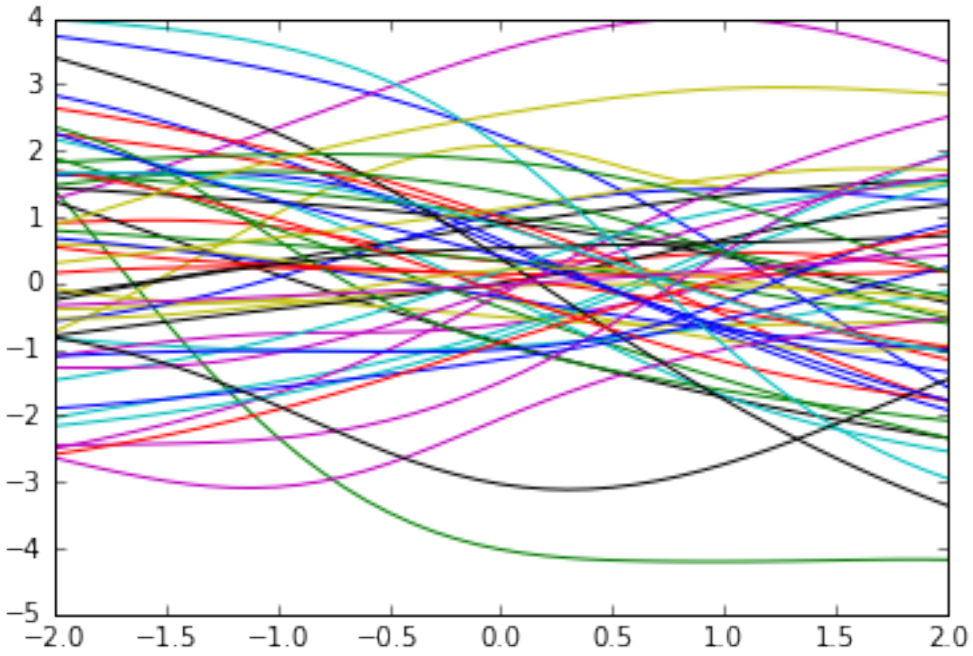


0.0.3 Exercise 2. - Onat's Version

```
In [52]: def calculate_result(w, a, b):
             results = []
             for i in range(-20, 21):
                 x = i / 10.0
                 sum = 0.0
                 for j in range(0, w.shape[0]):
                     sum += w[j, 0] * math.tanh(a[j, 0] * (x - b[j, 0]))
                 results.append(sum)
             return results

In [53]: import random
         w = np.matrix(np.arange(500).reshape((10, 50)), dtype = float)
         a = np.matrix(np.arange(500).reshape((10, 50)), dtype = float)
         b = np.matrix(np.arange(500).reshape((10, 50)), dtype = float)
         x_axis = []
         for i in range(-20, 21):
             x_axis.append(i / 10.0)

         for i in range(0, 50):
             for j in range(0, 10):
                 w[j, i] = random.gauss(0, 1)
                 a[j, i] = random.gauss(0, 2)
                 b[j, i] = random.uniform(-2.0, 2.0)

         for i in range(0, 50):
             results = calculate_result(w[:, i], a[:, i], b[:, i])
             plt.plot(x_axis, results)
         plt.show()
```



In [54]:
```
a_2 = np.matrix(np.arange(500).reshape((10, 50)), dtype=float)

for i in range(0, 50):
    for j in range(0, 10):
        a_2[j, i] = random.gauss(0, 0.5)
for i in range(0, 50):
    results_2 = calculate_result(w[:, i], a_2[:, i], b[:, i])
    plt.plot(x_axis, results_2)
plt.show()
```



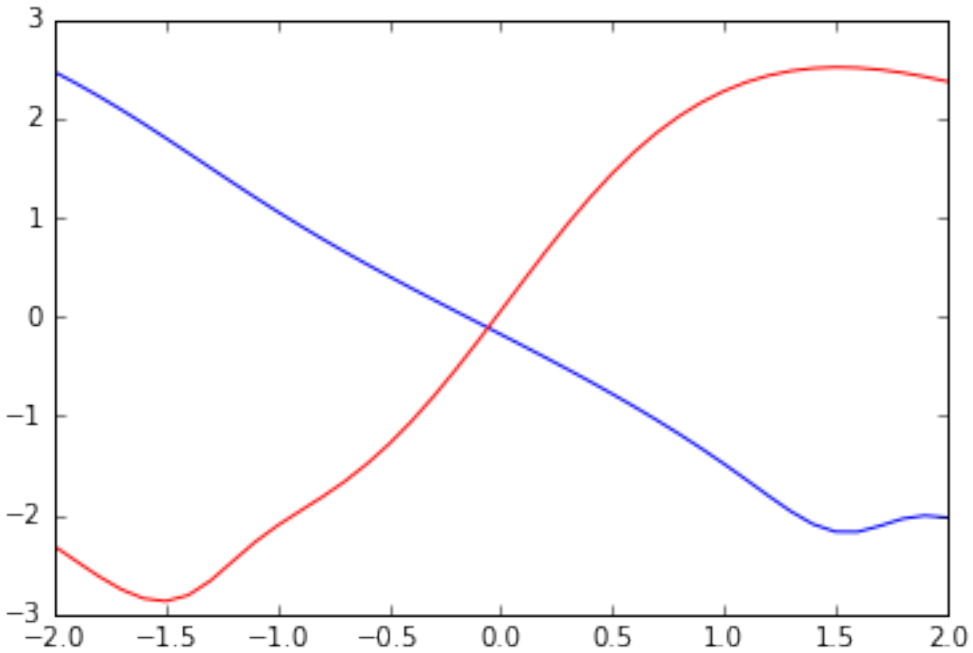
In [55]:
```
error_1 = 0.0
error_2 = 0.0
errors_array_1 = []
errors_array_2 = []
for i in range(0, 50):
    error_1 = 0.0
    results = calculate_result(w[:, i], a[:, i], b[:, i])
    for j in range(0, len(x_axis)):
        error_1 += math.pow(results[j] - -1 * x_axis[j], 2)
    error_1 /= len(x_axis)
    errors_array_1.append(error_1)

for i in range(0, 50):
    error_2 = 0.0
    results = calculate_result(w[:, i], a_2[:, i], b[:, i])
    for j in range(0, len(x_axis)):
        error_2 +=math.pow(results[j] - -1 * x_axis[j], 2)
    error_2 /= len(x_axis)
    errors_array_2.append(error_2)

print('With sigma 2')
print(errors_array_1)
```

```
With sigma 2
[22.234524017006652, 14.419203091251218, 15.178396848453922, 0.6991873088407118, 14
```

In [56]:
```
print('With sigma 0.5')
print(errors_array_2)
```

```
With sigma 0.5
[3.4268944172719005, 9.959892232236971, 1.2532305042935197, 2.598689603368287, 2.94
```

In [57]:
```
best_result_for_group_1 = np.argmin(errors_array_1)
a_best = a[:, best_result_for_group_1]
b_best = b[:, best_result_for_group_1]
w_best = w[:, best_result_for_group_1]
res_1 = calculate_result(w_best, a_best, b_best)
best_result_for_group_2 = np.argmin(errors_array_2)
a_best = a[:, best_result_for_group_2]
b_best = b[:, best_result_for_group_2]
w_best = w[:, best_result_for_group_2]
res_2 = calculate_result(w_best, a_best, b_best)
plt.plot(x_axis, res_1, c = 'b')
plt.plot(x_axis, res_2, c = 'r')
plt.show()
```



In [ ]: